Computer Vision System Toolbox™

Reference

R2013a
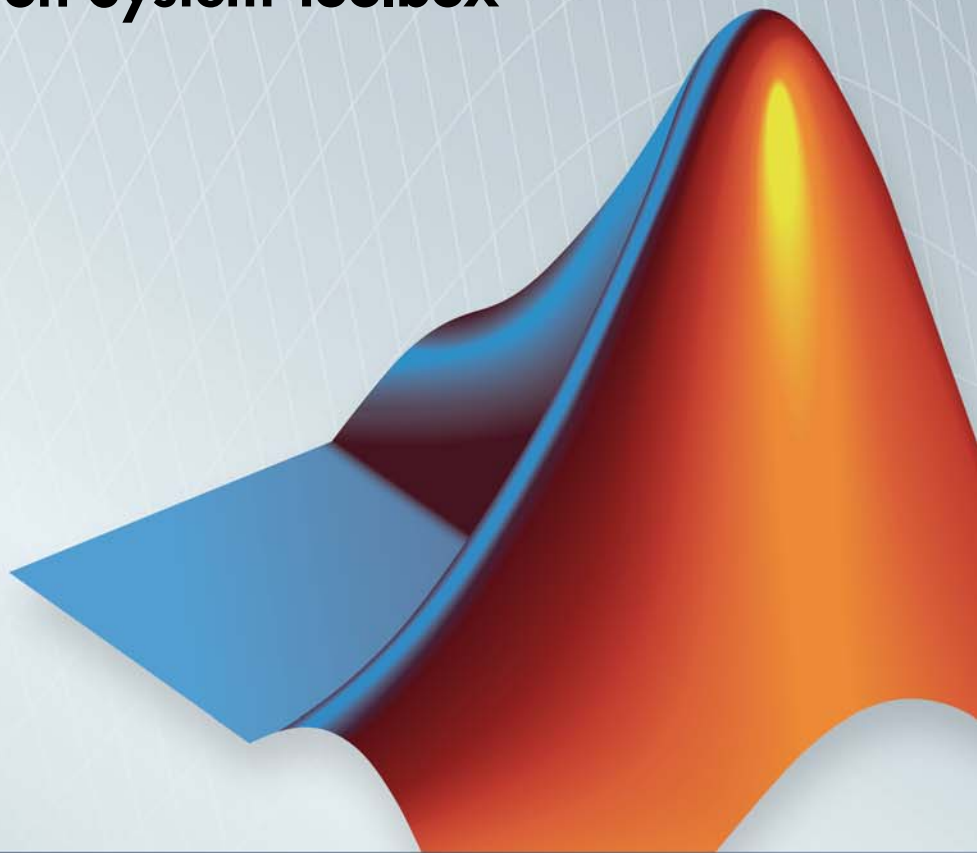
MATLAB®

MathWorks®

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Computer Vision System Toolbox™ Reference

© COPYRIGHT 2000–2013 by MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

Contents

Blocks — Alphabetical List

2-D Autocorrelation

Purpose   Compute 2-D autocorrelation of input matrix

Library   Statistics

visionstatistics

Description   The 2-D Autocorrelation block computes the two-dimensional autocorrelation of the input matrix. Assume that input matrix A has dimensions (Ma, Na). The equation for the two-dimensional discrete autocorrelation is
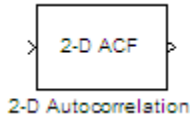
$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) \cdot conj(A(m+i,n+j))$$

where $0 \leq i < 2Ma - 1$ and $0 \leq j < 2Na - 1$.

The output of this block has dimensions $(2Ma - 1, 2Na - 1)$.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values or a scalar, vector, or matrix that represents one plane of the RGB video stream	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
Output	Autocorrelation of the input matrix	Same as Input port	Yes

If the data type of the input is floating point, the output of the block has the same data type.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D Autocorrelation block for fixed-point signals.
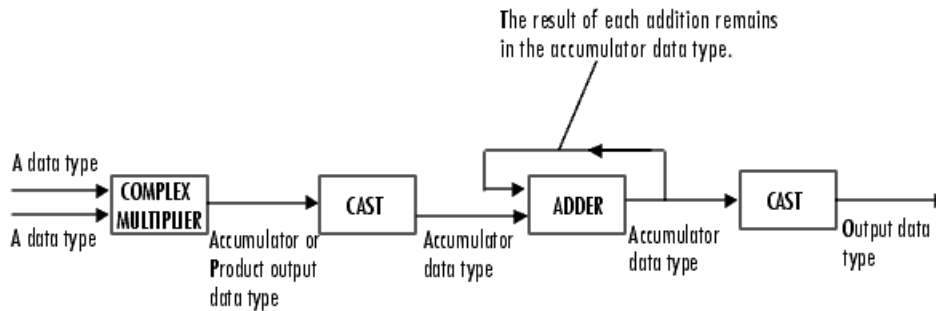
You can set the product output, accumulator, and output data types in the block mask as discussed in "Dialog Box" on page 1-4.

The output of the multiplier is in the product output data type if at least one of the inputs to the multiplier is real. If both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types".

2-D Autocorrelation

**Dialog
Box**

The **Main** pane of the 2-D Autocorrelation dialog box appears as shown in the following figure.

The **Data Types** pane of the 2-D Autocorrelation dialog box appears as shown in the following figure.

Function Block Parameters: 2-D Autocorrelation

2-D Autocorrelation

Compute 2-D autocorrelation of input matrix.

| Main | Data Types |

Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.

Fixed-point operational parameters

Rounding mode: Floor   Overflow mode: Wrap

Fixed-point data types

Data Type

Product output   Same as input

Accumulator   Same as product output

Output   Same as input

☐ Lock data type settings against changes by the fixed-point tools

OK   Cancel   Help   Apply

Rounding mode
> Select the "Rounding Modes" for fixed-point operations.

Overflow mode
> Select the Overflow mode for fixed-point operations.

Product output
> Specify the product output data type. See "Fixed-Point Data Types" on page 1-3 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

> - When you select `Same as input`, these characteristics match those of the input to the block.

> - When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

> - When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox™ software is 0.

Accumulator
> Use this parameter to specify how to designate the accumulator word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-3 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block. The accumulator data type is only used when both inputs to the multiplier are complex.

> - When you select `Same as product output`, these characteristics match those of the product output.

> - When you select `Same as input`, these characteristics match those of the input to the block.

> - When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

> - When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software is `0`.

Output

Choose how to specify the output word length and fraction length.

- When you select Same as input, these characteristics match those of the input to the block.

- When you select Binary point scaling, you can enter the word length and the fraction length of the output, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see fxptdlg, a reference page on the Fixed-Point Tool in the Simulink® documentation.

See Also

2-D Correlation	Computer Vision System Toolbox
2-D Histogram	Computer Vision System Toolbox
2-D Mean	Computer Vision System Toolbox
2-D Median	Computer Vision System Toolbox
2-D Standard Deviation	Computer Vision System Toolbox
2-D Variance	Computer Vision System Toolbox
2-D Maximum	Computer Vision System Toolbox
2-D Minimum	Computer Vision System Toolbox

2-D Convolution

Purpose   Compute 2-D discrete convolution of two input matrices

Library   Filtering

visionfilter

Description
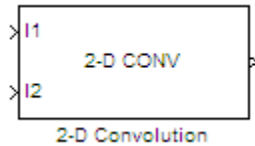
The 2-D Convolution block computes the two-dimensional convolution of two input matrices. Assume that matrix A has dimensions (Ma, Na) and matrix B has dimensions (Mb, Nb). When the block calculates the full output size, the equation for the 2-D discrete convolution is

$$C(i, j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) * B(i-m, j-n)$$

where $0 \le i < Ma + Mb - 1$ and $0 \le j < Na + Nb - 1$.

Port	Input/Output	Supported Data Types	Complex Values Supported
I1	Matrix of intensity values or a matrix that represents one plane of the RGB video stream	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
I2	Matrix of intensity values or a matrix that	Same as I1 port	Yes

Port	Input/Output	Supported Data Types	Complex Values Supported
	represents one plane of the RGB video stream		
Output	Convolution of the input matrices	Same as I1 port	Yes

If the data type of the input is floating point, the output of the block has the same data type.

The dimensions of the output are dictated by the **Output size** parameter. Assume that the input at port I1 has dimensions (Ma, Na) and the input at port I2 has dimensions (Mb, Nb). If, for the **Output size** parameter, you choose Full, the output is the full two-dimensional convolution with dimensions ($Ma+Mb$-1, $Na+Nb$-1). If, for the **Output size** parameter, you choose Same as input port I1, the output is the central part of the convolution with the same dimensions as the input at port I1. If, for the **Output size** parameter, you choose Valid, the output is only those parts of the convolution that are computed without the zero-padded edges of any input. This output has dimensions (Ma-Mb+1, Na-Nb+1). However, if all(size(I1)<size(I2)), the block errors out.

If you select the **Output normalized convolution** check box, the block's output is divided by sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2))), where I1p is the portion of the I1 matrix that aligns with the I2 matrix. See "Example 2" on page 1-12 for more information.

Note When you select the **Output normalized convolution** check box, the block input cannot be fixed point.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D Convolution block for fixed-point signals.

2-D Convolution
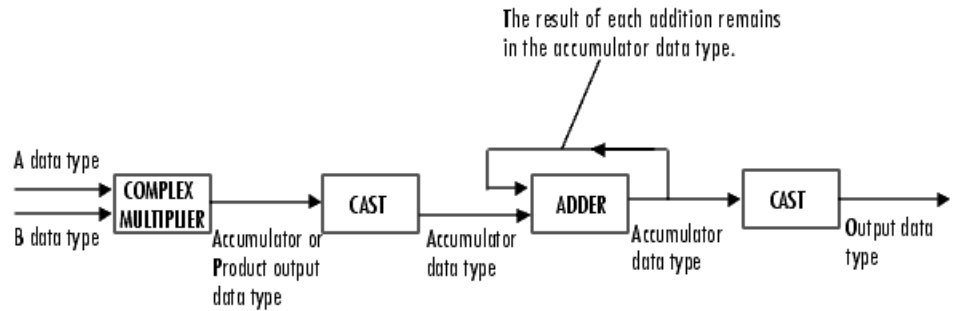
The result of each addition remains in the accumulator data type.

A data type → COMPLEX MULTIPLIER → (Accumulator or Product output data type) → CAST → (Accumulator data type) → ADDER → (Accumulator data type) → CAST → (Output data type)

B data type →

You can set the product output, accumulator, and output data types in the block mask as discussed in "Dialog Box" on page 1-15.

The output of the multiplier is in the product output data type if at least one of the inputs to the multiplier is real. If both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types".

Examples   **Example 1**

Suppose I1, the first input matrix, has dimensions (4,3) and I2, the second input matrix, has dimensions (2,2). If, for the **Output size** parameter, you choose Full, the block uses the following equations to determine the number of rows and columns of the output matrix:

$$C_{full_{rows}} = I1_{rows} + I2_{rows} - 1 = 5$$

$$C_{full_{columns}} = I1_{columns} + I2_{columns} - 1 = 4$$

The resulting matrix is

$$C_{\text{full}} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \\ c_{40} & c_{41} & c_{42} & c_{43} \end{bmatrix}$$

If, for the **Output size** parameter, you choose Same as input port I1, the output is the central part of C_{full} with the same dimensions as the input at port I1, (4,3). However, since a 4-by-3 matrix cannot be extracted from the exact center of C_{full}, the block leaves more rows and columns on the top and left side of the C_{full} matrix and outputs:

$$C_{\text{same}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \\ c_{41} & c_{42} & c_{43} \end{bmatrix}$$

If, for the **Output size** parameter, you choose Valid, the block uses the following equations to determine the number of rows and columns of the output matrix:

$$C_{\text{valid}_{\text{rows}}} = I1_{\text{rows}} - I2_{\text{rows}} + 1 = 3$$

$$C_{\text{valid}_{\text{columns}}} = I1_{\text{columns}} - I2_{\text{columns}} + 1 = 2$$

In this case, it is always possible to extract the exact center of C_{full}. Therefore, the block outputs

$$C_{\text{full}} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

Example 2

In convolution, the value of an output element is computed as a weighted sum of neighboring elements.

For example, suppose the first input matrix represents an image and is defined as

```
I1 = [17   24    1    8   15
      23    5    7   14   16
       4    6   13   20   22
      10   12   19   21    3
      11   18   25    2    9]
```

The second input matrix also represents an image and is defined as

```
I2 = [8    1    6
      3    5    7
      4    9    2]
```

The following figure shows how to compute the (1,1) output element (zero-based indexing) using these steps:

1 Rotate the second input matrix, I2, 180 degrees about its center element.

2 Slide the center element of I2 so that it lies on top of the (0,0) element of I1.

3 Multiply each element of the rotated I2 matrix by the element of I1 underneath.

4 Sum the individual products from step 3.

Hence the (1,1) output element is

$$0 \cdot 2 + 0 \cdot 9 + 0 \cdot 4 + 0 \cdot 7 + 17 \cdot 5 + 24 \cdot 3 + 0 \cdot 6 + 23 \cdot 1 + 5 \cdot 8 = 220 \,.$$

2-D Convolution

Values of rotated I2 matrix

2	9	4
7	5	3
6	1	8

Alignment of center element of I2

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Image pixel values

Alignment of I2 matrix
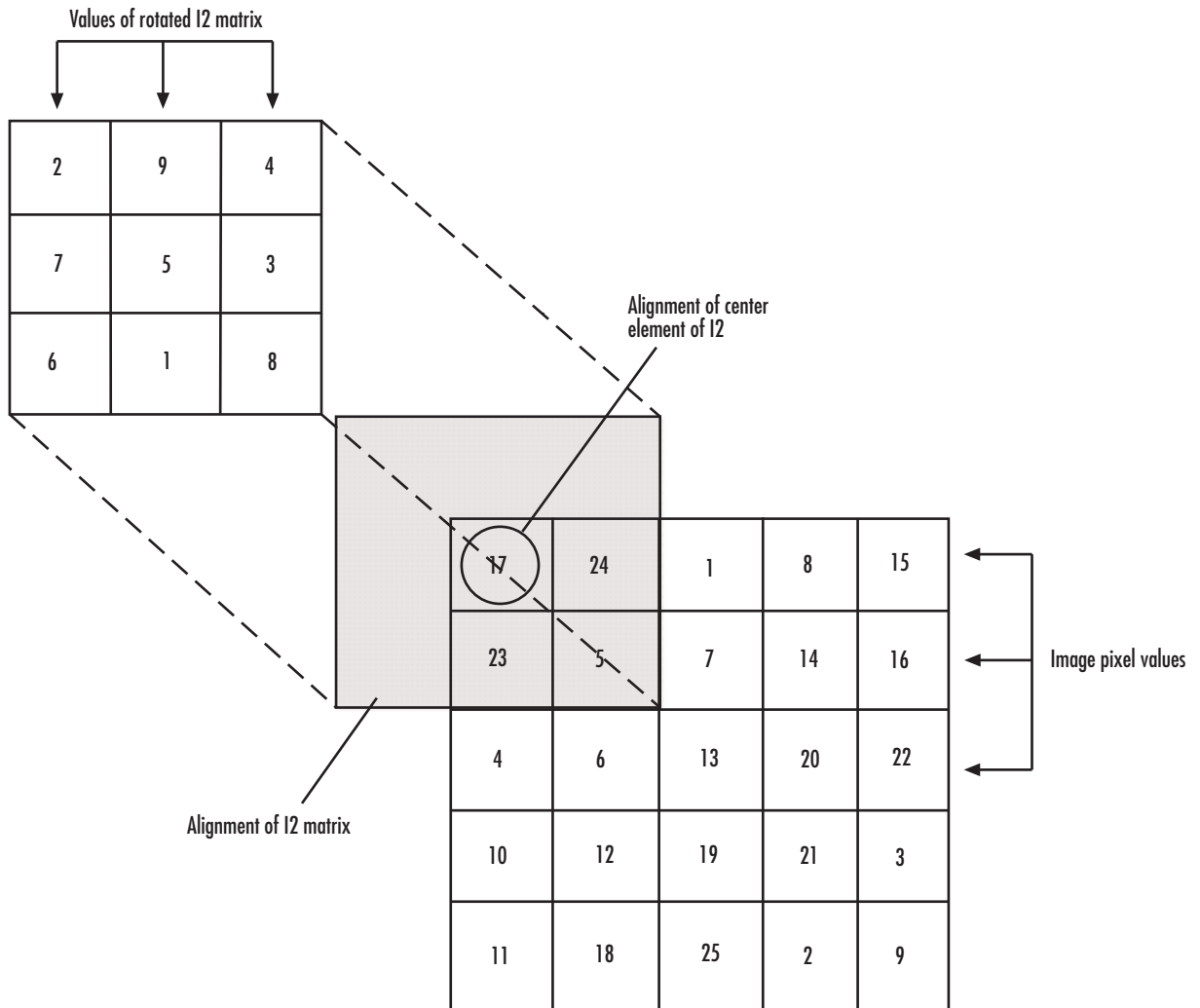
Computing the (1,1) Output of Convolution

The normalized convolution of the (1,1) output element is
`220/sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2)))` = 0.3459, where
`I1p = [0 0 0; 0 17 24; 0 23 5]`.

**Dialog
Box**

The **Main** pane of the 2-D Convolution dialog box appears as shown in
the following figure.

Function Block Parameters: 2-D Convolution

2-D Convolution

Performs two-dimensional convolution on two inputs.

Use the Output size parameter to specify the dimensions of the output. Assume that the input at port I1 has
dimensions (Ma, Na) and the input at port I2 has dimensions (Mb, Nb). If you choose Full, the output is the
full two-dimensional convolution with dimensions (Ma+Mb-1, Na+Nb-1). If you choose Same as input port
I1, the output is the central part of the convolution with the same dimensions as the input at port I1. If you
choose Valid, the output is only those parts of the convolution that are computed without the zero-padded
edges of any input. This output has dimensions (Ma-Mb+1, Na-Nb+1).

You can normalize the output only when the input is floating point.

Main   Data Types

Parameters

Output size: Full

☐ Normalized output

OK   Cancel   Help   Apply

2-D Convolution

Output size

> This parameter controls the size of the output scalar, vector, or matrix produced as a result of the convolution between the two inputs. If you choose `Full`, the output has dimensions ($Ma+Mb$-1, $Na+Nb$-1). If you choose `Same as input port I1`, the output has the same dimensions as the input at port I1. If you choose `Valid`, output has dimensions ($Ma-Mb$+1, $Na-Nb$+1).

Output normalized convolution

> If you select this check box, the block's output is normalized.

The **Data Types** pane of the 2-D Convolution dialog box appears as shown in the following figure.

Rounding mode
    Select the rounding mode for fixed-point operations.

Overflow mode
    Select the Overflow mode for fixed-point operations.

Product output

Use this parameter to specify how to designate the product output word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-9 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in theComputer Vision System Toolbox software is 0.

The Product Output inherits its sign according to the inputs. If either or both input **I1** and **I2** are signed, the Product Output will be signed. Otherwise, the Product Output is unsigned. The following table shows all cases.

Sign of Input I1	Sign of Input I2	Sign of Product Output
unsigned	unsigned	unsigned
unsigned	signed	signed
signed	unsigned	signed
signed	signed	signed

Accumulator

Use this parameter to specify how to designate the accumulator word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-9 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block. The accumulator data type is only used when both inputs to the multiplier are complex:

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also   2-D FIR Filter   Computer Vision System Toolbox
                                            software

2-D Correlation

Purpose Compute 2-D cross-correlation of two input matrices

Library Statistics

visionstatistics
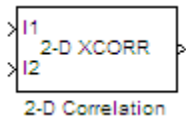
Description The 2-D Correlation block computes the two-dimensional cross-correlation of two input matrices. Assume that matrix A has dimensions (*Ma*, *Na*) and matrix B has dimensions (*Mb*, *Nb*). When the block calculates the full output size, the equation for the two-dimensional discrete cross-correlation is

$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) \cdot conj(B(m+i,n+j))$$

where $0 \le i < Ma + Mb - 1$ and $0 \le j < Na + Nb - 1$.

Port	Input/Output	Supported Data Types	Complex Values Supported
I1	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
I2	Scalar, vector, or matrix of intensity values or a scalar, vector, or matrix that represents one plane of the RGB video stream	Same as I1 port	Yes
Output	Convolution of the input matrices	Same as I1 port	Yes

If the data type of the input is floating point, the output of the block is the same data type.

The dimensions of the output are dictated by the **Output size** parameter and the sizes of the inputs at ports I1 and I2. For example, assume that the input at port I1 has dimensions (*Ma*, *Na*) and the input at port I2 has dimensions (*Mb*, *Nb*). If, for the **Output size** parameter, you choose Full, the output is the full two-dimensional cross-correlation with dimensions (*Ma*+*Mb*-1, *Na*+*Nb*-1). If, for the **Output size** parameter, you choose Same as input port I1, the output is the central part of the cross-correlation with the same dimensions as the input at port I1. If, for the **Output size** parameter, you choose Valid, the output is only those parts of the cross-correlation that are computed without the zero-padded edges of any input. This output has dimensions (Ma-Mb+1, Na-Nb+1). However, if all(size(I1)<size(I2)), the block errors out.

If you select the **Normalized output** check box, the block's output is divided by sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2))), where I1p is the portion of the I1 matrix that aligns with the I2 matrix. See "Example 2" on page 1-24 for more information.

Note When you select the **Normalized output** check box, the block input cannot be fixed point.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D Correlation block for fixed-point signals.

2-D Correlation
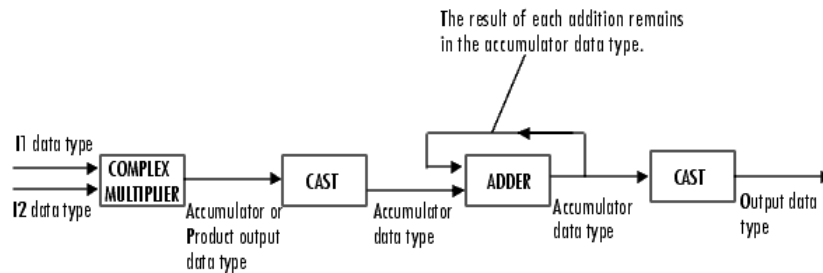
The result of each addition remains in the accumulator data type.

You can set the product output, accumulator, and output data types in the block mask as discussed in "Dialog Box" on page 1-27.

The output of the multiplier is in the product output data type if at least one of the inputs to the multiplier is real. If both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types".

Examples

Example 1

Suppose I1, the first input matrix, has dimensions (4,3). I2, the second input matrix, has dimensions (2,2). If, for the **Output size** parameter, you choose Full, the block uses the following equations to determine the number of rows and columns of the output matrix:

$$C_{\text{full}_{\text{rows}}} = I1_{\text{rows}} + I2_{\text{rows}} - 1 = 4 + 2 - 1 = 5$$

$$C_{\text{full}_{\text{columns}}} = I1_{\text{columns}} + I2_{\text{columns}} - 1 = 3 + 2 - 1 = 4$$

The resulting matrix is

$$C_{\text{full}} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \\ c_{40} & c_{41} & c_{42} & c_{43} \end{bmatrix}$$

If, for the **Output size** parameter, you choose Same as input port I1, the output is the central part of C_{full} with the same dimensions as the input at port I1, (4,3). However, since a 4-by-3 matrix cannot be extracted from the exact center of C_{full}, the block leaves more rows and columns on the top and left side of the C_{full} matrix and outputs:

$$C_{\text{same}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \\ c_{41} & c_{42} & c_{43} \end{bmatrix}$$

If, for the **Output size** parameter, you choose Valid, the block uses the following equations to determine the number of rows and columns of the output matrix:

$$C_{\text{valid}_{\text{rows}}} = I1_{\text{rows}} - I2_{\text{rows}} + 1 = 3$$

$$C_{\text{valid}_{\text{columns}}} = I1_{\text{columns}} - I2_{\text{columns}} + 1 = 2$$

In this case, it is always possible to extract the exact center of C_{full}. Therefore, the block outputs

$$C_{\text{full}} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

Example 2

In cross-correlation, the value of an output element is computed as a weighted sum of neighboring elements.

For example, suppose the first input matrix represents an image and is defined as

```
I1 = [17   24    1    8   15
      23    5    7   14   16
       4    6   13   20   22
      10   12   19   21    3
      11   18   25    2    9]
```

The second input matrix also represents an image and is defined as

```
I2 = [8    1    6
      3    5    7
      4    9    2]
```

The following figure shows how to compute the (2,4) output element (zero-based indexing) using these steps:

1 Slide the center element of I2 so that lies on top of the (1,3) element of I1.

2 Multiply each weight in I2 by the element of I1 underneath.

3 Sum the individual products from step 2.

The (2,4) output element from the cross-correlation is
$1 \cdot 8 + 8 \cdot 1 + 15 \cdot 6 + 7 \cdot 3 + 14 \cdot 5 + 16 \cdot 7 + 13 \cdot 4 + 20 \cdot 9 + 22 \cdot 2 = 585$.

2-D Correlation

Values of I2 matrix

8	1	6
3	5	7
4	9	2

Alignment of I2 matrix

Image pixel values

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Alignment of center element of I2
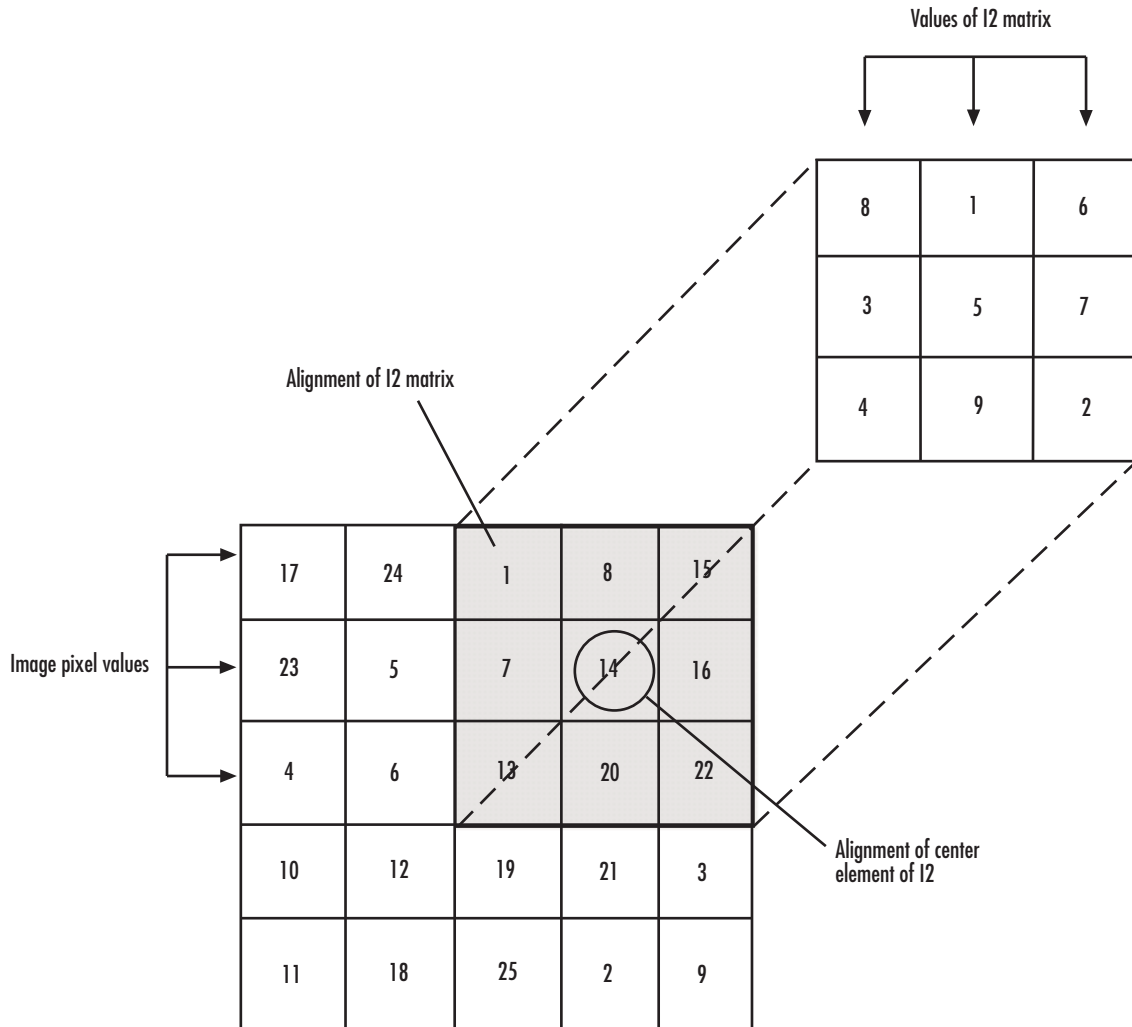
Computing the (2,4) Output of Cross-Correlation

The normalized cross-correlation of the (2,4) output element is
`585/sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2))) = 0.8070`, where
`I1p = [1 8 15; 7 14 16; 13 20 22]`.

Dialog Box

The **Main** pane of the 2-D Correlation dialog box appears as shown in the following figure.

Output size

This parameter controls the size of the output scalar, vector, or matrix produced as a result of the cross-correlation between the two inputs. If you choose `Full`, the output has dimensions (*Ma+Mb*-1, *Na+Nb*-1). If you choose `Same as input port I1`, the output has the same dimensions as the input at port I1. If you choose `Valid`, output has dimensions (*Ma-Mb*+1, *Na-Nb*+1).

Normalized output

If you select this check box, the block's output is normalized.

The **Data Types** pane of the 2-D Correlation dialog box appears as shown in the following figure.

2-D Correlation

Function Block Parameters: 2-D Correlation

2-D Correlation

Performs two-dimensional cross-correlation between two inputs.

Use the Output size parameter to specify the dimensions of the output. Assume that the input at port I1 has dimensions (Ma, Na) and the input at port I2 has dimensions (Mb, Nb). If you choose Full, the output is the full two-dimensional cross-correlation with dimensions (Ma+Mb-1, Na+Nb-1). If you choose Same as input port I1, the output is the central part of the cross-correlation with the same dimensions as the input at port I1. If you choose Valid, the output is only those parts of the cross-correlation that are computed without the zero-padded edges of any input. This output has dimensions (Ma-Mb+1, Na-Nb+1).

You can normalize the output only when the input is floating point.

| Main | Data Types |

Settings on this pane only apply when block inputs are fixed-point signals.

Fixed-point operational parameters

Rounding mode: Floor   Overflow mode: Wrap

Fixed-point data types

Mode

Product output   Same as first input

Accumulator   Same as product output

Output   Same as first input

☐ Lock data type settings against changes by the fixed-point tools

OK   Cancel   Help   Apply

Rounding mode

Select the "Rounding Modes" for fixed-point operations.

Overflow mode

Select the Overflow mode for fixed-point operations.

Product output

Specify the product output data type. See "Fixed-Point Data Types" on page 1-21 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select Same as first input, these characteristics match those of the first input to the block.

- When you select Binary point scaling, you can enter the word length and the fraction length of the product output, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

The Product Output inherits its sign according to the inputs. If either or both input **I1** and **I2** are signed, the Product Output will be signed. Otherwise, the Product Output is unsigned. The table below show all cases.

Sign of Input I1	Sign of Input I2	Sign of Product Output
unsigned	unsigned	unsigned
unsigned	signed	signed
signed	unsigned	signed
signed	signed	signed

Accumulator

Use this parameter to specify how to designate the accumulator word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-21 and"Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block. The accumulator data type is only used when both inputs to the multiplier are complex:

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also

2-D Autocorrelation	Computer Vision System Toolbox
2-D Histogram	Computer Vision System Toolbox
2-D Mean	Computer Vision System Toolbox
2-D Median	Computer Vision System Toolbox
2-D Standard Deviation	Computer Vision System Toolbox
2-D Variance	Computer Vision System Toolbox
2-D Maximum	Computer Vision System Toolbox
2-D Minimum	Computer Vision System Toolbox

2-D DCT

Purpose   Compute 2-D discrete cosine transform (DCT)

Library   Transforms

visiontransforms

Description   The 2-D DCT block calculates the two-dimensional discrete cosine transform of the input signal. The equation for the two-dimensional DCT is

$$F(m,n) = \frac{2}{\sqrt{MN}} C(m)C(n) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)m\pi}{2M} \cos\frac{(2y+1)n\pi}{2N}$$

where $C(m), C(n) = 1/\sqrt{2}$ for $m, n = 0$ and $C(m), C(n) = 1$ otherwise.

The number of rows and columns of the input signal must be powers of two. The output of this block has dimensions the same dimensions as the input.

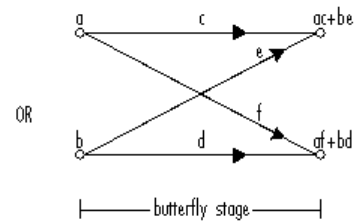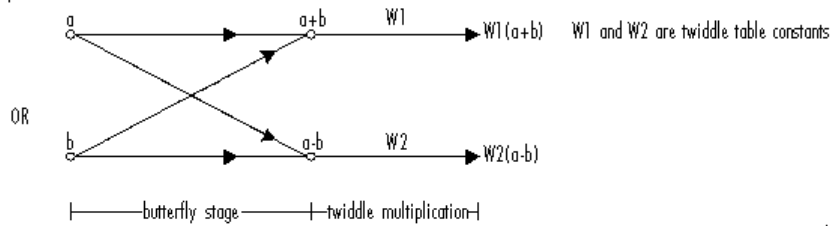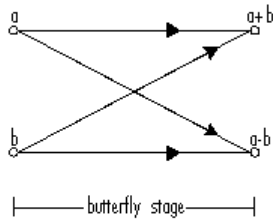| Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Output	2-D DCT of the input	Same as Input port	No

If the data type of the input signal is floating point, the output of the block is the same data type.

Use the **Sine and cosine computation** parameter to specify how the block computes the sine and cosine terms in the DCT algorithm. If you select `Trigonometric fcn`, the block computes the sine and cosine values during the simulation. If you select `Table lookup`, the block computes and stores the trigonometric values before the simulation starts. In this case, the block requires extra memory.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D DCT block for fixed-point signals. Inputs are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.

W1 and W2 are twiddle table constants

Butterfly Stage Data Types
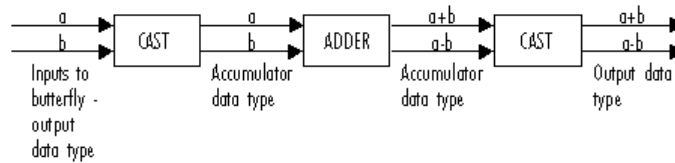
Twiddle Multiplication Data Types
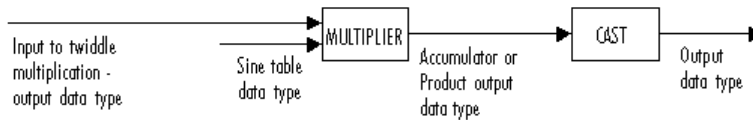
The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types". You can set the sine table, product output, accumulator, and output data types in the block mask as discussed in the next section.

Dialog Box

The **Main** pane of the 2-D DCT dialog box appears as shown in the following figure.
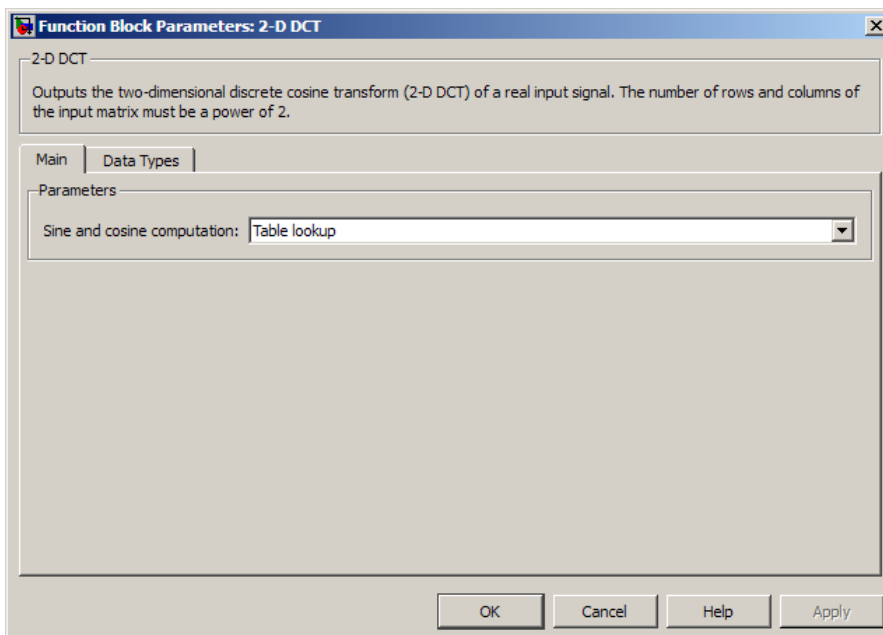
Sine and cosine computation

Specify how the block computes the sine and cosine terms in the DCT algorithm. If you select Trigonometric fcn, the block computes the sine and cosine values during the simulation. If you select Table lookup, the block computes and stores the

trigonometric values before the simulation starts. In this case, the block requires extra memory.

The **Data Types** pane of the 2-D DCT dialog box appears as shown in the following figure.
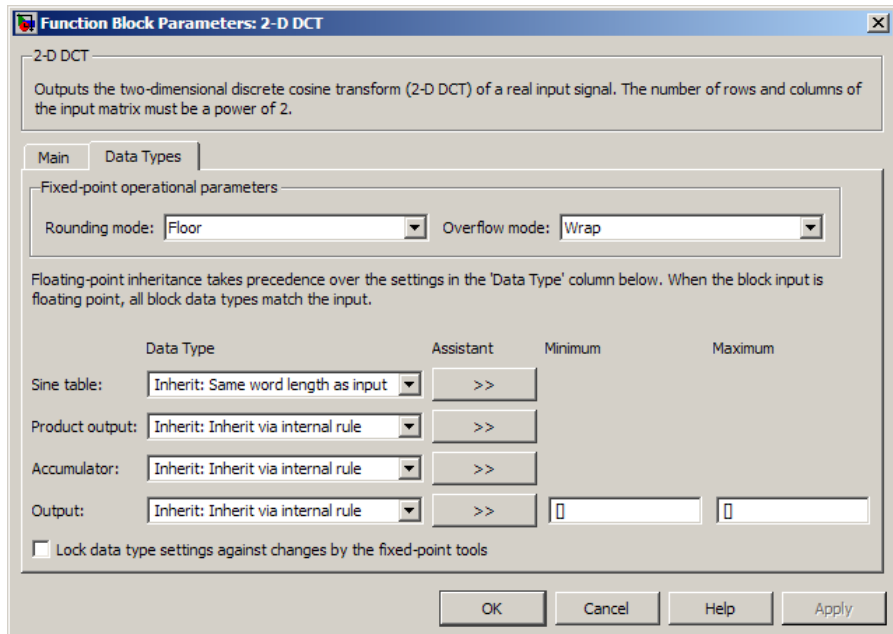
Rounding mode

Select the "Rounding Modes" for fixed-point operations. The sine table values do not obey this parameter; they always round to `Nearest`.

Overflow mode

Select the Overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

> Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:
>
> - A rule that inherits a data type, for example, `Inherit: Same word length as input`
>
> - An expression that evaluates to a valid data type, for example, `fixdt(1,16)`
>
> The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

> Specify the product output data type. See "Fixed-Point Data Types" on page 1-33 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:
>
> - A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
>
> - An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`
>
> Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.
>
> See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

> Specify the accumulator data type. See "Fixed-Point Data Types" on page 1-33 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:
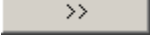
- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button � ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 1-33 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{ideal\ output} = WL_{input} + floor(\log_2(DCT\ length - 1)) + 1$$

  $$FL_{ideal\ output} = FL_{input}$$

  Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button ⌐ ›› ⌐ to
display the **Data Type Assistant**, which helps you set the
Output data type parameter.

See "Specify Data Types Using Data Type Assistant" for more
information.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you
specify in this block mask from being overridden by the
autoscaling tool in the Fixed-Point Tool. For more information,
see `fxptdlg`, a reference page on the Fixed-Point Tool in the
Simulink documentation.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from
overriding the data types you specify on the block mask. For more
information, see `fxptdlg`, a reference page on the Fixed-Point
Tool in the Simulink documentation.

References   [1] Chen, W.H, C.H. Smith, and S.C. Fralick, "A fast computational
algorithm for the discrete cosine transform," *IEEE Trans. Commun.*,
vol. COM-25, pp. 1004-1009. 1977.

[2] Wang, Z. "Fast algorithms for the discrete W transform and for
the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal
Processing*, vol. ASSP-32, pp. 803-816, Aug. 1984.

See Also

2-D IDCT	Computer Vision System Toolbox software
2-D FFT	Computer Vision System Toolbox software
2-D IFFT	Computer Vision System Toolbox software

2-D FFT

Purpose   Compute two-dimensional fast Fourier transform of input

Library   Transforms

              `visiontransforms`
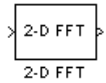
Description   The 2-D FFT block computes the fast Fourier transform (FFT). The block does the computation of a two-dimensional M-by-N input matrix in two steps. First it computes the one-dimensional FFT along one dimension (row or column). Then it computes the FFT of the output of the first step along the other dimension (column or row).

The output of the 2-D FFT block is equivalent to the MATLAB® fft2 function:

```
y = fft2(A) % Equivalent MATLAB code
```

Computing the FFT of each dimension of the input matrix is equivalent to calculating the two-dimensional discrete Fourier transform (DFT), which is defined by the following equation:

$$F(m,n) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j\frac{2\pi mx}{M}}e^{-j\frac{2\pi ny}{N}}$$

where $0 \le m \le M-1$ and $0 \le n \le N-1$.

The output of this block has the same dimensions as the input. If the input signal has a floating-point data type, the data type of the output signal uses the same floating-point data type. Otherwise, the output can be any fixed-point data type. The block computes scaled and unscaled versions of the FFT.

The input to this block can be floating-point or fixed-point, real or complex, and conjugate symmetric. The block uses one of two possible FFT implementations. You can select an implementation based on the FFTW library [1], [2], or an implementation based on a collection of Radix-2 algorithms. You can select Auto to allow the block to choose the implementation.

Port Description

Port	Description	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
Output	2-D FFT of the input	Same as Input port	Yes

FFTW Implementation

The FFTW implementation provides an optimized FFT calculation including support for power-of-two and non-power-of-two transform

lengths in both simulation and code generation. Generated code using the FFTW implementation will be restricted to those computers which are capable of running MATLAB. The input data type must be floating-point.

Radix-2 Implementation

The Radix-2 implementation supports bit-reversed processing, fixed or floating-point data, and allows the block to provide portable C-code generation using the Simulink Coder. The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or if possible choose the FFTW implementation.

With Radix-2 selected, the block implements one or more of the following algorithms:

- Butterfly operation

- Double-signal algorithm

- Half-length algorithm

- Radix-2 decimation-in-time (DIT) algorithm

- Radix-2 decimation-in-frequency (DIF) algorithm

Radix-2 Algorithms for Real or Complex Input Complexity Floating-Point Signals

Other Parameter Settings	Algorithms Used for IFFT Computation
☐ Input is in bit-reversed order ☐ Input is conjugate symmetric	Butterfly operation and radix-2 DIT
☑ Input is in bit-reversed order ☐ Input is conjugate symmetric	Radix-2 DIF

Other Parameter Settings	Algorithms Used for IFFT Computation
☐ Input is in bit-reversed order ☑ Input is conjugate symmetric	Butterfly operation and radix-2 DIT in conjunction with the half-length and double-signal algorithms
☑ Input is in bit-reversed order ☑ Input is conjugate symmetric	Radix-2 DIF in conjunction with the half-length and double-signal algorithms

Radix-2 Algorithms for Real or Complex Input Complexity Fixed-Point Signals

Other Parameter Settings	Algorithms Used for IFFT Computation
☐ Input is in bit-reversed order ☐ Input is conjugate symmetric	Butterfly operation and radix-2 DIT
☑ Input is in bit-reversed order ☐ Input is conjugate symmetric	Radix-2 DIF

Note The **Input is conjugate symmetric** parameter cannot be used for fixed-point signals.

Radix-2 Optimization for the Table of Trigonometric Values

In certain situations, the block's Radix–2 algorithm computes all the possible trigonometric values of the twiddle factor

$$e^{j\frac{2\pi k}{K}}$$

where K is the greater value of either M or N and $k = 0, \cdots, K - 1$. The block stores these values in a table and retrieves them during simulation. The number of table entries for fixed-point and floating-point is summarized in the following table:

Number of Table Entries for N-Point FFT	
floating-point	3 $N/4$
fixed-point	N

Fixed-Point Data Types

The following diagrams show the data types used in the FFT block for fixed-point signals. You can set the sine table, accumulator, product output, and output data types displayed in the diagrams in the FFT dialog box as discussed in "Dialog Box" on page 1-46.

Inputs to the FFT block are first cast to the output data type and stored in the output buffer. Each butterfly stage then processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type. The block multiplies in a twiddle factor before each butterfly stage in a decimation-in-time FFT and after each butterfly stage in a decimation-in-frequency FFT.

Decimation-in-time IFFT
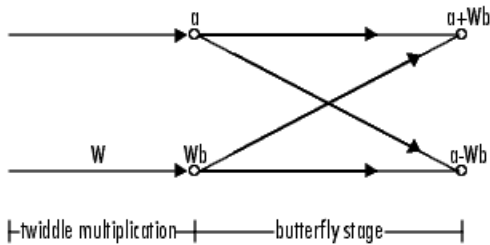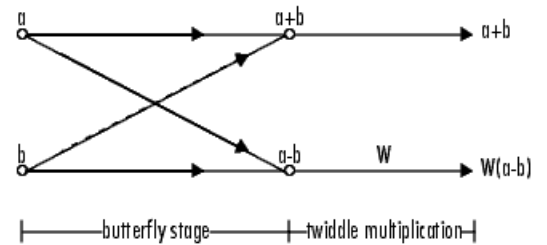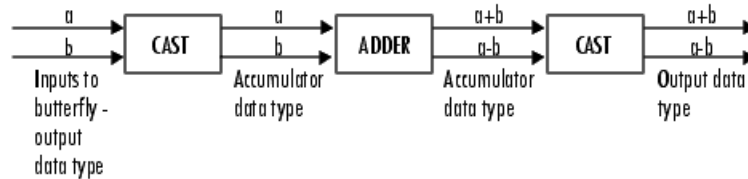
├─twiddle multiplication─┤├───────butterfly stage───────┤

Decimation-in-frequency IFFT

├───────butterfly stage───────┤├─twiddle multiplication─┤

Butterfly stage data types

Twiddle multiplication data types

The multiplier output appears in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, refer to "Multiplication Data Types".

2-D FFT

Dialog Box

The **Main** pane of the 2-D FFT dialog box appears as shown in the following figure.
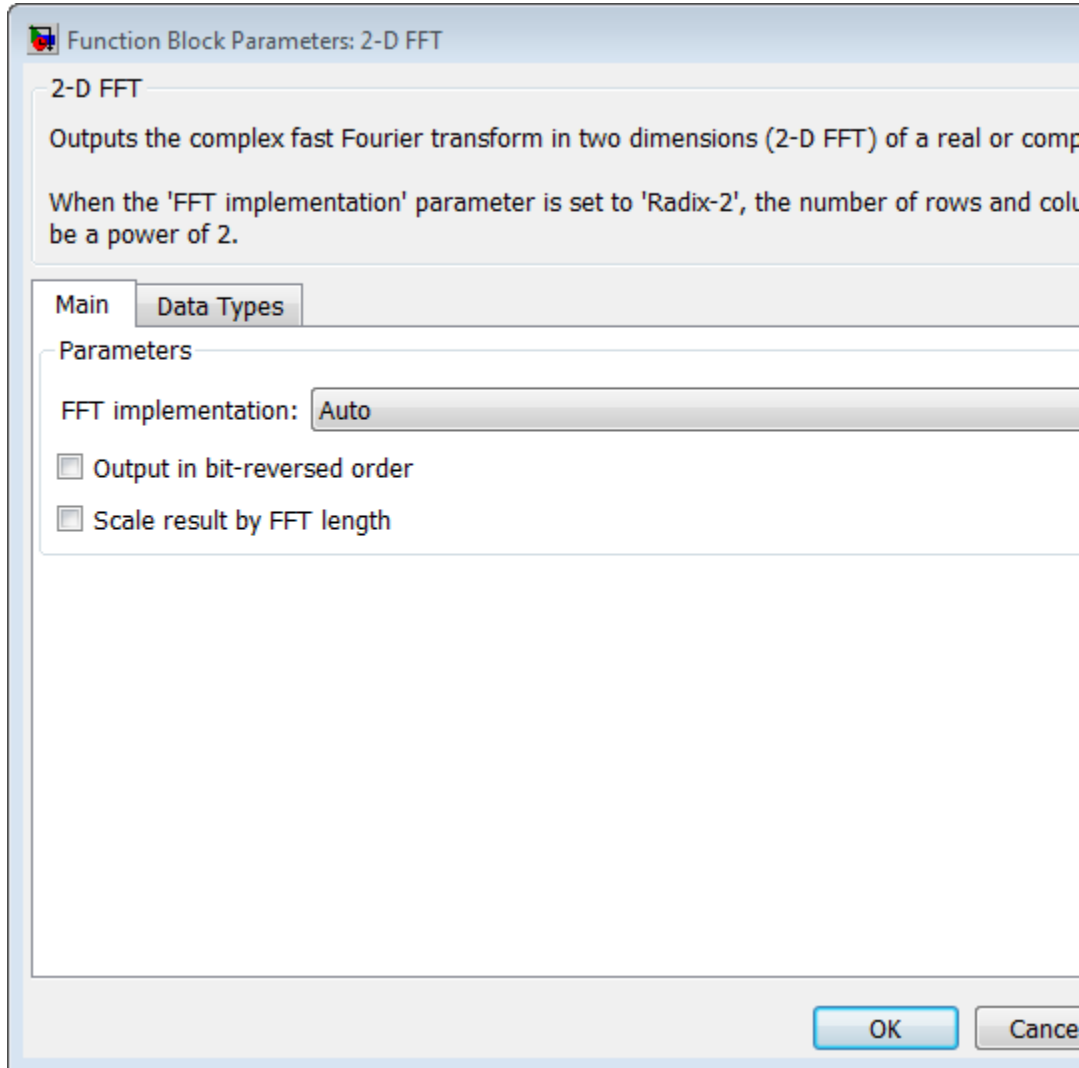
FFT implementation

Set this parameter to FFTW [1], [2] to support an arbitrary length input signal. The block restricts generated code with FFTW implementation to host computers capable of running MATLAB.

Set this parameter to Radix-2 for bit-reversed processing, fixed or floating-point data, or for portable C-code generation using the "Simulink Coder™". The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or if possible choose the FFTW implementation. See "Radix-2 Implementation" on page 1-42.

Set this parameter to Auto to let the block choose the FFT implementation. For non-power-of-two transform lengths, the block restricts generated code to MATLAB host computers.

Output in bit-reversed order

Designate the order of the output channel elements relative to the ordering of the input elements. When you select this check box, the output channel elements appear in bit-reversed order relative to the input ordering. If you clear this check box, the output channel elements appear in linear order relative to the input ordering.

Linearly ordering the output requires extra data sorting manipulation. For more information, see "Bit-Reversed Order" on page 1-51.

Scale result by FFT length

When you select this parameter, the block divides the output of the FFT by the FFT length. This option is useful when you want the output of the FFT to stay in the same amplitude range as its input. This is particularly useful when working with fixed-point data types.

The **Data Types** pane of the 2-D FFT dialog box appears as shown in the following figure.

2-D FFT

Rounding mode

Select the "Rounding Modes" for fixed-point operations. The sine table values do not obey this parameter; instead, they always round to `Nearest`.

Overflow mode

Select the Overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

Specify the product output data type. See Fixed-Point Data Types on page 44 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

Specify the accumulator data type. See Fixed-Point Data Types on page 44 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type

Specify the output data type. See Fixed-Point Data Types on page 44 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  – When you select the **Divide butterfly outputs by two** check box, the ideal output word and fraction lengths are the same as the input word and fraction lengths.

– When you clear the **Divide butterfly outputs by two** check box, the block computes the ideal output word and fraction lengths according to the following equations:

$$WL_{ideal\ output} = WL_{input} + floor(\log_2(FFT\ length - 1)) + 1$$

$$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, fixdt(1,16,0)

Click the **Show data type assistant** button [»] to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see fxptdlg, a reference page on the Fixed-Point Tool in the Simulink documentation.

Example   **Bit-Reversed Order**

Two numbers are bit-reversed values of each other when the binary representation of one is the mirror image of the binary representation of the other. For example, in a three-bit system, one and four are bit-reversed values of each other because the three-bit binary representation of one, 001, is the mirror image of the three-bit binary

representation of four, 100. The following diagram shows the row indices in linear order. To put them in bit-reversed order

1 Translate the indices into their binary representation with the minimum number of bits. In this example, the minimum number of bits is three because the binary representation of 7 is 111.

2 Find the mirror image of each binary entry, and write it beside the original binary representation.

3 Translate the indices back to their decimal representation.

The row indices now appear in bit-reversed order.
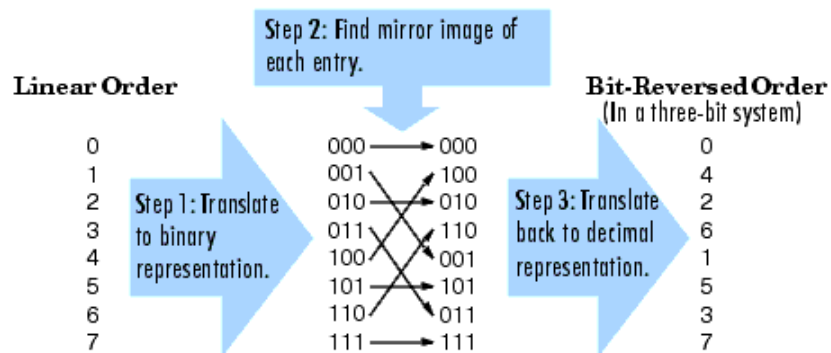
If, on the 2-D FFT block parameters dialog box, you select the **Output in bit-reversed order** check box, the block bit-reverses the order of both the columns and the rows. The next diagram illustrates the linear and bit-reversed outputs of the 2-D FFT block. The output values are the same, but they appear in different order.

Input to FFT block
(must be linear order)

$$\begin{bmatrix} 7 & 6 & 7 & 1 & 3 & 6 & 2 & 3 \\ 1 & 3 & 7 & 8 & 7 & 0 & 1 & 6 \\ 4 & 4 & 3 & 1 & 3 & 5 & 1 & 6 \\ 3 & 6 & 7 & 4 & 3 & 3 & 5 & 4 \\ 7 & 7 & 0 & 2 & 6 & 6 & 2 & 3 \\ 6 & 5 & 2 & 1 & 4 & 4 & 4 & 7 \\ 3 & 1 & 6 & 0 & 1 & 5 & 1 & 6 \\ 0 & 3 & 0 & 5 & 5 & 3 & 5 & 5 \end{bmatrix}$$

2-D FFT

2-D FFT

Output in linear order

Output in bit-reversed order
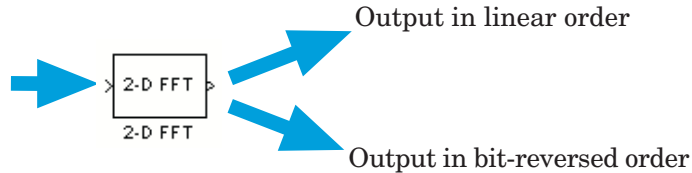
Output in linear order

Linearly ordered row and column indices

	0	1	2	3	4	5	6	7
0	245	$13.9 - 0.4i$	$10 - 5i$	$-15.9 + 21.6i$	-13	$-15.9 - 21.6i$	$10 + 5i$	13.9
1	$-4.3 - 10.3i$	$-27.6 - 6.6i$	$-5.6 + 13.1i$	$-3.4 + 8.7i$	$1.1 - i$	$-2.6i$	$-11.5 - 11i$	$6.2 + 13i$
2	$18 - 5i$	$-4.3 - 10.4i$	$19 - 24i$	$12.4 - 11.4i$	$6 - 3i$	$-5.7 + 16.4i$	$5 + 4i$	$5.5 + 1.4i$
3	$8.4 - 2.4i$	$-0.6 + 2.7i$	$-4.5 + 1.1i$	$17.6 + 9.4i$	$11 - 9i$	$-2.2 - 13i$	$-18.4 + 25.1i$	$34 + 0.5i$
4	-9	$16.3 + 5.9i$	$14 - 31i$	$17.7 + 23.9i$	1	$17.7 - 23.9i$	$14 + 31i$	$16.3 - 5.9i$
5	$8.4 + 2.4i$	$3.4 - 5.4i$	$-18.4 - 25.1i$	$-2.2 + 13.1i$	$11 + 9i$	$17.6 - 9.4i$	$-4.5 - 1.1i$	$-1 - 2.7i$
6	$18 + 5i$	$5.5 - 1.4i$	$5 - 4i$	$-5.7 - 16.4i$	$6 + 3i$	$12.5 + 11.3i$	$19 + 24i$	$-4.3 + 10.4i$
7	$-4.4 + 10.3i$	$6.2 - 13i$	$-11.5 + 11i$	$2.6i$	$1.1 + i$	$-3.4 - 8.7i$	$-5.6 - 13.1i$	$-27.6 + 6.6i$

Output in bit-reversed order

Bit-reversed row and column indices

	0	1	2	3	4	5	6	7
0	245	-13	$10 - 5i$	$10 + 5i$	$13.9 - 0.4i$	$-15.9 - 21.6i$	$-15.9 + 21.6i$	13.9
1	-9	1	$14 - 31i$	$14 + 31i$	$16.3 + 5.9i$	$17.7 - 23.9i$	$17.7 + 23.9i$	$16.3 - 5.9i$
2	$18 - 5i$	$6 - 3i$	$19 - 24i$	$5 + 4i$	$-4.3 - 10.4i$	$-5.7 + 16.4i$	$12.4 - 11.4i$	$5.5 + 1.4i$
3	$18 + 5i$	$6 + 3i$	$5 - 4i$	$19 + 24i$	$5.5 - 1.4i$	$12.5 + 11.3i$	$-5.7 - 16.4i$	$34 + 0.5i$
4	$-4.3 - 10.3i$	$1.1 - i$	$-5.6 + 13.1i$	$-11.5 - 11i$	$-27.6 - 6.6i$	$-2.6i$	$-3.4 + 8.7i$	$6.2 + 13i$
5	$8.4 + 2.4i$	$11 + 9i$	$-18.4 - 25.1i$	$-4.5 - 1.1i$	$3.4 - 5.4i$	$17.6 - 9.4i$	$-2.2 + 13i$	$-1 - 2.7i$
6	$8.4 - 2.4i$	$11 - 9i$	$-4.5 + 1.1i$	$-18.4 + 25.1i$	$-0.6 + 2.7i$	$-2.2 - 13i$	$17.6 + 9.4i$	$34 + 0.5i$
7	$-4.4 + 10.3i$	$1.1 + i$	$-11.5 + 11i$	$-5.6 - 13.1$	$6.2 - 13i$	$-3.4 - 8.7i$	$2.6i$	$-27.6 + 6.6i$

2-D FFT

References   [1] FFTW (http://www.fftw.org)

[2] Frigo, M. and S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT,"*Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, 1998, pp. 1381-1384.

See Also

2-D DCT	Computer Vision System Toolbox software
2-D IDCT	Computer Vision System Toolbox software
2-D IFFT	Computer Vision System Toolbox software
2-D IFFT	Computer Vision System Toolbox software
bitrevorder	Signal Processing Toolbox software
fft	MATLAB
ifft	MATLAB
"Simulink Coder"	Simulink Coder

Purpose   Compute two-dimensional fast Fourier transform of input

Library   Transforms

Description
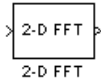
> `2-D FFT`
>
> 2-D FFT

Note The 2-D FFT block is obsolete. It may be removed in a future version of the Computer Vision System Toolbox software. Use the replacement block 2-D FFT.

Refer to "FFT and IFFT Support for Non-Power-of-Two Transform Length with FFTW Library" in the R2011b Release Notes for details related to these changes.

See the 2-D FFT documentation to view the reference material related to the obsolete block.

2-D FIR Filter

Purpose   Perform 2-D FIR filtering on input matrix

Library   Filtering

                visionfilter

Description   The 2-D Finite Impulse Response (FIR) filter block filters the input matrix I using the coefficient matrix H or the coefficient vectors HH and HV.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	- Double-precision floating point - Single-precision floating point - Fixed point 8-, 16-, 32-bit signed integer 8-, 16-, 32-bit unsigned integer	Yes
H	Matrix of filter coefficients	Same as I port.	Yes
HH	Vector of filter coefficients	Same as I port. The input to ports HH and HV must be the same data type.	Yes
HV	Vector of filter coefficients	Same as I port. The input to ports HH and HV must be the same data type.	Yes
PVal	Scalar value that represents the constant pad value	Input must have the same data type as the input to I port.	Yes
Output	Scalar, vector, or matrix of filtered values	Same as I port.	Yes

If the input has a floating-point data type, then the output uses the same data type. Otherwise, the output can be any fixed-point data type.

Select the **Separable filter coefficients** check box if your filter coefficients are separable. Using separable filter coefficients reduces the amount of calculations the block must perform to compute the output. For example, suppose your input image is *M*-by-*N* and your filter coefficient matrix is x-by-y. For a nonseparable filter with the **Output size** parameter set to `Same as input port I`, it would take

$$x \cdot y \cdot M \cdot N$$

multiply-accumulate (MAC) operations for the block to calculate the output. For a separable filter, it would only take

$$(x + y) \cdot M \cdot N$$

MAC operations. If you do not know whether or not your filter coefficients are separable, use the `isfilterseparable` function.

Here is an example of the function syntax, `[S, HCOL, HROW] = isfilterseparable(H)`. The `isfilterseparable` function takes the filter kernel, `H`, and returns `S`, `HCOL` and `HROW`. Here, `S` is a Boolean variable that is 1 if the filter is separable and 0 if it is not. `HCOL` is a vector of vertical filter coefficients, and `HROW` is a vector of horizontal filter coefficients.

Use the **Coefficient source** parameter to specify how to define your filter coefficients. If you select the **Separable filter coefficients** check box and then select a **Coefficient source** of `Specify via dialog`, the **Vertical coefficients (across height)** and **Horizontal coefficients (across width)** parameters appear in the dialog box. You can use these parameters to enter vectors of vertical and horizontal filter coefficients, respectively.

You can also use the variables `HCOL` and `HROW`, the output of the `isfilterseparable` function, for these parameters. If you select the **Separable filter coefficients** check box and then select a **Coefficient source** of `Input port`, ports HV and HH appear on the block. Use these ports to specify vectors of vertical and horizontal filter coefficients.

2-D FIR Filter

If you clear the **Separable filter coefficients** check box and select a **Coefficient source** of `Specify via dialog`, the **Coefficients** parameter appears in the dialog box. Use this parameter to enter your matrix of filter coefficients.

If you clear the **Separable filter coefficients** check box and select a **Coefficient source** of `Input port`, port **H** appears on the block. Use this port to specify your filter coefficient matrix.

The block outputs the result of the filtering operation at the Output port. The `Output size` parameter and the sizes of the inputs at ports **I** and **H** dictate the dimensions of the output. For example, assume that the input at port I has dimensions (Mi, Ni) and the input at port H has dimensions (Mh, Nh). If you select an **Output size** of `Full`, the output has dimensions ($Mi+Mh-1$, $Ni+Nh-1$). If you select an **Output size** of `Same as input port I`, the output has the same dimensions as the input at port I. If you select an **Output size** of `Valid`, the block filters the input image only where the coefficient matrix fits entirely within it, so no padding is required. The output has dimensions ($Mi-Mh+1$, $Ni-Nh+1$). However, if `all(size(I)<size(H))`, the block errors out.

Use the **Padding options** parameter to specify how to pad the boundary of your input matrix. To pad your matrix with a constant value, select `Constant`. To pad your input matrix by repeating its border values, select `Replicate`. To pad your input matrix with its mirror image, select `Symmetric`. To pad your input matrix using a circular repetition of its elements, select `Circular`. For more information on padding, see the Image Pad block reference page.

If, for the **Padding options** parameter, you select `Constant`, the **Pad value source** parameter appears in the dialog box. If you select `Specify via dialog`, the **Pad value** parameter appears in the dialog box. Use this parameter to enter the constant value with which to pad your matrix. If you select **Pad value source** of `Input port`, the PVal port appears on the block. Use this port to specify the constant value with which to pad your matrix. The pad value must be real if the input image is real. You will get an error message if the pad value is complex when the input image is real.

Use the **Filtering based on** parameter to specify the algorithm by which the block filters the input matrix. If you select Convolution and set the **Output size** parameter to Full, the block filters your input using the following algorithm

$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) * H(i-m, j-n)$$

where $0 \le i < Ma + Mh - 1$ and $0 \le j < Na + Nh - 1$. If you select Correlation and set the **Output size** parameter to Full, the block filters your input using the following algorithm

$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) \cdot conj(H(m+i, n+j))$$

where $0 \le i < Ma + Mh - 1$ and $0 \le j < Na + Nh - 1$.

TThe imfilter function from the Image Processing Toolbox™ product similarly performs N-D filtering of multidimensional images.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D FIR Filter block for fixed-point signals.
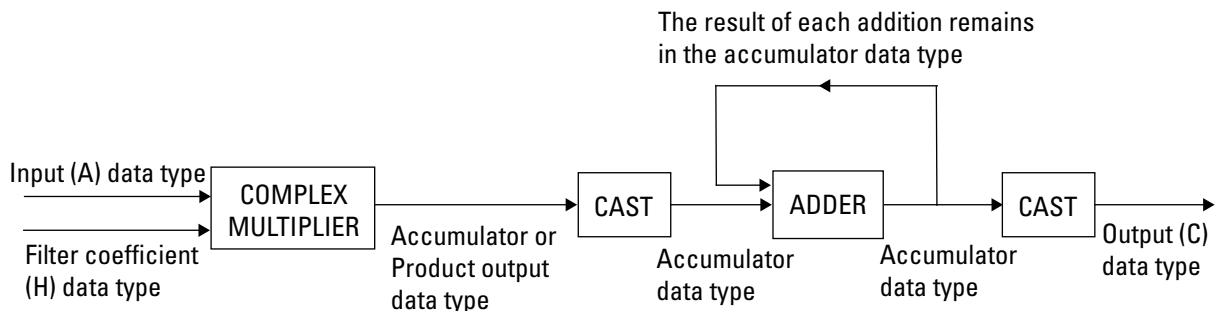
2-D FIR Filter

You can set the coefficient, product output, accumulator, and output data types in the block mask as discussed in "Dialog Box" on page 1-60.

The output of the multiplier is in the product output data type if at least one of the inputs to the multiplier is real. If both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types".

Dialog Box

The **Main** pane of the 2-D FIR Filter dialog box appears as shown in the following figure.
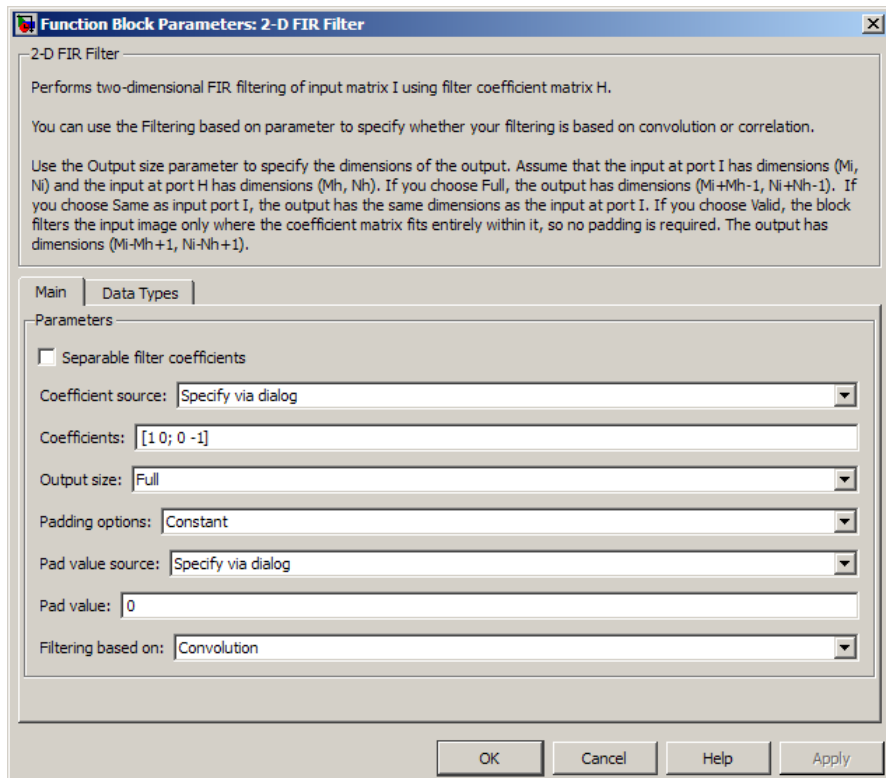
Separable filter coefficients

Select this check box if your filter coefficients are separable. Using separable filter coefficients reduces the amount of calculations the block must perform to compute the output.

Coefficient source

Specify how to define your filter coefficients. Select `Specify via dialog` to enter your coefficients in the block parameters dialog box. Select `Input port` to specify your filter coefficient matrix using port H or ports HH and HV.

Coefficients

Enter your real or complex-valued filter coefficient matrix. This parameter appears if you clear the **Separable filter coefficients** check box and then select a **Coefficient source** of `Specify via dialog`. Tunable.

Vertical coefficients (across height)

Enter the vector of vertical filter coefficients for your separable filter. This parameter appears if you select the **Separable filter coefficients** check box and then select a **Coefficient source** of `Specify via dialog`.

Horizontal coefficients (across width)

Enter the vector of horizontal filter coefficients for your separable filter. This parameter appears if you select the **Separable filter coefficients** check box and then select a **Coefficient source** of `Specify via dialog`.

Output size

This parameter controls the size of the filtered output. If you choose `Full`, the output has dimensions ($Ma+Mh$-1, $Na+Nh$-1). If you choose `Same as input port I`, the output has the same dimensions as the input at port I If you choose `Valid`, output has dimensions ($Ma-Mh$+1, $Na-Nh$+1).

Padding options

Specify how to pad the boundary of your input matrix. Select `Constant` to pad your matrix with a constant value. Select `Replicate` to pad your input matrix by repeating its border

values. Select `Symmetric`to pad your input matrix with its mirror image. Select `Circular` to pad your input matrix using a circular repetition of its elements. This parameter appears if you select an **Output size** of `Full` or `Same as input port I`.

Pad value source

Use this parameter to specify how to define your constant boundary value. Select `Specify via dialog` to enter your value in the block parameters dialog box. Select `Input port` to specify your constant value using the PVal port. This parameter appears if you select a **Padding options** of `Constant`.

Pad value

Enter the constant value with which to pad your matrix. This parameter is visible if, for the **Pad value source** parameter, you select `Specify via dialog`. Tunable. The pad value must be real if the input image is real. You will get an error message if the pad value is complex when the input image is real.

Filtering based on

Specify the algorithm by which the block filters the input matrix. You can select `Convolution` or `Correlation`.

The **Data Types** pane of the 2-D FIR Filter dialog box appears as shown in the following figure.
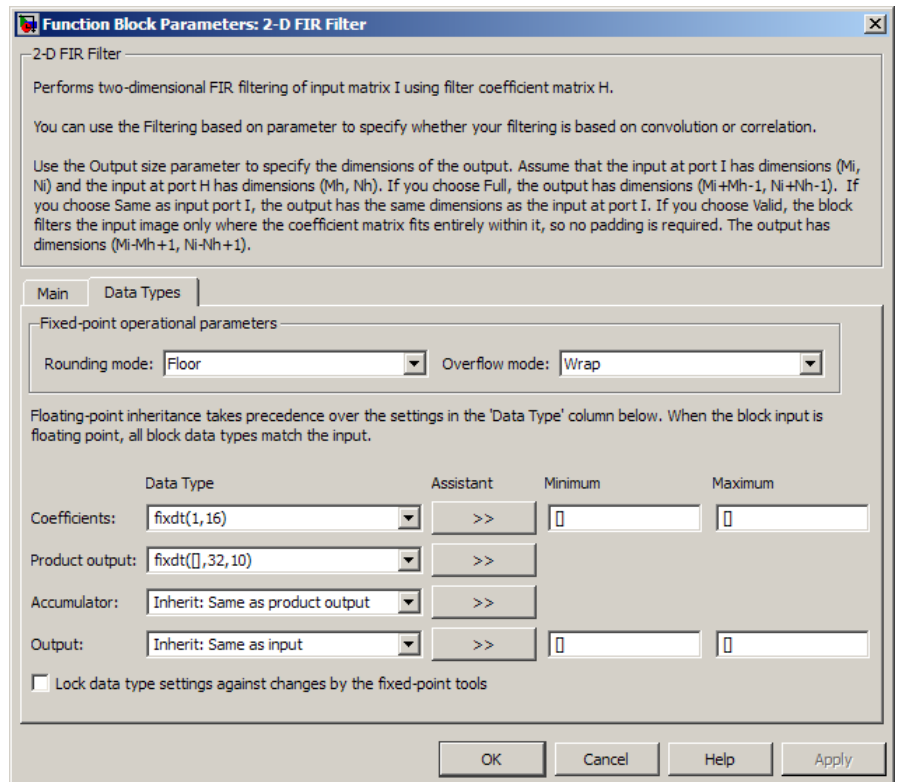
Rounding mode

Select the "Rounding Modes" for fixed-point operations.

Overflow mode

Select the Overflow mode for fixed-point operations.

Coefficients

Choose how to specify the word length and the fraction length of the filter coefficients.

- When you select Inherit: Same word length as input, the word length of the filter coefficients match that of the input to the block. In this mode, the block automatically sets

the fraction length of the coefficients to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `fixdt(1,16)`, you can enter the word length of the coefficients, in bits. In this mode, the block automatically sets the fraction length of the coefficients to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `fixdt(1,16,0)`, you can enter the word length and the fraction length of the coefficients, in bits.

- When you select `<data type expression>`, you can enter the data type expression.

The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; instead, they always saturated and rounded to `Nearest`.

Click the **Show data type assistant** button ⎵⎵⎵ ›› ⎵⎵⎵ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Product output

Use this parameter to specify how to designate the product output word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-59 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit:  Same as input`, these characteristics match those of the input to the block.

- When you select `fixdt([],16,0)`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `<data type expression>`, you can enter the data type expression.

If you set the **Coeffient source** (on the **Main** tab) to `Input port` the Product Output will inherit its sign according to the inputs. If either or both input **I1** and **I2** are signed, the Product Output will be signed. Otherwise, the Product Output is unsigned. The following table shows all cases.

Sign of Input I1	Sign of Input I2	Sign of Product Output
unsigned	unsigned	unsigned
unsigned	signed	signed
signed	unsigned	signed
signed	signed	signed

Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator

Use this parameter to specify how to designate the accumulator word and fraction lengths. Refer to "Fixed-Point Data Types" on page 1-59 and "Multiplication Data Types" in the DSP System Toolbox™ documentation for illustrations depicting the use of the accumulator data type in this block. The accumulator data type is only used when both inputs to the multiplier are complex:

- When you select `Inherit:  Same as input`, these characteristics match those of the input to the block.

- When you select `Inherit:  Same as product output`, these characteristics match those of the product output.

- When you select `fixdt([],16,0)`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. All signals in the Computer Vision System Toolbox software have a bias of 0.

Click the **Show data type assistant** button ⌐ ›› ⌐ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Inherit:  Same as input`, these characteristics match those of the input to the block.

- When you select `fixdt([],16,0)`, you can enter the word length and the fraction length of the output, in bits.

  You can choose to set signedness of the output to `Auto`, `Signed` or `Unsigned`.

- When you select `<data type expression>`, you can enter the a data type expression.

Click the **Show data type assistant** button ⌐ ›› ⌐ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Lock data type settings against change by the fixed-point tools
    Select this parameter to prevent the fixed-point tools from
    overriding the data types you specify on the block mask. For more
    information, see fxptdlg, a reference page on the Fixed-Point
    Tool in the Simulink documentation.

See Also   imfilter   Image Processing Toolbox

2-D Histogram

Purpose   Generate histogram of input or sequence of inputs

Library   Statistics

                   visionstatistics
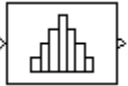
Description   The 2-D Histogram block computes the frequency distribution of the
                   elements in the input. You must use the **Find the histogram over**
                   parameter to specify whether the block computes the histogram for
                   Each column of the input or of the Entire input. The **Running
                   histogram** check box allows you to select between basic operation and
                   running operation, as described below.

                   The block distributes the elements of the input into the number of
                   discrete bins specified by the **Number of bins** parameter, n.

                   ```
                   y = hist(u,n)     % Equivalent MATLAB code
                   ```

                   The 2-D Histogram block sorts all complex input values into bins
                   according to their magnitude.

                   The histogram value for a given bin represents the frequency of
                   occurrence of the input values bracketed by that bin. You specify
                   the upper boundary of the highest-valued bin in the **Upper limit of
                   histogram** parameter, B_M, and the lower boundary of the lowest-valued
                   bin in the **Lower limit of histogram** parameter, B_m. The bins have
                   equal width of

                   $$\Delta = \frac{B_M - B_m}{n}$$

                   and centers located at

                   $$B_m + \left(k + \frac{1}{2}\right)\Delta \qquad k = 0,1,2,...,n-1$$

                   Input values that fall on the border between two bins are placed into
                   the lower valued bin; that is, each bin includes its upper boundary.
                   For example, a bin of width 4 centered on the value 5 contains the

1-68

input value 7, but not the input value 3. Input values greater than the **Upper limit of histogram** parameter or less than **Lower limit of histogram** parameter are placed into the highest valued or lowest valued bin, respectively.

The values you enter for the **Upper limit of histogram** and **Lower limit of histogram** parameters must be real-valued scalars. `NaN` and `inf` are not valid values for the **Upper limit of histogram** and **Lower limit of histogram** parameters.

Basic Operation

When the **Running histogram** check box is not selected, the 2-D Histogram block computes the frequency distribution of the current input.

When you set the **Find the histogram over** parameter to `Each column`, the 2-D Histogram block computes a histogram for each column of the M-by-N matrix independently. The block outputs an n-by-N matrix, where n is the **Number of bins** you specify. The jth column of the output matrix contains the histogram for the data in the jth column of the M-by-N input matrix.

When you set the **Find the histogram over** parameter to `Entire input`, the 2-D Histogram block computes the frequency distribution for the entire input vector, matrix or N-D array. The block outputs an n-by-1 vector, where n is the **Number of bins** you specify.

Running Operation

When you select the **Running histogram** check box, the 2–D Histogram block computes the frequency distribution of both the past and present data for successive inputs. The block resets the histogram (by emptying all of the bins) when it detects a reset event at the optional Rst port. See "Resetting the Running Histogram" on page 1-70 for more information on how to trigger a reset.

When you set the **Find the histogram over** parameter to `Each column`, the 2-D Histogram block computes a running histogram for each column of the M-by-N matrix. The block outputs an n-by-N matrix, where n is the **Number of bins** you specify. The jth column of the

output matrix contains the running histogram for the *j*th column of the *M*-by-*N* input matrix.

When you set the **Find the histogram over** parameter to `Entire input`, the 2-D Histogram block computes a running histogram for the data in the first dimension of the input. The block outputs an *n*-by-1 vector, where *n* is the **Number of bins** you specify.

Note When the 2-D Histogram block is used in running mode and the input data type is non-floating point, the output of the histogram is stored as a `uint32` data type. The largest number that can be represented by this data type is $2^{32}-1$. If the range of the `uint32` data type is exceeded, the output data will wrap back to 0.

Resetting the Running Histogram

The block resets the running histogram whenever a reset event is detected at the optional Rst port. The reset signal and the input data signal must be the same rate.

You specify the reset event using the **Reset port** menu:

- `None` — Disables the Rst port

- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

Rising edge

Rising edge

Rising edge

Rising edge

Not a rising edge because it conti
a rise from a negative value to zer

- Falling edge — Triggers a reset operation when the Rst input does one of the following:

  - Falls from a positive value to a negative value or zero

  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

Falling edge   Falling edge

Falling edge   Falling edge

Not a falling edge because it contir
a fall from a positive value to zero.

- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described earlier)

- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Dialog Box

The **Main** pane of the 2-D Histogram block dialog appears as follows.

Function Block Parameters: 2-D Histogram

2-D Histogram

Generate a histogram of the elements in the specified dimension of the input or across
time (running histogram).

| Main | Data Types |

Parameters

Lower limit of histogram: 0

Upper limit of histogram: 1

Number of bins: 256

Find the histogram over: Entire input

☐ Normalized

☐ Running histogram

| OK | Cancel | Help | Ap |
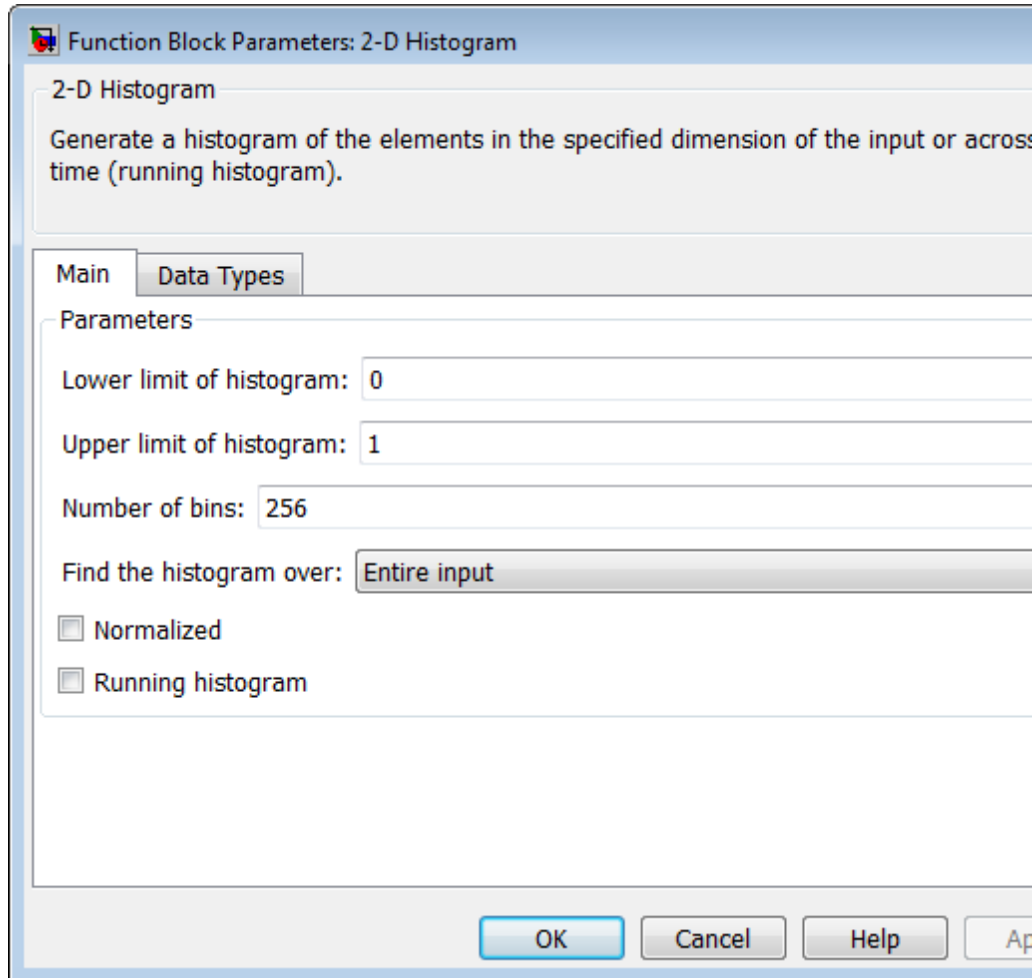
Lower limit of histogram

Enter a real-valued scalar for the lower boundary, B_m, of the lowest-valued bin. NaN and inf are not valid values for B_m. Tunable.

Upper limit of histogram

Enter a real-valued scalar for the upper boundary, B_M, of the highest-valued bin. NaN and inf are not valid values for B_M. Tunable.

Number of bins

The number of bins, n, in the histogram.

Find the histogram over

Specify whether the block finds the histogram over the entire input or along each column of the input.

Note The option will be removed in a future release.

Normalized

When selected, the output vector, v, is normalized such that sum(v) = 1.

Use of this parameter is not supported for fixed-point signals.

Running histogram

Set to enable the running histogram operation, and clear to enable basic histogram operation. For more information, see "Basic Operation" on page 1-69 and "Running Operation" on page 1-69.

Reset port

The type of event that resets the running histogram. For more information, see "Resetting the Running Histogram" on page 1-70. The reset signal and the input data signal must be the same rate. This parameter is enabled only when you select the **Running histogram** check box. For more information, see "Running Operation" on page 1-69.

The **Data Types** pane of the 2-D Histogram block dialog appears as follows.
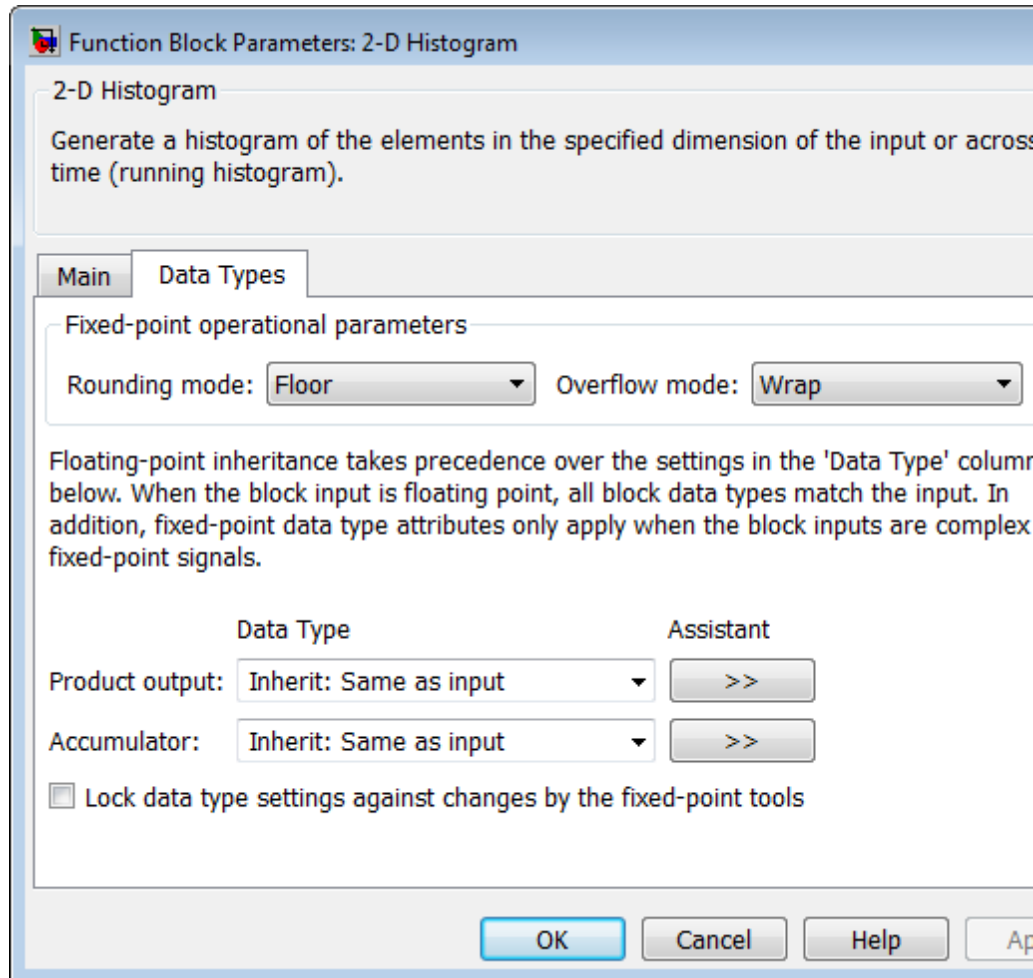
> **Note** The fixed-point parameters listed are only used for fixed-point complex inputs, which are distributed by squared magnitude.

Rounding mode

Select the "Rounding Modes" for fixed-point operations.

Overflow mode

Select the Overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See "Multiplication Data Types" for illustrations depicting the use of the product output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

Specify the accumulator data type. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point
	• Single-precision floating point
	• Fixed point (signed and unsigned)
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point
	• Single-precision floating point
	• 32-bit unsigned integers
Rst	• Double-precision floating point
	• Single-precision floating point
	• Boolean
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers

2-D Histogram

Examples   **Calculate Histogram of R, G, and B Values**
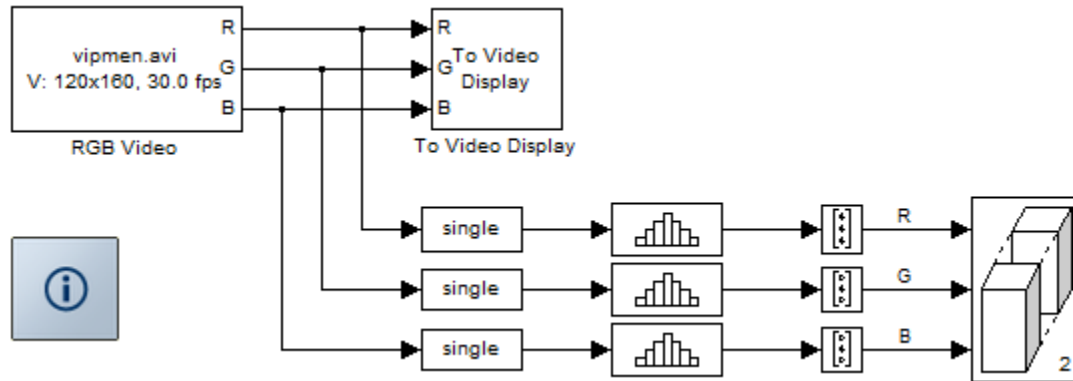
The Histogram Display example uses the Histogram block to calculate the histograms of R, G, and B values in each video frame.

See Also   hist   MATLAB

Purpose   Generate histogram of each input matrix

Library   Statistics

Description
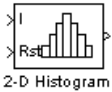
2-D Histogram

Note The 2-D Histogram block is obsolete. It may be removed in a future version of the Computer Vision System Toolbox software. Use the replacement block Histogram.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Histogram documentation to view the reference material related to the obsolete block.

2-D IDCT

Purpose   Compute 2-D inverse discrete cosine transform (IDCT)

Library   Transforms

visiontransforms

Description   The 2-D IDCT block calculates the two-dimensional inverse discrete cosine transform of the input signal. The equation for the two-dimensional IDCT is

$$f(x,y) = \frac{2}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} C(m)C(n)F(m,n)\cos\frac{(2x+1)m\pi}{2M}\cos\frac{(2y+1)n\pi}{2N}$$

where *F(m,n)* is the DCT of the signal *f(x,y)* and $C(m),C(n) = \dfrac{1}{\sqrt{2}}$ for $m,n = 0$ and $C(m),C(n) = 1$ otherwise.

The number of rows and columns of the input signal must be powers of two. The output of this block has dimensions the same dimensions as the input.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Output	2-D IDCT of the input	Same as Input port	No

If the data type of the input signal is floating point, the output of the block is the same data type.

Use the **Sine and cosine computation** parameter to specify how the block computes the sine and cosine terms in the IDCT algorithm. If you select `Trigonometric fcn`, the block computes the sine and cosine values during the simulation. If you select `Table lookup`, the block computes and stores the trigonometric values before the simulation starts. In this case, the block requires extra memory.

Fixed-Point Data Types

The following diagram shows the data types used in the 2-D IDCT block for fixed-point signals. Inputs are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.

2-D IDCT
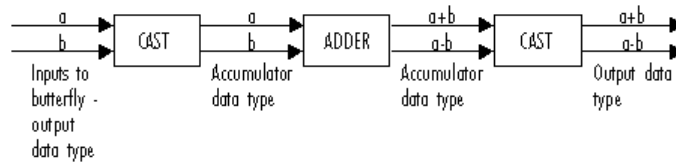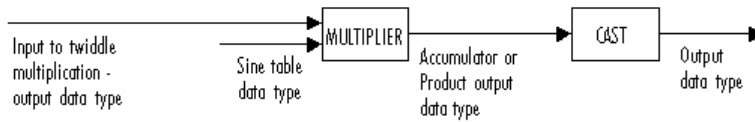
a   a+b

b   a-b

├────── butterfly stage ──────┤

OR

a   a+b   W1   W1(a+b)   W1 and W2 are twiddle table constants

b   a-b   W2   W2(a-b)

├────── butterfly stage ──────┼── twiddle multiplication ──┤

OR

a   c   ac+be

          e

          f

b   d   af+bd

├────── butterfly stage ──────┤

Butterfly Stage Data Types

a   a   a+b   a+b

b   [CAST] b [ADDER] a-b [CAST] a-b

Inputs to   Accumulator   Accumulator   Output data
butterfly -   data type   data type   type
output
data type

Twiddle Multiplication Data Types

[MULTIPLIER]   [CAST]

Input to twiddle   Accumulator or   Output
multiplication -   Sine table Product output   data type
output data type data type data type

The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types". You can set the sine table, product output, accumulator, and output data types in the block mask as discussed in the next section.

Dialog Box

The **Main** pane of the 2-D IDCT dialog box appears as shown in the following figure.

Sine and cosine computation

    Specify how the block computes the sine and cosine terms in the IDCT algorithm. If you select Trigonometric fcn, the block computes the sine and cosine values during the simulation. If

you select `Table lookup`, the block computes and stores the trigonometric values before the simulation starts. In this case, the block requires extra memory.

The **Data Types** pane of the 2-D IDCT dialog box appears as shown in the following figure.

Rounding mode
Select the "Rounding Modes" for fixed-point operations.

Overflow mode
Select the Overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit:  Same word length as input`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 1-81 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button $\boxed{\quad \gg \quad}$ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 1-81 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button ![>>] to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 1-81 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit: Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{ideal\ output} = WL_{input} + floor(\log_2(DCT\ length - 1)) + 1$$

  $$FL_{ideal\ output} = FL_{input}$$

  Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button `>>` to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling tool in the Fixed-Point Tool. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References

[1] Chen, W.H, C.H. Smith, and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform,"*IEEE Trans. Commun.*, vol. COM-25, pp. 1004-1009. 1977.

[2] Wang, Z. "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 803-816, Aug. 1984.

See Also

2-D DCT	Computer Vision System Toolbox software
2-D FFT	Computer Vision System Toolbox software
2-D IFFT	Computer Vision System Toolbox software

2-D IFFT

Purpose   2-D Inverse fast Fourier transform of input

Library   Transforms

visiontransforms

Description
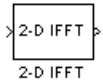
The 2-D IFFT block computes the inverse fast Fourier transform (IFFT) of an *M*-by-*N* input matrix in two steps. First, it computes the one-dimensional IFFT along one dimension (row or column). Next, it computes the IFFT of the output of the first step along the other dimension (column or row).

The output of the IFFT block is equivalent to the MATLAB `ifft2` function:

```
y = ifft2(A)      % Equivalent MATLAB code
```

Computing the IFFT of each dimension of the input matrix is equivalent to calculating the two-dimensional inverse discrete Fourier transform (IDFT), which is defined by the following equation:

$$f(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(m,n) e^{j\frac{2\pi mx}{M}} e^{j\frac{2\pi ny}{N}}$$

where $0 \le x \le M-1$ and $0 \le y \le N-1$.

The output of this block has the same dimensions as the input. If the input signal has a floating-point data type, the data type of the output signal uses the same floating-point data type. Otherwise, the output can be any fixed-point data type. The block computes scaled and unscaled versions of the IFFT.

The input to this block can be floating-point or fixed-point, real or complex, and conjugate symmetric. The block uses one of two possible FFT implementations. You can select an implementation based on the FFTW library [1], [2], or an implementation based on a collection of Radix-2 algorithms. You can select Auto to allow the block to choose the implementation.

Port Description

Port	Description	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
Output	2-D IFFT of the input	Same as Input port	Yes

FFTW Implementation

The FFTW implementation provides an optimized FFT calculation including support for power-of-two and non-power-of-two transform lengths in both simulation and code generation. Generated code using the FFTW implementation will be restricted to MATLAB host computers. The data type must be floating-point. Refer to "Simulink Coder" for more details on generating code.

Radix-2 Implementation

The Radix-2 implementation supports bit-reversed processing, fixed or floating-point data, and allows the block to provide portable C-code generation using the "Simulink Coder". The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or if possible choose the FFTW implementation.

With Radix-2 selected, the block implements one or more of the following algorithms:

- Butterfly operation

- Double-signal algorithm

- Half-length algorithm

- Radix-2 decimation-in-time (DIT) algorithm

- Radix-2 decimation-in-frequency (DIF) algorithm

Radix-2 Algorithms for Real or Complex Input Complexity Floating-Point Signals

Parameter Settings	Algorithms Used for IFFT Computation
☐ Input is in bit-reversed order ☐ Input is conjugate symmetric	Butterfly operation and radix-2 DIT
☑ Input is in bit-reversed order ☐ Input is conjugate symmetric	Radix-2 DIF
☐ Input is in bit-reversed order ☑ Input is conjugate symmetric	Butterfly operation and radix-2 DIT in conjunction with the half-length and double-signal algorithms
☑ Input is in bit-reversed order ☑ Input is conjugate symmetric	Radix-2 DIF in conjunction with the half-length and double-signal algorithms

Radix-2 Algorithms for Real or Complex Input Complexity Fixed-Point Signals

Other Parameter Settings	Algorithms Used for IFFT Computation
☐ Input is in bit-reversed order ☐ Input is conjugate symmetric	Butterfly operation and radix-2 DIT
☑ Input is in bit-reversed order ☐ Input is conjugate symmetric	Radix-2 DIF

Note The **Input is conjugate symmetric** parameter cannot be used for fixed-point signals.

Radix-2 Optimization for the Table of Trigonometric Values

In certain situations, the block's Radix–2 algorithm computes all the possible trigonometric values of the twiddle factor

$$e^{j\frac{2\pi k}{K}}$$

where K is the greater value of either M or N and $k = 0, \cdots, K-1$. The block stores these values in a table and retrieves them during simulation. The number of table entries for fixed-point and floating-point is summarized in the following table:

Number of Table Entries for N-Point FFT	
floating-point	3 N/4
fixed-point	N

Fixed-Point Data Types

The following diagrams show the data types used in the IFFT block for fixed-point signals. You can set the sine table, accumulator, product

output, and output data types displayed in the diagrams in the IFFT dialog box as discussed in "Dialog Box" on page 1-94.

Inputs to the IFFT block are first cast to the output data type and stored in the output buffer. Each butterfly stage then processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type. The block multiplies in a twiddle factor before each butterfly stage in a decimation-in-time IFFT and after each butterfly stage in a decimation-in-frequency IFFT.

Decimation-in-time IFFT
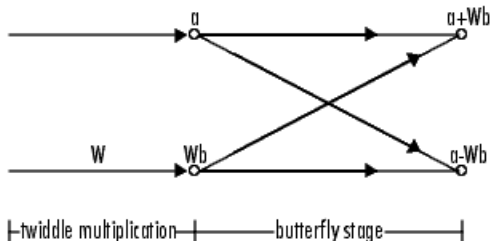
Decimation-in-frequency IFFT
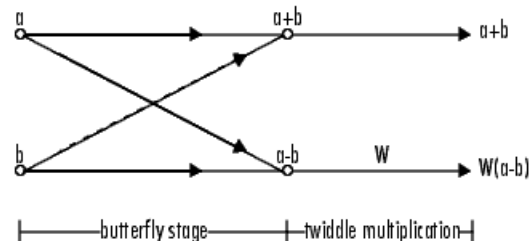
Butterfly stage data types
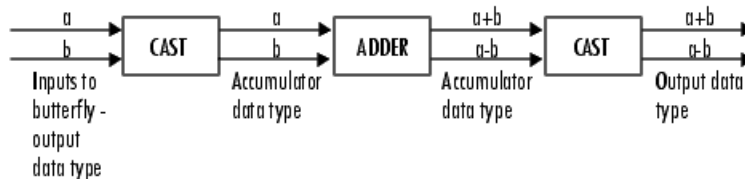
Twiddle multiplication data types

The multiplier output appears in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, refer to "Multiplication Data Types" in the DSP System Toolbox documentation.

2-D IFFT

Dialog Box

The **Main** pane of the 2-D IFFT dialog box appears as shown in the following figure.
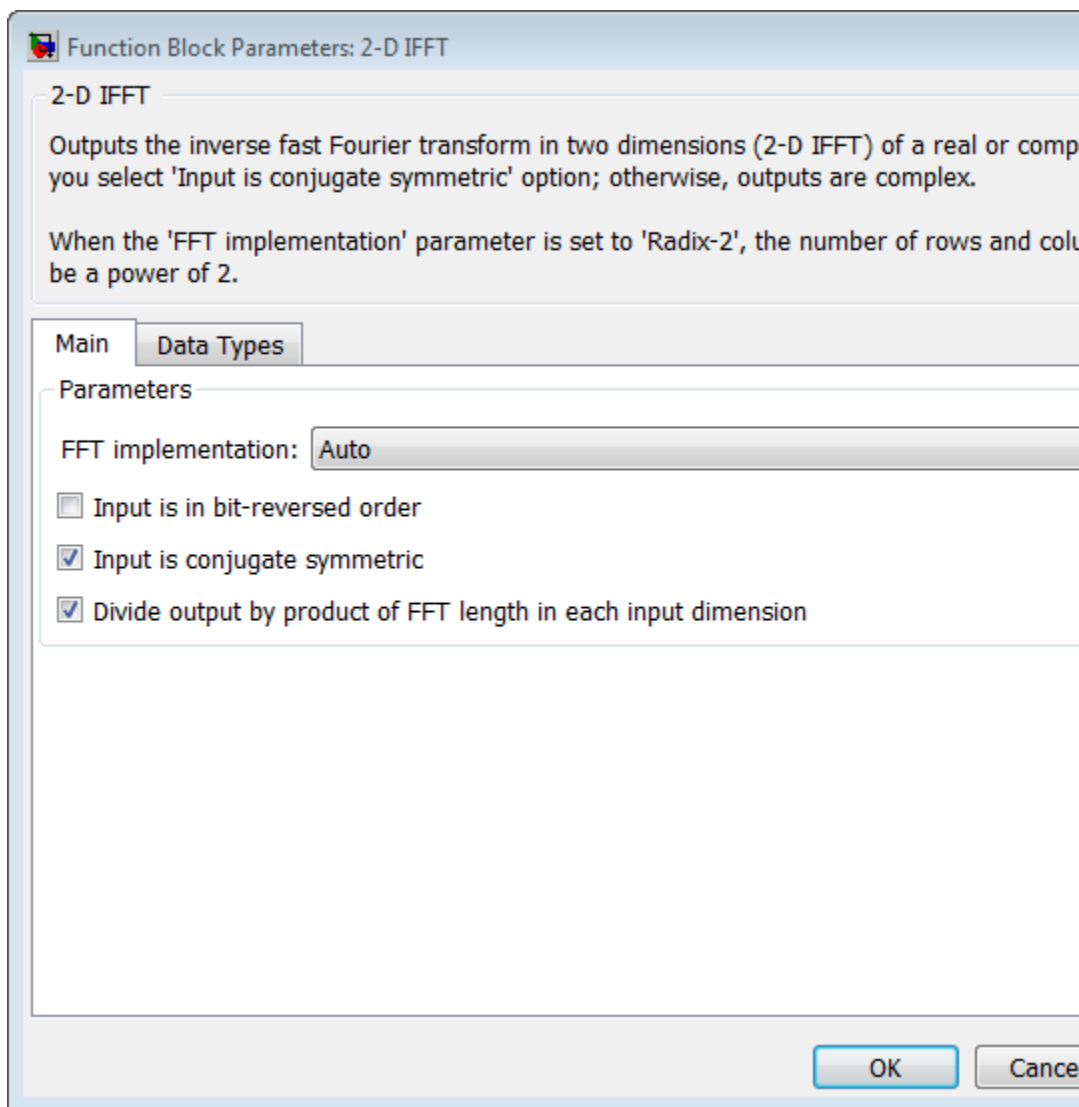
1-94

FFT implementation

Set this parameter to FFTW [1], [2] to support an arbitrary length input signal. The block restricts generated code with FFTW implementation to MATLAB host computers.

Set this parameter to Radix-2 for bit-reversed processing, fixed or floating-point data, or for portable C-code generation using the "Simulink Coder". The dimensions of the input matrix, M and N, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or if possible choose the FFTW implementation. See "Radix-2 Implementation" on page 1-89.

Set this parameter to Auto to let the block choose the FFT implementation. For non-power-of-two transform lengths, the block restricts generated code to MATLAB host computers.

Input is in bit-reversed order

Select or clear this check box to designate the order of the input channel elements. Select this check box when the input should appear in reversed order, and clear it when the input should appear in linear order. The block yields invalid outputs when you do not set this parameter correctly. This check box only appears when you set the **FFT implementation** parameter to Radix-2 or Auto.

For more information ordering of the output, see "Bit-Reversed Order" on page 1-51. The 2-D FFT block bit-reverses the order of both the columns and the rows.

Input is conjugate symmetric

Select this option when the block inputs both floating point and conjugate symmetric, and you want real-valued outputs. This parameter cannot be used for fixed-point signals. Selecting this check box optimizes the block's computation method.

The FFT block yields conjugate symmetric output when you input real-valued data. Taking the IFFT of a conjugate symmetric input

matrix produces real-valued output. Therefore, if the input to the block is both floating point and conjugate symmetric, and you select the this check box, the block produces real-valued outputs.

If the IFFT block inputs conjugate symmetric data and you do not select this check box, the IFFT block outputs a complex-valued signal with small imaginary parts. The block outputs invalid data if you select this option with non conjugate symmetric input data.

Divide output by product of FFT length in each input dimension
Select this check box to compute the scaled IFFT. The block computes scaled and unscaled versions of the IFFT. If you select this option, the block computes the scaled version of the IFFT. The unscaled IFFT is defined by the following equation:

$$f(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(m,n) e^{j\frac{2\pi mx}{M}} e^{j\frac{2\pi ny}{N}}$$

where $0 \le x \le M-1$ and $0 \le y \le N-1$.

The scaled version of the IFFT multiplies the above unscaled

version by $\frac{1}{MN}$.

The **Data Types** pane of the 2-D IFFT dialog box appears as shown in the following figure.

Function Block Parameters: 2-D IFFT

2-D IFFT

Outputs the inverse fast Fourier transform in two dimensions (2-D IFFT) of a real or c
you select 'Input is conjugate symmetric' option; otherwise, outputs are complex.

When the 'FFT implementation' parameter is set to 'Radix-2', the number of rows and
be a power of 2.

| Main | Data Types |

Fixed-point operational parameters

Rounding mode: Floor ▼   Overflow mode: Wrap

Floating-point inheritance takes precedence over the settings in the 'Data Type' colum
is floating point, all block data types match the input. In addition, for fixed-point sine ta
is always equal to the word length minus one.

	Data Type	Assistant	Minimum
Sine table:	Inherit: Same word length as input ▼	>>	
Product output:	Inherit: Inherit via internal rule ▼	>>	
Accumulator:	Inherit: Inherit via internal rule ▼	>>	
Output:	Inherit: Inherit via internal rule ▼	>>	[]

☐ Lock data type settings against changes by the fixed-point tools

OK   Ca

Rounding mode

> Select the "Rounding Modes" for fixed-point operations. The sine table values do not obey this parameter; instead, they always round to `Nearest`.

Overflow mode

> Select the Overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

> Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:
>
> - A rule that inherits a data type, for example, `Inherit:  Same word length as input`
>
> - An expression that evaluates to a valid data type, for example, `fixdt(1,16)`
>
> The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

> Specify the product output data type. See Fixed-Point Data Types on page 91 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:
>
> - A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
>
> - An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`
>
> Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

Specify the accumulator data type. SeeFixed-Point Data Types on page 91 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type

Specify the output data type. See Fixed-Point Data Types on page 91 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  - When you select the **Divide butterfly outputs by two** check box, the ideal output word and fraction lengths are the same as the input word and fraction lengths.

      – When you clear the **Divide butterfly outputs by two** check box, the block computes the ideal output word and fraction lengths according to the following equations:

$$WL_{ideal\ output} = WL_{input} + floor(\log_2(FFT\ length - 1)) + 1$$

$$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References   [1] FFTW (`http://www.fftw.org`)

[2] Frigo, M. and S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT,"*Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, 1998, pp. 1381-1384.

See Also

2-D DCT	Computer Vision System Toolbox software
2-D FFT	Computer Vision System Toolbox software
2-D IDCT	Computer Vision System Toolbox software
2-D FFT	Computer Vision System Toolbox software
2-D IFFT	Computer Vision System Toolbox software
bitrevorder	Signal Processing Toolbox software
fft	MATLAB
ifft	MATLAB
"Simulink Coder"	Simulink

2-D IFFT (To Be Removed)

Purpose   Compute 2-D IFFT of input

Library   Transforms

                   viptransforms

Description

Note The 2-D IFFT block is obsolete. It may be removed in a future version of the Computer Vision System Toolbox software. Use the replacement block 2-D IFFT.

Refer to "FFT and IFFT Support for Non-Power-of-Two Transform Length with FFTW Library" in the R2011b Release Notes for details related to these changes.

See the 2-D IFFT documentation to view the reference material related to the obsolete block.

Purpose   Find maximum values in input or sequence of inputs

Library   Statistics

visionstatistics

Description   The 2-D Maximum block identifies the value and/or position of the smallest element in each row or column of the input, or along a specified dimension of the input. The 2-D Maximum block can also track the maximum values in a sequence of inputs over a period of time.

The 2-D Maximum block supports real and complex floating-point, fixed-point, and Boolean inputs. Real fixed-point inputs can be either signed or unsigned, while complex fixed-point inputs must be signed. The output data type of the maximum values match the data type of the input. The block outputs double index values, when the input is double, and uint32 otherwise.

Port Descriptions

Port	Input/Output	Supported Data Types
Input	Scalar, vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Rst	Scalar value	• Double-precision floating point

Port	Input/Output	Supported Data Types
		• Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Val	Maximum value output based on the "Value Mode" on page 1-105	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Idx	One-based output location of the maximum value based on the "Index Mode" on page 1-106	• Double-precision floating point • 32-bit unsigned integers

Value Mode

When you set the **Mode** parameter to `Value`, the block computes the maximum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each sample time, and outputs the array *y*. Each element in *y* is the maximum value in the corresponding column, row, vector, or entire input. The output *y* depends on the setting of the **Find the maximum value over** parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the maximum value of each vector over the second dimension of the input. For an *M*-by-*N* input matrix, the block outputs an *M*-by-1 column vector at each sample time.

- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the maximum value of each vector over the first dimension of the input. For an *M*-by-*N* input matrix, the block outputs a 1-by-*N* row vector at each sample time.

  In this mode, the block treats length-*M* unoriented vector inputs as *M*-by-1 column vectors.

- `Entire input` — The output at each sample time is a scalar that contains the maximum value in the *M*-by-*N*-by-*P* input matrix.

- `Specified dimension` — The output at each sample time depends on **Dimension**. When you set **Dimension** to `1`, the block output is the same as when you select `Each column`. When you set **Dimension** to `2`, the block output is the same as when you select `Each row`. When you set **Dimension** to `3`, the block outputs an *M*-by-*N* matrix containing the maximum value of each vector over the third dimension of the input, at each sample time.

For complex inputs, the block selects the value in each row or column of the input, along vectors of a specified dimension of the input, or of

the entire input that has the maximum magnitude squared as shown below. For complex value $u = a + bi$, the magnitude squared is $a^2 + b^2$.
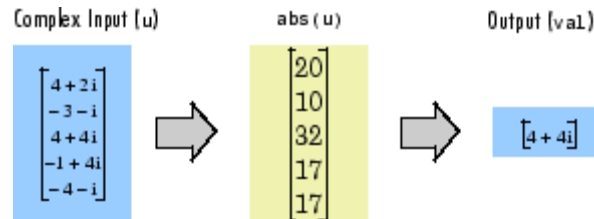
Index Mode

When you set the **Mode** parameter to Index, the block computes the maximum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input, and outputs the index array *I*. Each element in *I* is an integer indexing the maximum value in the corresponding column, row, vector, or entire input. The output *I* depends on the setting of the **Find the maximum value over** parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- Each row — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the index of the maximum value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector.

- Each column — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the index of the maximum value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector.

  In this mode, the block treats length-*M* unoriented vector inputs as *M*-by-1 column vectors.

- Entire input — The output at each sample time is a 1-by-3 vector that contains the location of the maximum value in the *M*-by-*N*-by-*P*

input matrix. For an input that is an *M*-by-*N* matrix, the output will be a 1-by-2 vector of one-based [x y] location coordinates for the maximum value.

- Specified dimension — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select Each column. If **Dimension** is set to 2, the output is the same as when you select Each row. If **Dimension** is set to 3, the output at each sample time is an *M*-by-*N* matrix containing the indices of the maximum values of each vector over the third dimension of the input.

When a maximum value occurs more than once, the computed index corresponds to the first occurrence. For example, when the input is the column vector [3 2 1 2 3]', the computed one-based index of the maximum value is 1 rather than 5 when Each column is selected.

When inputs to the block are double-precision values, the index values are double-precision values. Otherwise, the index values are 32-bit unsigned integer values.

Value and Index Mode

When you set the **Mode** parameter to Value and Index, the block outputs both the maxima and the indices.

Running Mode

When you set the **Mode** parameter to Running, the block tracks the maximum value of each channel in a time sequence of *M*-by-*N* inputs. In this mode, the block treats each element as a channel.

Resetting the Running Maximum

The block resets the running maximum whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

You specify the reset event in the **Reset port** menu:

- None — Disables the Rst port.

- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:

  - Falls from a positive value to a negative value or zero

  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

Falling edge   Falling edge

Not a falling edge because it is a continuation of a fall from a positive value to zero.

Falling edge   Falling edge

Falling edge

- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above)

- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

---

Note When running simulations in the Simulink `MultiTasking` mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This applies to any mode other than running mode and when you set the **Find the maximum value over** parameter to `Entire input` and you select the **Enable ROI processing** check box. ROI processing applies only for 2-D inputs.

You can specify a rectangle, line, label matrix, or binary mask ROI type.

Use the binary mask to specify which pixels to highlight or select.

Use the label matrix to label regions. Pixels set to `0` represent the background. Pixels set to `1` represent the first object, pixels set to `2`,

represent the second object, and so on. Use the **Label Numbers** port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter. For more information about the format of the input to the ROI port when you set the ROI to a rectangle or a line, see the Draw Shapes block reference page.

ROI Output Statistics
Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Data Types** pane of the block dialog are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 1-105. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

2-D Maximum

Dialog Box

The **Main** pane of the Maximum block dialog appears as follows.
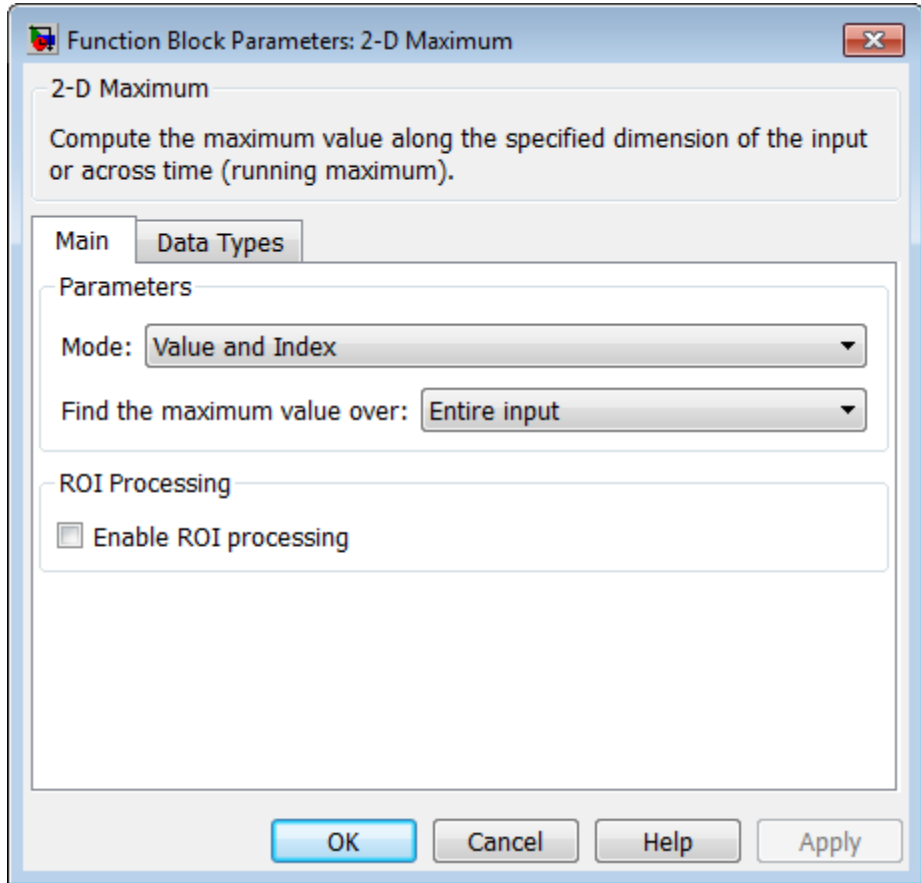
Mode

Specify the block's mode of operation:

- Value and Index — Output both the value and the index location.

- `Value` — Output the maximum value of each input matrix. For more information, see "Value Mode" on page 1-105.

- `Index`— Output the one-based index location of the maximum value. For more information, see "Index Mode" on page 1-106.

- `Running` — Track the maximum value of the input sequence over time. For more information, see "Running Mode" on page 1-107.

For the `Value`, `Index`, and `Value and Index` modes, the 2-D Maximum block produces identical results as the MATLAB `max` function when it is called as `[y I] = max(u,[],D)`, where *u* and *y* are the input and output, respectively, *D* is the dimension, and *I* is the index.

Find the maximum value over
Specify whether the block should find the maximum of the entire input each row, each column, or dimensions specified by the **Dimension** parameter.

Reset port
Specify the reset event that causes the block to reset the running maximum. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter appears only when you set the **Mode** parameter to `Running`. For information about the possible values of this parameter, see "Resetting the Running Maximum" on page 1-107.

Dimension
Specify the dimension (one-based value) of the input signal, over which the maximum is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter applies only when you set the **Find the maximum value over** parameter to `Specified dimension`.

Enable ROI processing
Select this check box to calculate the statistical value within a particular region of each image. This parameter applies only

when you set the **Find the maximum value over** parameter to `Entire input`, and the block is not in running mode.

ROI type

Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

When you set this parameter to `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the **Flag** port appears on the block.

When you set this parameter to `Label matrix`, the **Label** and **Label Numbers** ports appear on the block and the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block.

See ROI Output Statistics on page 110 for details.

ROI portion to process

Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter applies only when you set the **ROI type** parameter to `Rectangles`.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter does not apply when you set the **ROI type** parameter, to `Binary mask`.

Output flag indicating if ROI is within image bounds

When you select this check box, the **Flag** port appears on the block. This check box applies only when you set the **ROI type** parameter to `Rectangles` or `Lines`. For a description of the **Flag** port output, see the tables in "ROI Processing" on page 1-109.

Output flag indicating if label numbers are valid

When you select this check box, the **Flag** port appears on the block. This check box applies only when you set the **ROI type** parameter to Label matrix. For a description of the **Flag** port output, see the tables in "ROI Processing" on page 1-109.

The **Data Types** pane of the Maximum block dialog appears as follows.
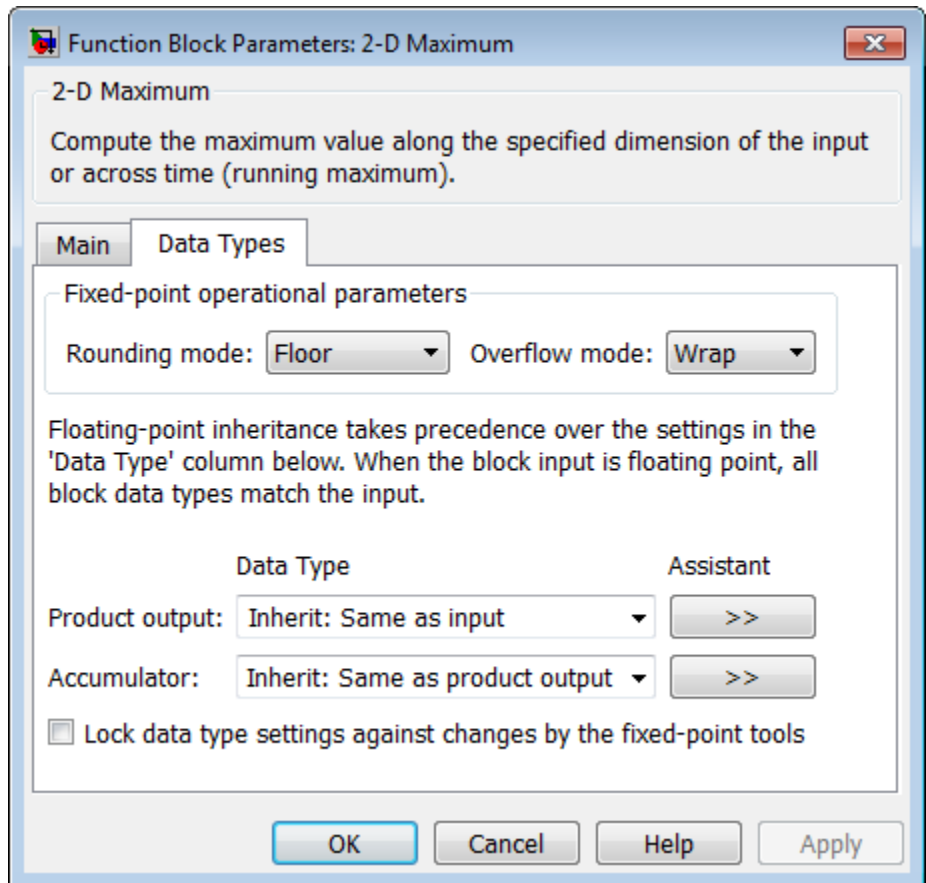
2-D Maximum

> **Note** The parameters on the **Data Types** pane are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 1-105. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Rounding mode
> Select the "Rounding Modes" for fixed-point operations.

Overflow mode
> Select the Overflow mode for fixed-point operations.

Product output data type
> Specify the product output data type. See "Fixed-Point Data Types" on page 1-111 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:
>
> - A rule that inherits a data type, for example, Inherit: Same as input
>
> - An expression that evaluates to a valid data type, for example, fixdt([],16,0)
>
> Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.
>
> See "Specify Data Types Using Data Type Assistant" for more information.
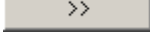
Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 1-111 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit:  Same as product output`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.
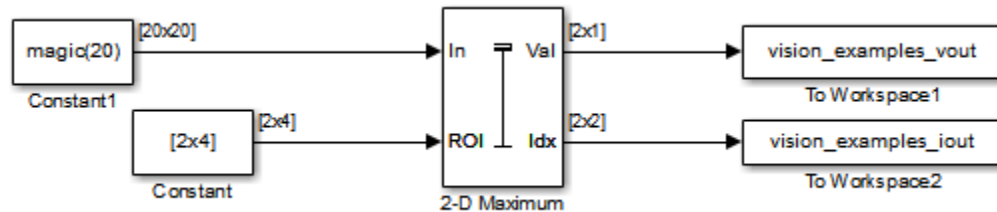
Examples

The ex_vision_2dmaximum example finds the maximum value within two ROIs. The model outputs the maximum values and their one-based [x y] coordinate locations.

2-D Maximum

See Also

2-D Mean	Computer Vision System Toolbox
2-D Minimum	Computer Vision System Toolbox
MinMax	Simulink
max	MATLAB

Purpose   Find maximum values in an input or sequence of inputs

Library   vipobslib

Description

2-D Maximum
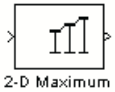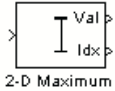
2-D Maximum

Note This 2-D Maximum block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated 2-D Maximum block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the 2-D Maximum documentation to view the reference material related to the obsolete block.

2-D Mean

Purpose   Find mean value of each input matrix

Library   Statistics

visionstatistics

Description   The 2-D Mean block computes the mean of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The 2-D Mean block can also track the mean value in a sequence of inputs over a period of time. To track the mean value in a sequence of inputs, select the **Running mean** check box.

Port Description

Port	Supported Data Types
Input	• Double-precision floating point
	• Single-precision floating point
	• Fixed point
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point
	• Single-precision floating point
	• Boolean
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Flag	• Boolean

Basic Operation

When you do not select the **Running mean** check box, the block computes the mean value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time. Each element in the output array y is the mean value of the corresponding column, row, vector, or entire input. The output array, y, depends on the setting of the **Find the mean**

value over parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Entire input` — The output at each sample time is a scalar that contains the mean value of the *M*-by-*N*-by-*P* input matrix.

  ```
  y = mean(u(:))     % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the mean value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector.

  ```
  y = mean(u,2)     % Equivalent MATLAB code
  ```

- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the mean value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector.

  ```
  y = mean(u)     % Equivalent MATLAB code
  ```

  In this mode, the block treats length-*M* unoriented vector inputs as *M*-by-1 column vectors.

- `Specified dimension` — The output at each sample time depends on the value of the **Dimension** parameter. If you set the **Dimension** to 1, the output is the same as when you select `Each column`. If you set the **Dimension** to 2, the output is the same as when you select `Each row`. If you set the **Dimension** to 3, the output at each sample time is an *M*-by-*N* matrix containing the mean value of each vector over the third dimension of the input.

  ```
  y = mean(u,Dimension)     % Equivalent MATLAB code
  ```

The mean of a complex input is computed independently for the real and imaginary components, as shown in the following figure.
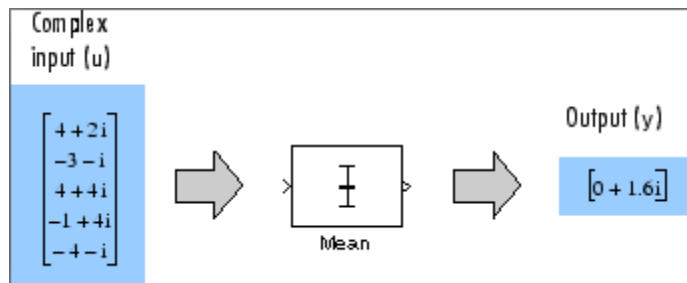
Running Operation

When you select the **Running mean** check box, the block tracks the mean value of each channel in a time sequence of inputs. In this mode, the block treats each element as a channel.

Resetting the Running Mean

The block resets the running mean whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

When a reset event occurs, the running mean for each channel is initialized to the value in the corresponding channel of the current input.

You specify the reset event by the **Reset port** parameter:

- None disables the Rst port.

- Rising edge — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

Rising edge

Rising edge

Not a rising edge because it is
a continuation of a rise from
a negative value to zero.

Rising edge   Rising edge

- Falling edge — Triggers a reset operation when the Rst input does
  one of the following:

  - Falls from a positive value to a negative value or zero

  - Falls from zero to a negative value, where the fall is not a
    continuation of a fall from a positive value to zero (see the following
    figure)

Falling edge   Falling edge

Not a falling edge because it is
a continuation of a fall from
a positive value to zero.

Falling edge

Falling edge

- Either edge — Triggers a reset operation when the Rst input is a
  Rising edge or Falling edge (as described earlier)

- Non-zero sample — Triggers a reset operation at each sample time
  that the Rst input is not zero

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the mean value over** parameter is set to Entire input and the **Running mean** check box is not selected. ROI processing is only supported for 2-D inputs.

- A binary mask is a binary image that enables you to specify which pixels to highlight, or select.

- In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to Label matrix, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix.

- For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes reference page.

For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

> **Note** For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select Binary mask.

If, for the **ROI type** parameter, you select Rectangles or Lines, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select Label matrix, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The following diagram shows the data types used within the Mean block for fixed-point signals.

2-D Mean
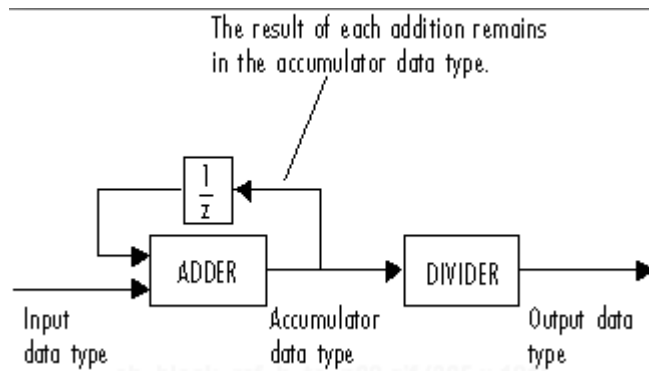
The result of each addition remains in the accumulator data type.

Input data type — Accumulator data type — Output data type

You can set the accumulator and output data types in the block dialog, as discussed in "Dialog Box" on page 1-128.

Dialog Box

The **Main** pane of the Mean block dialog appears as follows.
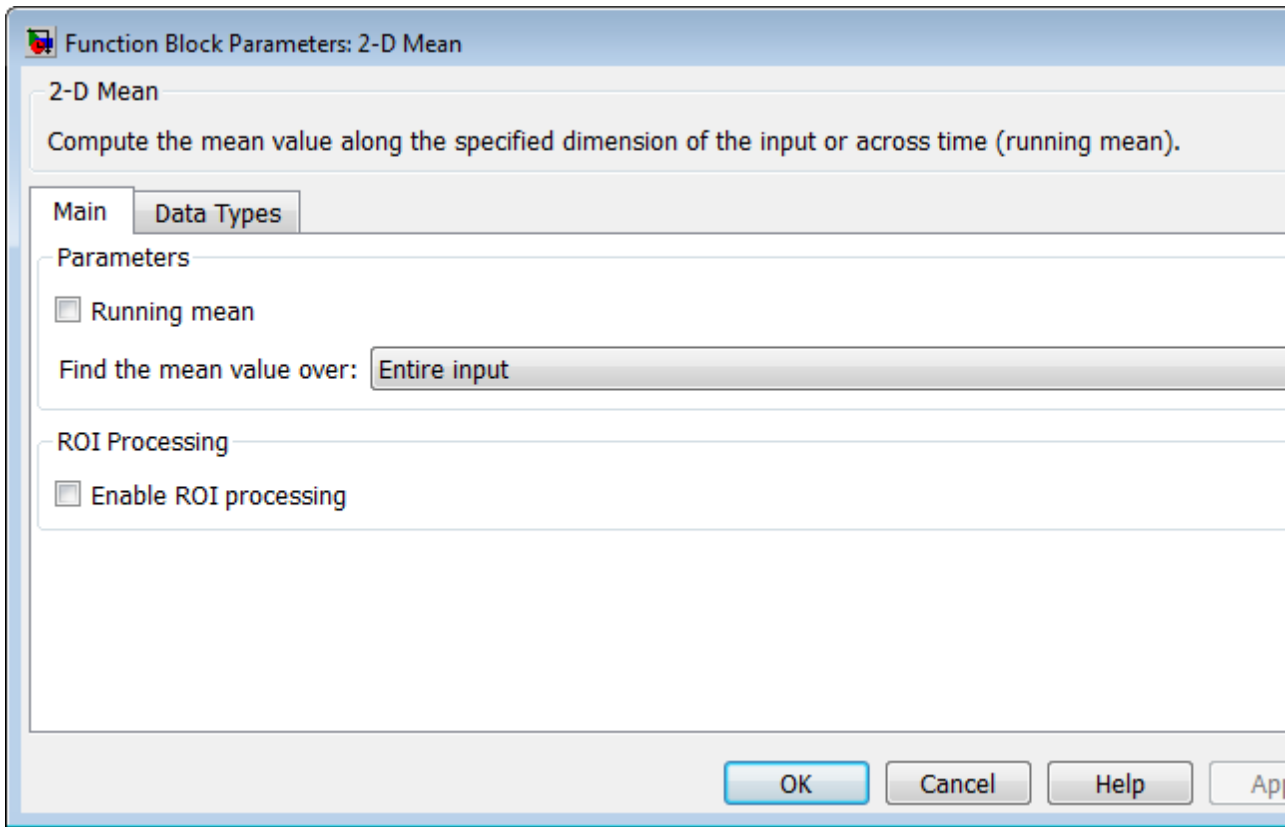
Running mean

Enables running operation when selected.

Reset port

Specify the reset event that causes the block to reset the running mean. The sample time of the input to the Rst port must be a positive integer multiple of the input sample time. This parameter appears only when you select the **Running mean** check box. For more information, see "Resetting the Running Mean" on page 1-123.

Find the mean value over

    Specify whether to find the mean value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 1-121.

Dimension

    Specify the dimension (one-based value) of the input signal, over which the mean is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the mean value over** parameter is set to Specified dimension.

Enable ROI Processing

    Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the mean value over** parameter is set to Entire input, and the block is not in running mode.

ROI type

    Specify the type of ROI you want to use. Your choices are Rectangles, Lines, Label matrix, or Binary mask.

ROI portion to process

    Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify Rectangles.

Output

    Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select Binary mask.

Output flag

    ☑ **Output flag indicating if ROI is within image bounds**

    ☑ **Output flag indicating if label numbers are valid**

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 1-125.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select Rectangles or Lines as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select Label matrix for the **ROI type** parameter.

The **Data Types** pane of the Mean block dialog appears as follows.

2-D Mean

Rounding mode
> Select the "Rounding Modes" for fixed-point operations.

Overflow mode
> Select the Overflow mode for fixed-point operations.

Accumulator data type
> Specify the accumulator data type. See "Fixed-Point Data Types"
> on page 1-127 for illustrations depicting the use of the accumulator
> data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 1-127 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Signal Ranges")

- Automatic scaling of fixed-point data types

2-D Mean

Maximum

Specify the maximum value that the block should output. The default value, [], is equivalent to Inf. Simulink software uses this value to perform:

- Simulation range checking (see "Signal Ranges")
- Automatic scaling of fixed-point data types
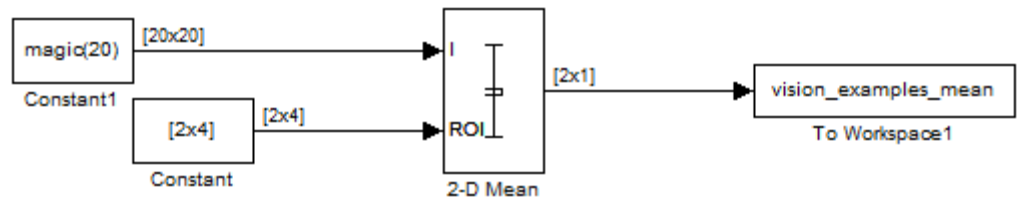
Example

The ex_vision_2dmean calculates the mean value within two ROIs.

See Also

2-D Maximum	Computer Vision System Toolbox
2D-Median	Computer Vision System Toolbox
2-D Minimum	Computer Vision System Toolbox
2-D Standard Deviation	Computer Vision System Toolbox
mean	MATLAB

Purpose   Find mean value of each input matrix

Library   Statistics

Description
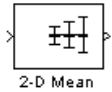
2-D Mean

Note This 2-D Mean block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated 2-D Mean block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the 2-D Mean documentation to view the reference material related to the obsolete block.

2-D Median

Purpose	Find 2-D Median value of each input matrix
Library	Statistics
	`visionstatistics`

Description

The 2-D Median block computes the median value of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The median of a set of input values is calculated as follows:

1 The values are sorted.

2 If the number of values is odd, the median is the middle value.

3 If the number of values is even, the median is the average of the two middle values.

For a given input *u*, the size of the output array y depends on the setting of the **Find the median value over** parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Entire input` — The output at each sample time is a scalar that contains the median value of the *M*-by-*N*-by-*P* input matrix.

  ```
  y = median(u(:)) % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the median value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output is an *M*-by-1 column vector.

  ```
  y = median(u,2)  % Equivalent MATLAB code
  ```

- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the median value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector.

```
y = median(u)   % Equivalent MATLAB code
```

For convenience, length-*M* 1-D vector inputs are treated as *M*-by-1 column vectors when the block is in this mode. Sample-based length-*M* row vector inputs are also treated as *M*-by-1 column vectors.

- Specified dimension — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select Each column. If **Dimension** is set to 2, the output is the same as when you select Each row. If **Dimension** is set to 3, the output at each sample time is an *M*-by-*N* matrix containing the median value of each vector over the third dimension of the input.

```
y = median(u,Dimension)   % Equivalent MATLAB code
```

The block sorts complex inputs according to their magnitude.

Fixed-Point Data Types

For fixed-point inputs, you can specify accumulator, product output, and output data types as discussed in "Dialog Box" on page 1-138. Not all these fixed-point parameters are applicable for all types of fixed-point inputs. The following table shows when each kind of data type and scaling is used.

	Output data type	Accumulator data type	Product output data type
Even *M*	X	X	
Odd *M*	X		
Odd *M* and complex	X	X	X
Even *M* and complex	X	X	X

The accumulator and output data types and scalings are used for fixed-point signals when *M* is even. The result of the sum performed while calculating the average of the two central rows of the input matrix

2-D Median

is stored in the accumulator data type and scaling. The total result of the average is then put into the output data type and scaling.

The accumulator and product output parameters are used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before the input elements are sorted, as described in Description. The results of the squares of the real and imaginary parts are placed into the product output data type and scaling. The result of the sum of the squares is placed into the accumulator data type and scaling.

For fixed-point inputs that are both complex and have even M, the data types are used in all of the ways described. Therefore, in such cases, the accumulator type is used in two different ways.

Dialog Box

The **Main** pane of the 2–D Median block dialog appears as follows.
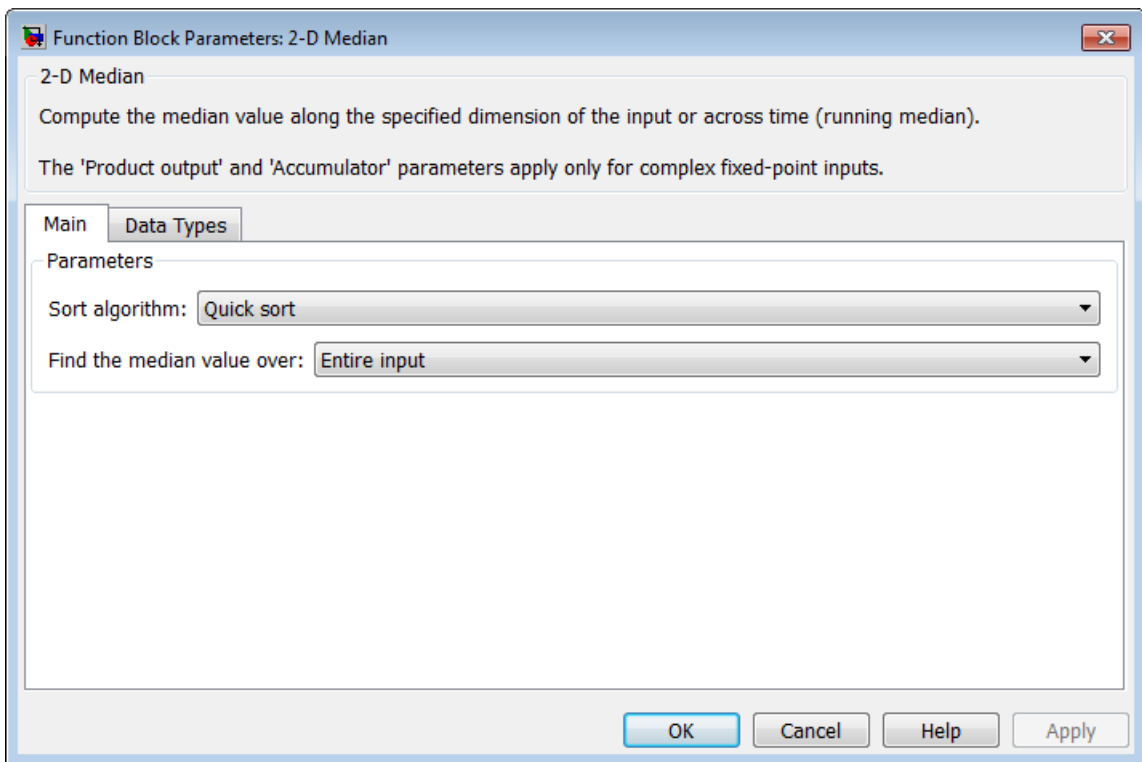
Sort algorithm

Specify whether to sort the elements of the input using a `Quick sort` or an `Insertion sort` algorithm.

Find the median value over

Specify whether to find the median value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see Description.

Dimension

Specify the dimension (one-based value) of the input signal, over which the median is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This

parameter is only visible when the **Find the median value over** parameter is set to `Specified dimension`.

The **Data Types** pane of the 2-D Median block dialog appears as follows.
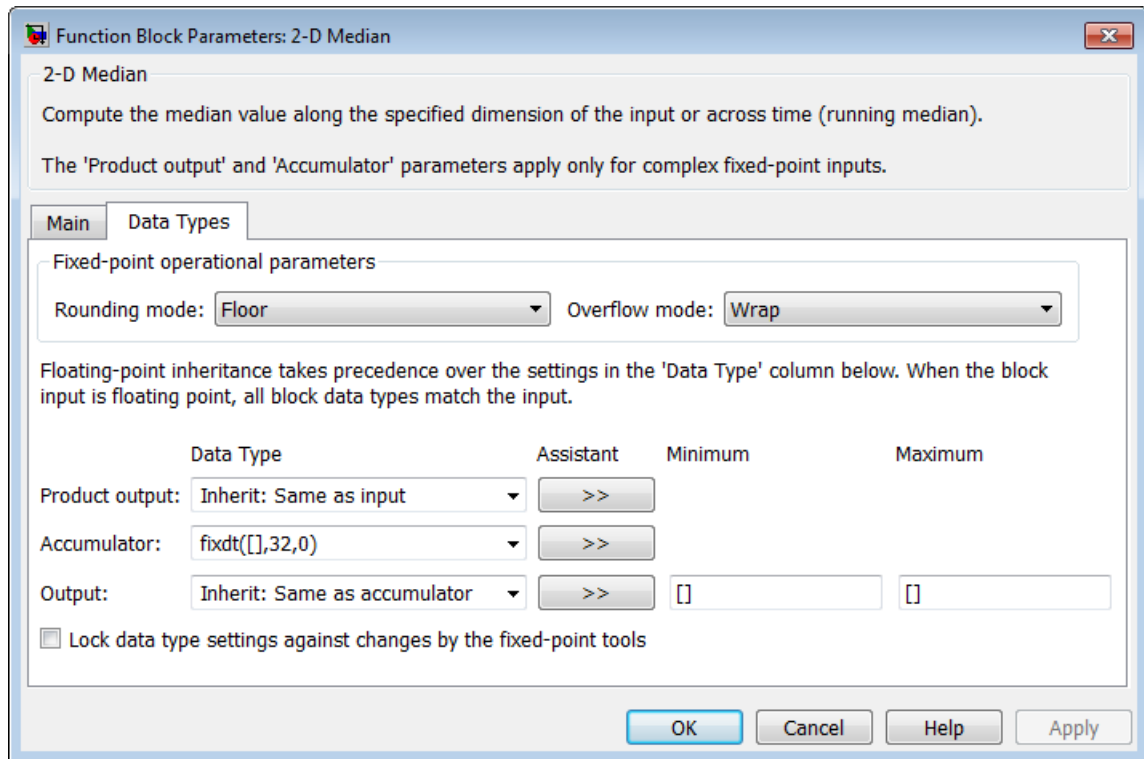
Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode

Select the Rounding mode for fixed-point operations.
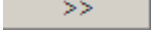
Overflow mode

Select the Overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 1-137 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 1-137 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button [>>] to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.

Output data type
> Specify the output data type. See "Fixed-Point Data Types" on page 1-137 for illustrations depicting the use of the output data type in this block. You can set it to:
>
> - A rule that inherits a data type, for example, `Inherit:  Same as accumulator`
>
> - An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`
>
> Click the **Show data type assistant** button `>>` to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.
>
> See "Specify Data Types Using Data Type Assistant" for more information.

Minimum
> Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:
>
> - Simulation range checking (see "Signal Ranges")
>
> - Automatic scaling of fixed-point data types

Maximum
> Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:
>
> - Simulation range checking (see "Signal Ranges")
>
> - Automatic scaling of fixed-point data types

Port	Supported Data Types
Input	• Double-precision floating point
	• Single-precision floating point
	• Fixed point (signed and unsigned)
	• 8-, 16-, 32-, and 128-bit signed integers
	• 8-, 16-, 32-, and 128-bit unsigned integers
Output	• Double-precision floating point
	• Single-precision floating point
	• Fixed point (signed and unsigned)
	• 8-, 16-, 32-, and 128-bit signed integers
	• 8-, 16-, 32-, and 128-bit unsigned integers

Supported Data Types (row label for table above)

Examples

Calculate Median Value Over Entire Input

The ex_vision_2dmedian calculates the median value over the entire input.

See Also

2-D Maximum	Computer Vision System Toolbox
2-D Mean	Computer Vision System Toolbox
2-D Minimum	Computer Vision System Toolbox

2-D Median

2-D Standard Deviation	Computer Vision System Toolbox
2-D Variance	Computer Vision System Toolbox
median	MATLAB

Purpose   Find minimum values in input or sequence of inputs

Library   Statistics

visionstatistics
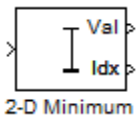
Description   The 2-D Minimum block identifies the value and/or position of the
smallest element in each row or column of the input, or along a specified
dimension of the input. The 2-D Minimum block can also track the
minimum values in a sequence of inputs over a period of time.

The 2-D Minimum block supports real and complex floating-point,
fixed-point, and Boolean inputs. Real fixed-point inputs can be either
signed or unsigned, while complex fixed-point inputs must be signed.
The output data type of the minimum values match the data type of
the input. The block outputs double index values, when the input is
double, and uint32 otherwise.

Port Description

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Scalar, vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
Rst	Scalar value	Boolean	No

Port	Input/Output	Supported Data Types	Complex Values Supported
Val	Minimum value output based on the "Value Mode" on page 1-146	Same as Input port	Yes
Idx	One-based output location of the minimum value based on the "Index Mode" on page 1-147	Same as Input port	No

Length-M 1-D vector inputs are treated as M-by-1 column vectors.

Value Mode

When you set the **Mode** parameter to Value, the block computes the minimum value in each row, column, entire input, or over a specified dimension. The block outputs each element as the minimum value in the corresponding column, row, vector, or entire input. The output depends on the setting of the **Find the minimum value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- Each row — The output at each sample time consists of an M-by-1-by-P array, where each element contains the minimum value of each vector over the second dimension of the input. For an M-by-N input matrix, the block outputs an M-by-1 column vector at each sample time.

- Each column — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the minimum value of each vector over the first dimension of the input. For an M-by-N input matrix, the block outputs a 1-by-N row vector at each sample time.

  In this mode, the block treats length-M unoriented vector inputs as M-by-1 column vectors.

- `Entire input` — The output at each sample time is a scalar that contains the minimum value in the *M*-by-*N*-by-*P* input matrix.

- `Specified dimension` — The output at each sample time depends on **Dimension**. When you set **Dimension** to 1, the block output is the same as when you select `Each column`. When you set **Dimension** to 2, the block output is the same as when you select `Each row`. When you set **Dimension** to 3, the block outputs an *M*-by-*N* matrix containing the minimum value of each vector over the third dimension of the input, at each sample time.

For complex inputs, the block selects the value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input that has the minimum magnitude squared as shown below. For complex value $u = a + bi$, the magnitude squared is $a^2 + b^2$.
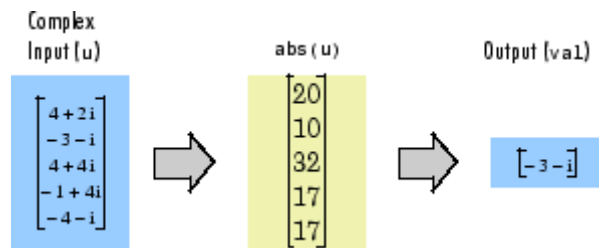
Index Mode

When you set the **Mode** parameter to `Index`, the block computes the minimum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input, and outputs the index array *I*. Each element in *I* is an integer indexing the minimum value in the corresponding column, row, vector, or entire input. The output *I* depends on the setting of the **Find the minimum value over** parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the index of the

minimum value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector.

- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the index of the minimum value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector.

  In this mode, the block treats length-*M* unoriented vector inputs as *M*-by-1 column vectors.

- `Entire input` — The output at each sample time is a 1-by-3 vector that contains the location of the minimum value in the *M*-by-*N*-by-*P* input matrix. For an input that is an *M*-by-*N* matrix, the output will be a 1-by-2 vector of one-based [x y] location coordinates for the minimum value.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output at each sample time is an *M*-by-*N* matrix containing the indices of the minimum values of each vector over the third dimension of the input.

When a minimum value occurs more than once, the computed index corresponds to the first occurrence. For example, when the input is the column vector `[-1 2 3 2 -1]'`, the computed one-based index of the minimum value is `1` rather than `5` when `Each column` is selected.

Value and Index Mode

When you set the **Mode** parameter to `Value and Index`, the block outputs both the minima, and the indices.

Running Mode

When you set the **Mode** parameter to `Running`, the block tracks the minimum value of each channel in a time sequence of *M*-by-*N* inputs. In this mode, the block treats each element as a channel.

Resetting the Running Minimum

The block resets the running minimum whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

You specify the reset event by the **Reset port** parameter:

- `None` — Disables the Rst port

- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:

  - Falls from a positive value to a negative value or zero

- Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

Falling edge   Falling edge

Falling edge   Falling edge   **Not a falling edge** because it is a continuation of a fall from a positive value to zero.

- Either edge — Triggers a reset operation when the Rst input is a Rising edge or Falling edge (as described above)

- Non-zero sample — Triggers a reset operation at each sample time that the Rst input is not zero

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This applies to any mode other than the running mode and when you set the **Find the minimum value over** parameter to Entire input and you select the **Enable ROI processing** check box. ROI processing applies only for 2-D inputs.

You can specify a rectangle, line, label matrix, or binary mask ROI type.

Use the binary mask to specify which pixels to highlight or select.

Use the label matrix to label regions. Pixels set to 0 represent the background. Pixels set to 1 represent the first object, pixels set to 2, represent the second object, and so on. Use the **Label Numbers** port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter. For more information about the format of the input to the ROI port when you set the ROI to a rectangle or a line, see the Draw Shapes block reference page.

ROI Output Statistics
Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Fixed-point** pane of the dialog box are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 1-146. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Dialog Box

The **Main** pane of the 2-D Minimum dialog box appears as shown in the following figure.
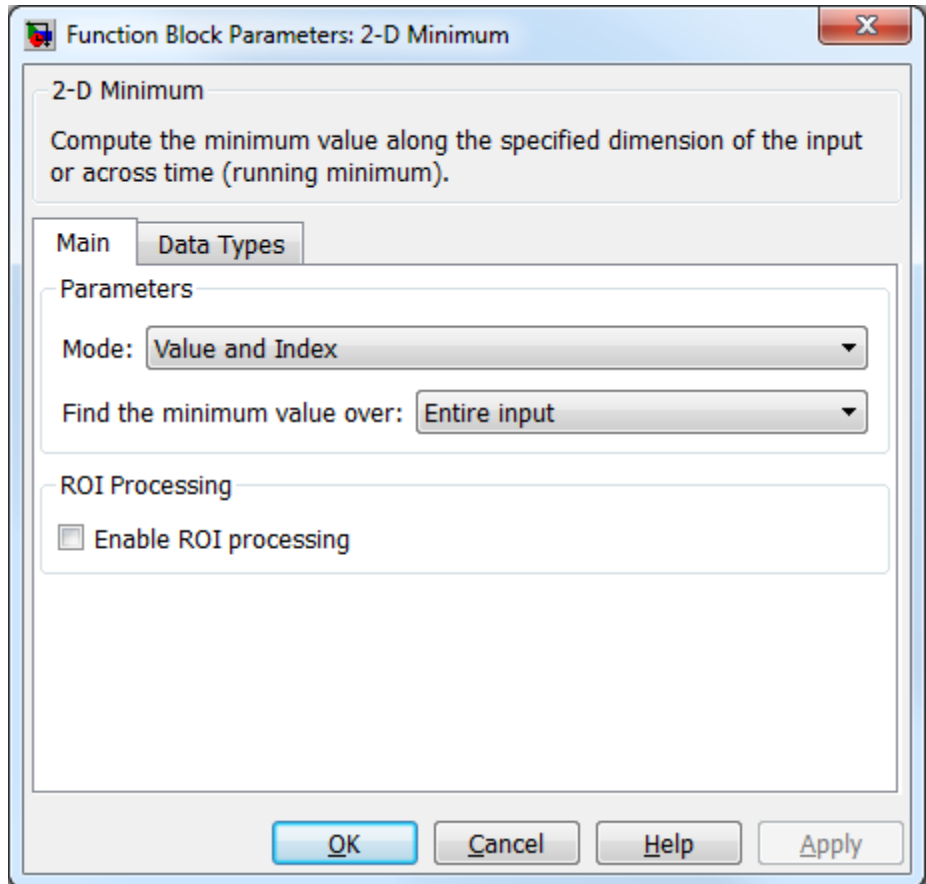
Mode

Specify the block's mode of operation:

- Value and Index — Output both the value and the index location.

- `Value` — Output the minimum value of each input matrix. For more information, see "Value Mode" on page 1-146

- `Index`— Output the one-based index location of the minimum value. For more information, see "Index Mode" on page 1-147

- `Running` — Track the minimum value of the input sequence over time. For more information, see "Running Mode" on page 1-149.

For the `Value`, `Index`, and `Value and Index` modes, the 2-D Minimum block produces identical results as the MATLAB `min` function when it is called as [y I] = min(u,[],D), where *u* and *y* are the input and output, respectively, *D* is the dimension, and *I* is the index.

Find the minimum value over
   Specify whether the block should find the minimum of the entire input each row, each column, or dimensions specified by the **Dimension** parameter.

Reset port
   Specify the reset event that causes the block to reset the running minimum. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter appears only when you set the **Mode** parameter to `Running`. For information about the possible values of this parameter, see "Resetting the Running Minimum" on page 1-149.

Dimension
   Specify the dimension (one-based value) of the input signal, over which the minimum is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter applies only when you set the **Find the minimum value over** parameter to `Specified dimension`.

Enable ROI processing
   Select this check box to calculate the statistical value within a particular region of each image. This parameter applies only

when you set the **Find the minimum value over** parameter to Entire input, and the block is not in running mode.

ROI type

Specify the type of ROI you want to use. Your choices are Rectangles, Lines, Label matrix, or Binary mask.

When you set this parameter to Rectangles or Lines, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the **Flag** port appears on the block.

When you set this parameter to Label matrix, the **Label** and **Label Numbers** ports appear on the block and the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block.

See Output = Individual statistics for each ROI on page 1-151 for details.

ROI portion to process

Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter applies only when you set the **ROI type** parameter to Rectangles.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter does not apply when you set the **ROI type** parameter, to Binary mask.

Output flag indicating if ROI is within image bounds

When you select this check box, the **Flag** port appears on the block. This check box applies only when you set the **ROI type** parameter to Rectangles or Lines. For a description of the **Flag** port output, see the tables in "ROI Processing" on page 1-150.

Output flag indicating if label numbers are valid
When you select this check box, the **Flag** port appears on the block. This check box applies only when you set the **ROI type** parameter to Label matrix. For a description of the **Flag** port output, see the tables in "ROI Processing" on page 1-150.

The **Fixed-point** pane of the 2-D Minimum dialog box appears as shown in the following figure.

Note The parameters on the **Data Types** pane are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 1-146. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Rounding mode

Select the "Rounding Modes" for fixed-point operations.

Overflow mode

Select the Overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 1-152 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit:  Same as input`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button ⟩⟩ to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.
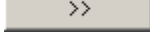
See "Specify Data Types Using Data Type Assistant" for more information.

2-D Minimum

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 1-152 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Specify Data Types Using Data Type Assistant" for more information.
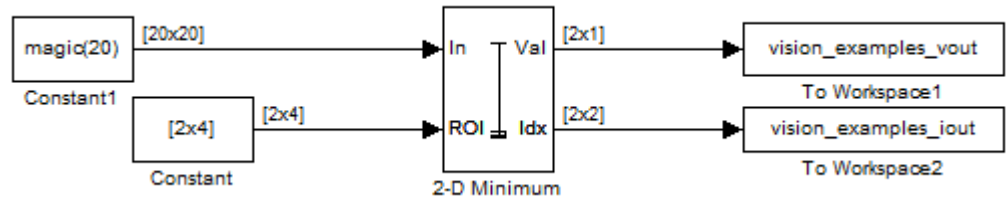
Examples

The ex_vision_2dminimum example finds the minimum value within two ROIs. The model outputs the minimum values and their one-based [x y] coordinate locations.

See Also

2-D Maximum	Computer Vision System Toolbox
2-D Mean	Computer Vision System Toolbox

MinMax	Simulink
2D-Histogram	Computer Vision System Toolbox
min	MATLAB

2-D Minimum (To Be Removed)

Purpose   Find minimum values in an input or sequence of inputs

Library   vipobslib

Description
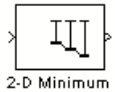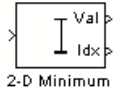
Note This 2-D Minimum block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated 2-D Minimum block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the 2-D Minimum documentation to view the reference material related to the obsolete block.

Purpose   Find standard deviation of each input matrix

Library   Statistics

visionstatistics

Description   The Standard Deviation block computes the standard deviation of each
row or column of the input, along vectors of a specified dimension of the
input, or of the entire input. The Standard Deviation block can also
track the standard deviation of a sequence of inputs over a period of
time. The **Running standard deviation** parameter selects between
basic operation and running operation.

Port Description

Port	Supported Data Types
Input	• Double-precision floating point
	• Single-precision floating point
Reset	• Double-precision floating point
	• Single-precision floating point
	• Boolean
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers

2-D Standard Deviation

Port	Supported Data Types
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point
Flag	• Boolean

Basic Operation

When you do not select the **Running standard deviation** check box, the block computes the standard deviation of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time, and outputs the array y. Each element in y contains the standard deviation of the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the standard deviation value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

• Entire input — The output at each sample time is a scalar that contains the standard deviation of the entire input.

```
y = std(u(:))      % Equivalent MATLAB code
```

- `Each Row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the standard deviation of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector.

```
y = std(u,0,2)     % Equivalent MATLAB code
```

- `Each Column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the standard deviation of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector.

```
y = std(u,0,1)     % Equivalent MATLAB code
```

In this mode, the block treats length-*M* unoriented vector inputs as *M*-by-1 column vectors.

- `Specified Dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to **1**, the output is the same as when you select `Each column`. If **Dimension** is set to **2**, the output is the same as when you select `Each row`. If **Dimension** is set to **3**, the output at each sample time is an *M*-by-*N* matrix containing the standard deviation of each vector over the third dimension of the input.

```
y = std(u,0,Dimension)     % Equivalent MATLAB code
```

For purely real or purely imaginary inputs, the standard deviation of the *j*th column of an *M*-by-*N* input matrix is the square root of its variance:

$$y_j = \sigma_j = \sqrt{\frac{\sum_{i=1}^{M} |u_{ij} - \mu_j|^2}{M-1}} \qquad 1 \le j \le N$$

2-D Standard Deviation

For complex inputs, the output is the *total standard deviation*, which equals the square root of the *total variance*, or the square root of the sum of the variances of the real and imaginary parts. The standard deviation of each column in an *M*-by-*N* input matrix is given by:

$$\sigma_j = \sqrt{\sigma_{j,\text{Re}}^2 + \sigma_{j,\text{Im}}^2}$$

Note The total standard deviation does *not* equal the sum of the real and imaginary standard deviations.

Running Operation

When you select the **Running standard deviation** check box, the block tracks the standard deviation of successive inputs to the block. In this mode, the block treats each element as a channel.

Resetting the Running Standard Deviation

The block resets the running standard deviation whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

You specify the reset event in the **Reset port** parameter:

- None disables the Rst port.

- Rising edge — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:

  - Falls from a positive value to a negative value or zero

  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described earlier)

- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

2-D Standard Deviation

> **Note** When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the standard deviation value over** parameter is set to Entire input and the **Running standard deviation** check box is not selected. ROI processing is only supported for 2-D inputs.

Use the **ROI type** parameter to specify whether the ROI is a rectangle, line, label matrix, or binary mask. A binary mask is a binary image that enables you to specify which pixels to highlight, or select. In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to Label matrix, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix. For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select Binary mask.

If, for the **ROI type** parameter, you select Rectangles or Lines, the **Output flag indicating if ROI is within image bounds** check box

appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select Label matrix, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Dialog Box

Compute the standard deviation along the specified dimension of the input or across time (running standard deviation).

Parameters

☐ Running standard deviation

Find the standard deviation value over: Entire input ▼

ROI Processing

☐ Enable ROI processing

Running standard deviation
> Enables running operation when selected.

Reset port
> Specify the reset event that causes the block to reset the running standard deviation. The sample time of the input to the Rst port must be a positive integer multiple of the input sample time. This parameter appears only when you select the **Running standard deviation** check box. For more information, see "Resetting the Running Standard Deviation" on page 1-164.

Find the standard deviation value over
> Specify whether to find the standard deviation value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 1-162.

Dimension

>Specify the dimension (one-based value) of the input signal, over which the standard deviation is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the standard deviation value over** parameter is set to `Specified dimension`.

Enable ROI Processing

>Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the standard deviation value over** parameter is set to `Entire input`, and the block is not in running mode.

ROI type

>Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process

>Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output

>Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

>☑ **Output flag indicating if ROI is within image bounds**

>☑ **Output flag indicating if label numbers are valid**

>When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 1-166.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select Rectangles or Lines as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select Label matrix for the **ROI type** parameter.
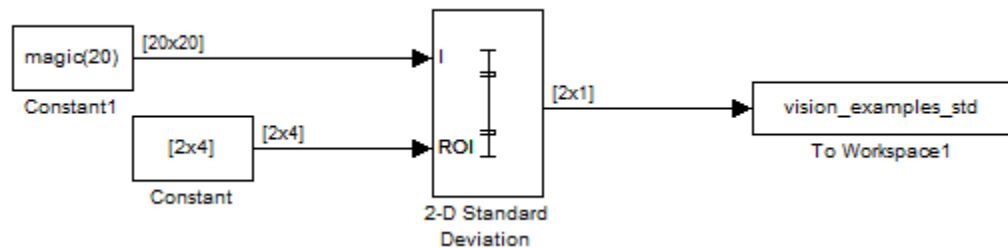
Example

The ex_vision_2dstd calculates the standard deviation value within two ROIs.

See Also

2-D Mean	Computer Vision System Toolbox
2-D Variance	Computer Vision System Toolbox
std	MATLAB

2-D Standard Deviation (To Be Removed)

Purpose   Find standard deviation of each input matrix

Library   Statistics

Description
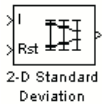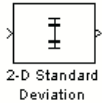
Note This 2-D Standard Deviation block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated 2-D Standard Deviation block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the 2-D Standard Deviation documentation to view the reference material related to the obsolete block.

Purpose   Compute variance of input or sequence of inputs

Library   Statistics

                   visionstatistics

Description   The 2-D Variance block computes the unbiased variance of each row
                   or column of the input, along vectors of a specified dimension of the
                   input, or of the entire input. The 2-D Variance block can also track the
                   variance of a sequence of inputs over a period of time. The **Running
                   variance** parameter selects between basic operation and running
                   operation.

Port Description

Port	Supported Data Types
Input	• Double-precision floating point
	• Single-precision floating point
	• Fixed point
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point
	• Single-precision floating point
	• Boolean
	• 8-, 16-, and 32-bit signed integers
	• 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Flag	• Boolean

Basic Operation

When you do not select the **Running variance** check box, the block computes the variance of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time, and outputs the array y. Each element in y is the variance of the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the variance value**

over parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the variance of the entire input.

  ```
  y = var(u(:))     % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the variance of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector.

  ```
  y = var(u,0,2)    % Equivalent MATLAB code
  ```

- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the variance of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector.

  ```
  y = var(u,0,1)    % Equivalent MATLAB code
  ```

  In this mode, the block treats length-M unoriented vector inputs as M-by-1 column vectors.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as that when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output at each sample time is an M-by-N matrix containing the variance of each vector over the third dimension of the input.

  ```
  y = var(u,0,Dimension)     % Equivalent MATLAB code
  ```

For purely real or purely imaginary inputs, the variance of an M-by-N matrix is the square of the standard deviation:

2-D Variance

$$y = \sigma^2 = \frac{\displaystyle\sum_{i=1}^{M}\sum_{j=1}^{N}\left|u_{ij}\right|^2 - \frac{\left|\displaystyle\sum_{i=1}^{M}\sum_{j=1}^{N}u_{ij}\right|^2}{M*N}}{M*N-1}$$

For complex inputs, the variance is given by the following equation:
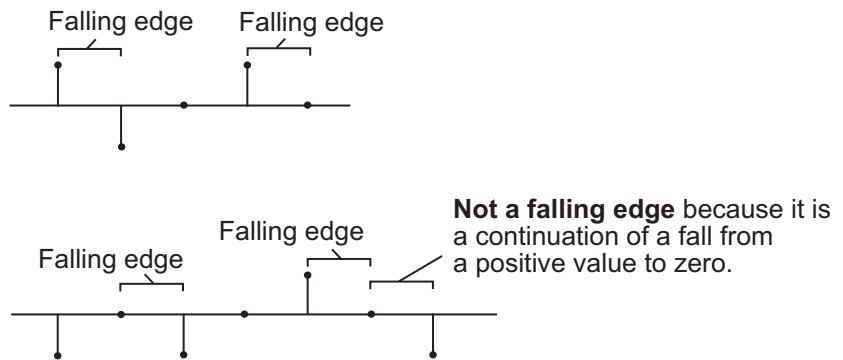
$$\sigma^2 = {\sigma_{Re}}^2 + {\sigma_{Im}}^2$$

Running Operation

When you select the **Running variance** check box, the block tracks the variance of successive inputs to the block. In this mode, the block treats each element as a channel.

Resetting the Running Variance

The block resets the running variance whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

You specify the reset event in the **Reset port** parameter:

- None disables the Rst port.

- Rising edge — Triggers a reset operation when the Rst input does one of the following:

  - Rises from a negative value to a positive value or zero

  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)

- `Falling edge` — Triggers a reset operation when the Rst input does one of the following:

  - Falls from a positive value to a negative value or zero

  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

- `Either edge` — Triggers a reset operation when the Rst input is a `Rising edge` or `Falling edge` (as described earlier)

- `Non-zero sample` — Triggers a reset operation at each sample time that the Rst input is not zero

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the variance value over**

parameter is set to `Entire input` and the **Running variance** check box is not selected. ROI processing is only supported for 2-D inputs.

Use the **ROI type** parameter to specify whether the ROI is a binary mask, label matrix, rectangle, or line. ROI processing is only supported for 2-D inputs.

- A binary mask is a binary image that enables you to specify which pixels to highlight, or select.

- In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix.

- For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes reference page.

Note For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

2-D Variance

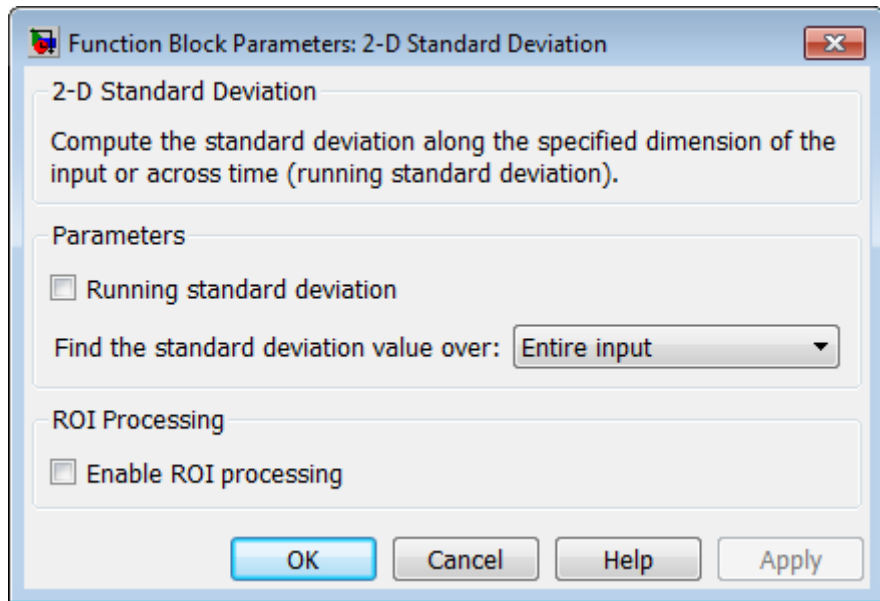### Output = Individual Statistics for Each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single Statistic for All ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select Label matrix, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual Statistics for Each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single Statistic for All ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Data Types** pane of the block dialog are only used for fixed-point inputs. For purely real or purely imaginary inputs, the variance of the input is the square of its standard deviation. For complex inputs, the output is the sum of the variance of the real and imaginary parts of the input.

The following diagram shows the data types used within the Variance block for fixed-point signals.
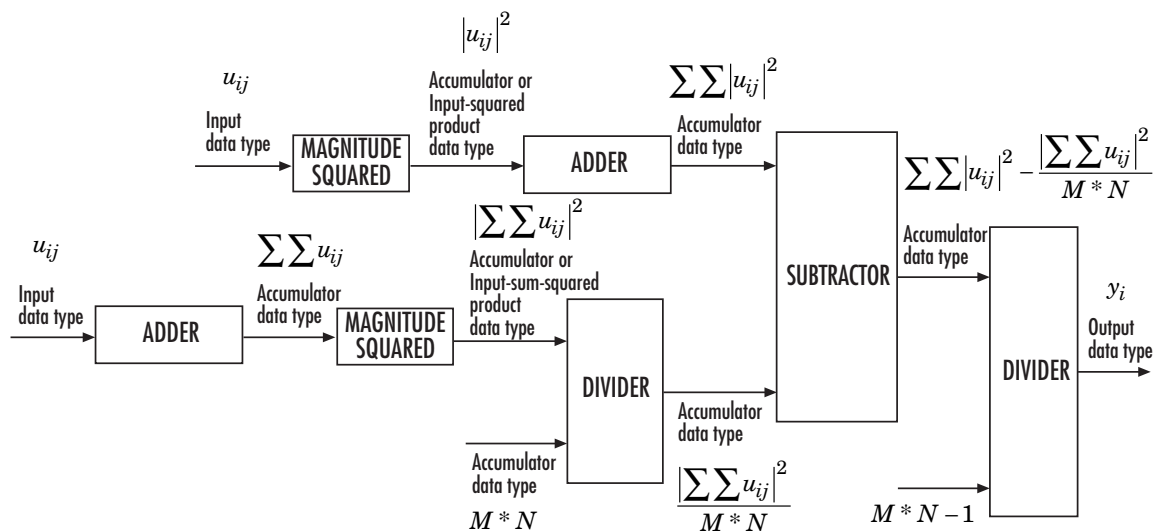
2-D Variance

The results of the magnitude-squared calculations in the figure are in the product output data type. You can set the accumulator, product output, and output data types in the block dialog as discussed in "Dialog Box" on page 1-183.

Dialog Box

The **Main** pane of the Variance block dialog appears as follows.
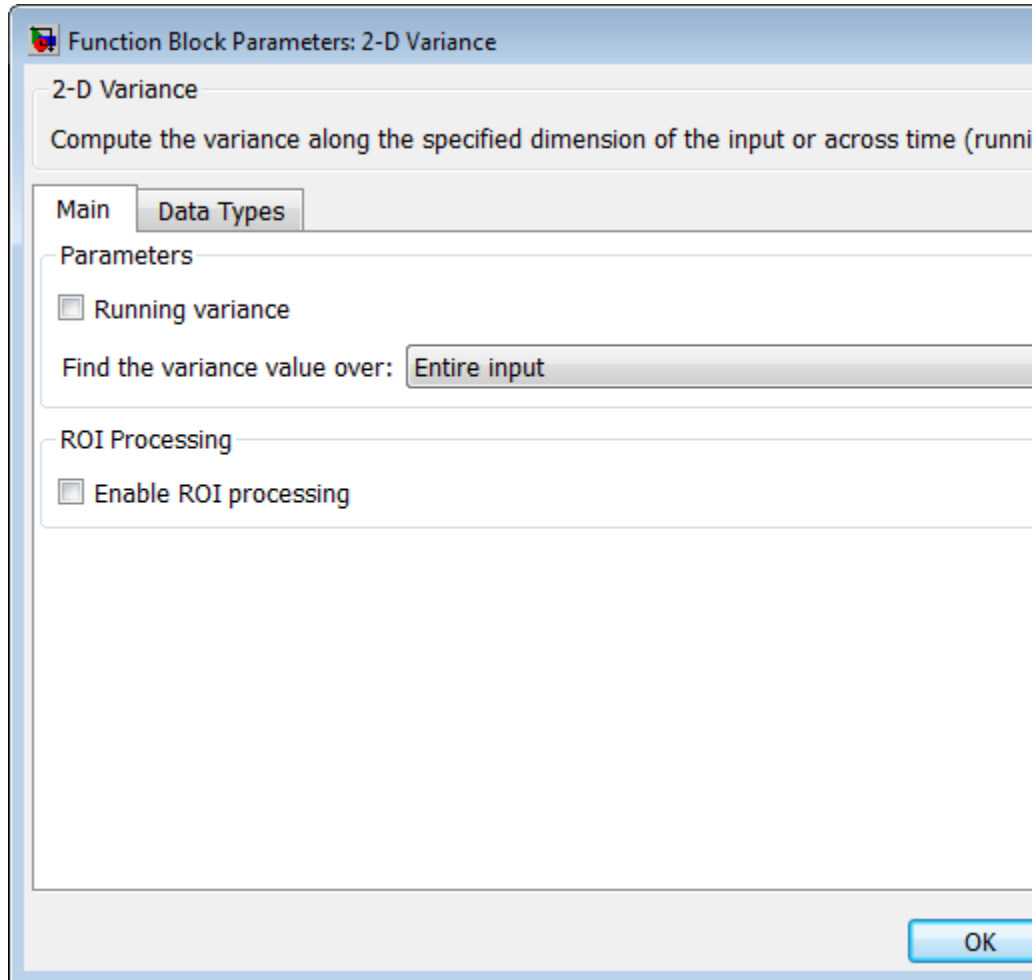
Running variance

Enables running operation when selected.

2-D Variance

Reset port

Specify the reset event that causes the block to reset the running variance. The sample time of the input to the Rst port must be a positive integer multiple of the input sample time. This parameter appears only when you select the **Running variance** check box. For more information, see "Resetting the Running Variance" on page 1-177

Find the variance value over

Specify whether to find the variance along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 1-174.

Dimension

Specify the dimension (one-based value) of the input signal, over which the variance is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the variance value over** parameter is set to `Specified dimension`.

Enable ROI Processing

Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the variance value over** parameter is set to `Entire input`, and the block is not in running mode.

Note Full ROI processing is available only if you have a Computer Vision System Toolbox license. If you do not have a Computer Vision System Toolbox license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

ROI type

Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process

> Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output

> Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

> ☑ **Output flag indicating if ROI is within image bounds**
>
> ☑ **Output flag indicating if label numbers are valid**
>
> When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 1-178.
>
> The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.
>
> The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

The **Data Types** pane of the Variance block dialog appears as follows.

2-D Variance
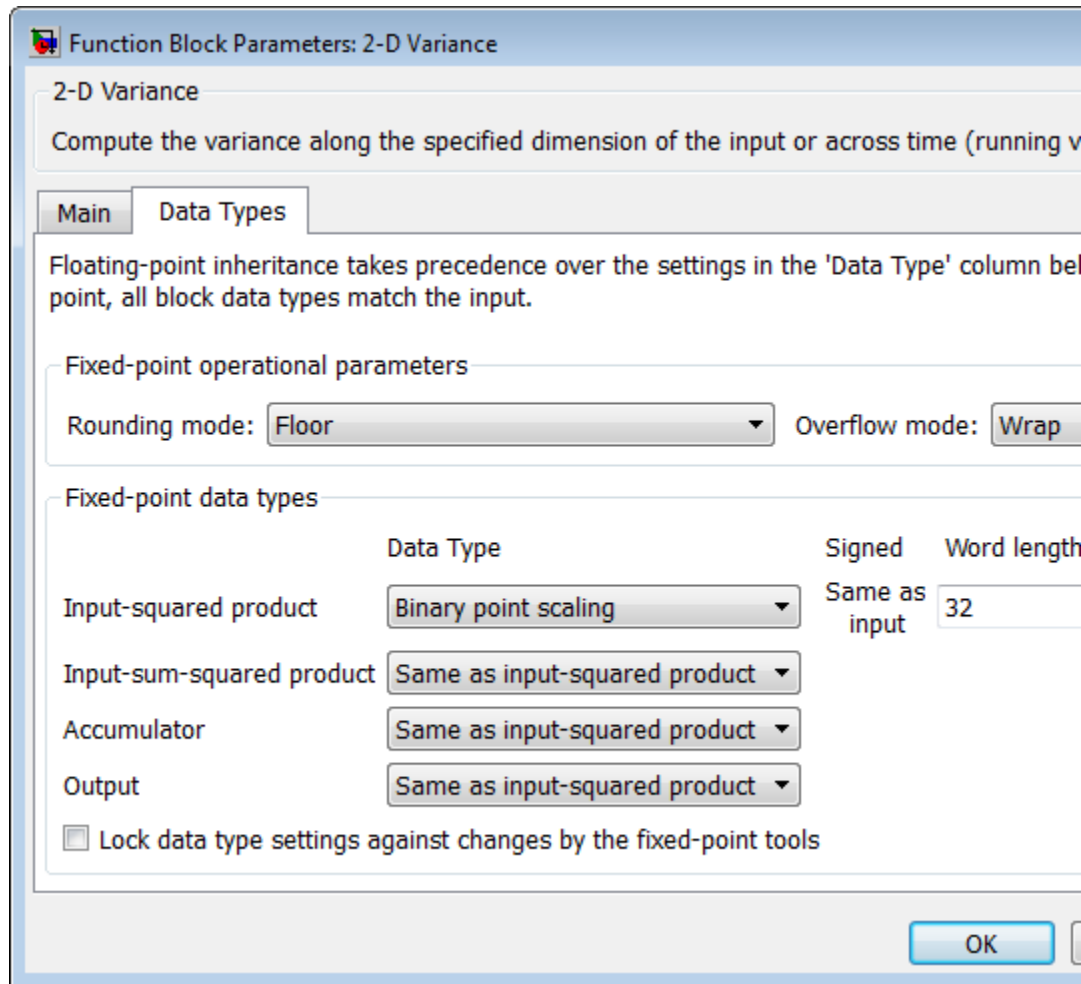
Rounding mode
    Select the "Rounding Modes" for fixed-point operations.

Overflow mode
    Select the Overflow mode for fixed-point operations.

> **Note** See "Fixed-Point Data Types" on page 1-181 for more information on how the product output, accumulator, and output data types are used in this block.

Input-squared product

Use this parameter to specify how to designate the input-squared product word and fraction lengths:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the input-squared product, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the input-squared product. This block requires power-of-two slope and a bias of zero.

Input-sum-squared product

Use this parameter to specify how to designate the input-sum-squared product word and fraction lengths:

- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the input-sum-squared product, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the input-sum-squared product. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block:

- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output
Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.
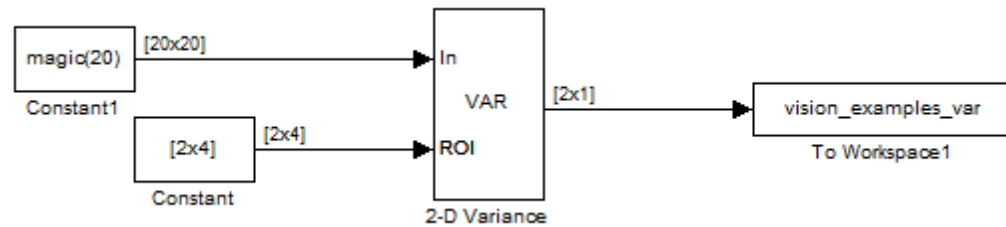
Example

The ex_vision_2dvar calculates the variance value within two ROIs.

See Also

2-D Mean	Computer Vision System Toolbox
2-D Standard Deviation	Computer Vision System Toolbox
var	MATLAB

2-D Variance (To Be Removed)

Purpose   Compute variance of each input matrix

Library   Statistics

Description

Note This 2-D Variance block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated 2-D Variance block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the 2-D Variance documentation to view the reference material related to the obsolete block.

Apply Geometric Transformation

Purpose   Apply projective or affine transformation to an image

Library   Geometric Transformations

visiongeotforms

Description   Use the Apply Geometric Transformation block to apply projective or affine transform to an image. You can use this block to transform the entire image or portions of the image with either polygon or rectangle Regions of Interest (ROIs).

Port Descriptions

Port	Description
Image	*M*-by-*N* or *M*-by-*N*-by-*P* input matrix. *M*: Number of rows in the image.
	N: Number of columns in the image.
	P: Number of color planes in the image.
TForm	When you set the **Transformation matrix source** parameter to Input port, the **TForm** input port accepts:
	• 3-by-2 matrix (affine transform) or a *Q*-by-6 matrix (multiple affine transforms).
	• 3-by-3 matrix (projective transform) or a *Q*-by-9 matrix (multiple projective transforms).
	Q: Number of transformations.
ROI	When you set the ROI source parameter to Input port, the ROI input port accepts: • 4-*element* vector rectangle ROI.
	• 2*L-element* vector polygon ROI.
	• *R*-by-4 matrix for multiple rectangle ROIs.
	• *R*-by-2*L* matrix for multiple polygon ROIs.

Apply Geometric Transformation

Port	Description
	R: Number of Region of Interests (ROIs). $L(L \geq 3)$: Number of vertices in a polygon ROI.

Transformations

The size of the transformation matrix will dictate the transformation type.

Affine Transformation

For affine transformation, the value of the pixel located at $[\hat{x}, \hat{y}]$ in the output image, is determined by the value of the pixel located at $[x, y]$ in the input image. The relationship between the input and the output point locations is defined by the following equations:

$$\begin{cases} \hat{x} = xh_1 + yh_2 + h_3 \\ \hat{y} = xh_4 + yh_5 + h_6 \end{cases}$$

where h_1, h_2, ... h_6 are transformation coefficients.

If you use one transformation, the transformation coefficients must be arranged as a 3-by-2 matrix as in:

$$H = \begin{bmatrix} h_1 & h_4 \\ h_2 & h_5 \\ h_3 & h_6 \end{bmatrix}$$

or in a 1-by-6 vector as in $H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 \end{bmatrix}$.

If you use more than one transformation, the transformation coefficients must be arranged as a Q-by-6 matrix, where each row has the format of $H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 \end{bmatrix}$, and Q is the number of transformations as in:

$$H = \begin{bmatrix} h_{11} & h_{12} & ... & h_{16} \\ h_{21} & h_{22} & ... & h_{26} \\ \vdots & \vdots & ... & \vdots \\ h_{Q1} & h_{Q2} & ... & h_{Q6} \end{bmatrix}$$

Projective Transformation

For projective transformation, the relationship between the input and the output points is defined by the following equations:

$$\begin{cases} \hat{x} = \dfrac{xh_1 + yh_2 + h_3}{xh_7 + yh_8 + h_9} \\ \hat{y} = \dfrac{xh_4 + yh_5 + h_6}{xh_7 + yh_8 + h_9} \end{cases}$$

where h_1, h_2, ... h_9 are transformation coefficients.

If you use one transformation, the transformation coefficients must be arranged as a 3-by-3 matrix as in:

$$H = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$$

or in a 1-by-9 vector as in, $H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}$.

If you use more than one transformation, the transformation coefficients must be arranged as a Q-by-9 matrix, where each row has the format

of $H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}$, and Q is the number of transformations. For example,

$$H = \begin{bmatrix} h_{11} & h_{12} & ... & h_{19} \\ h_{21} & h_{22} & ... & h_{29} \\ \vdots & \vdots & ... & \vdots \\ h_{Q1} & h_{Q2} & ... & h_{Q9} \end{bmatrix}$$

Apply Geometric Transformation

Dialog Box

The **Main** pane of the Apply Geometric Transformation dialog box appears as shown in the following figure.
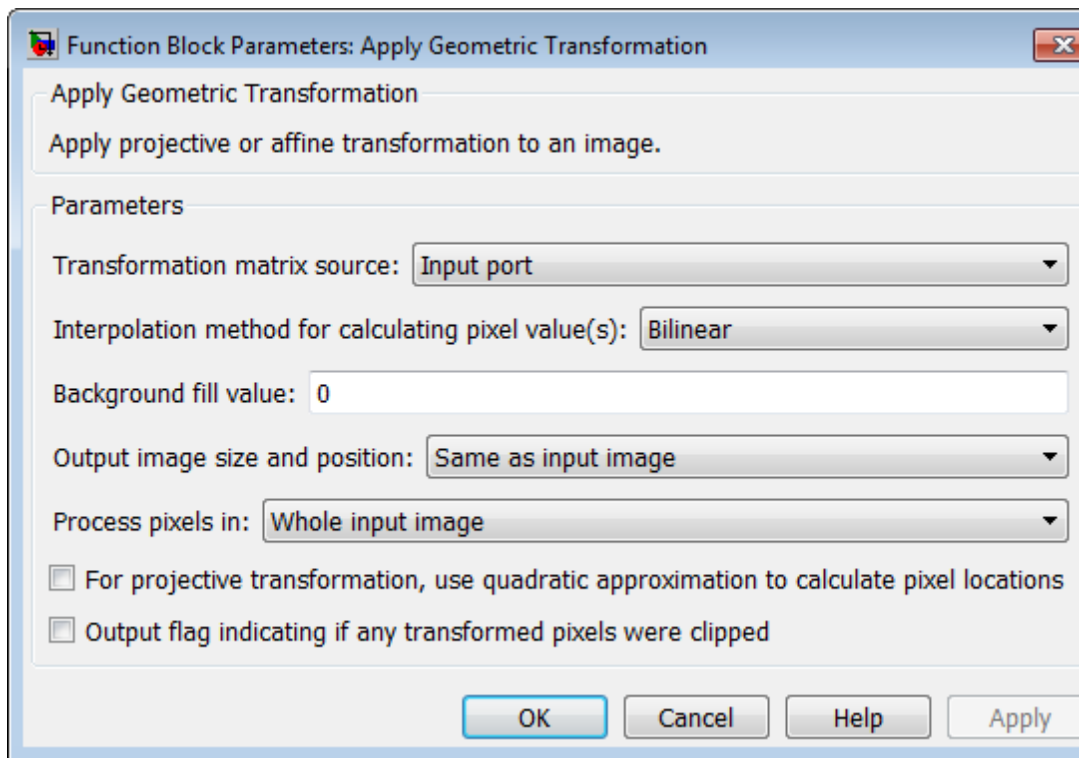
Transformation matrix source

Specify input matrix source, either `Specified via dialog`, or `Input port`. If you select `Specify via dialog`, you can enter the transformation matrix parameters in the parameter that appear with this selection.

Transformation matrix

Specify a 3-by-2, 3-by-3, *Q*-by-6,or a*Q*-by-9 matrix. This option appears when you set the **Transformation matrix source** parameter to `Specified via dialog`.

Interpolation method for calculating pixel value(s)

Specify interpolation method, either `Nearest neighbor`, `Bilinear`, or `Bicubic` interpolation to calculate output pixel values. See Geometric Transformation Interpolation Methods for an overview of these methods.

Background fill value

Specify the value of the pixels that are outside of the input image. Use either a scalar value of P-element vector.

Output image size and position

Specify the output image size to be either `Same as input image`, or `Specify via dialog`. If you select to `Specify via dialog`, you can specify the bounding box in the size and location parameters that appear with this selection.

Size [height width]

Specify the height and width for the output image size as `[height width]`. You can specify this parameter, along with the **Location of the upper left corner [x y]** parameter when you set the **Output image size and position** parameter to `Specify via dialog`.

Location of the upper left corner [x y]

Specify the [x y] location for the upper left corner of the output image. You can specify this parameter, along with the **Size [height width]** parameter, when you set the **Output image size and position** parameter to `Specify via dialog`.

Process pixels in

Specify the region in which to process pixels. Specify `Whole input image`, `Rectangle ROI`, or `Polygon ROI`. If you select `Rectangle ROI`, or `Polygon ROI` the **ROI source** parameter becomes available.

The transformations will be applied on the whole image, or on specified multiple ROIs. The table below outlines how transformation matrices are handled with an entire image and with single and multiple ROIs.

Apply Geometric Transformation

Number of Transformation Matrices	Region of Interest
One transformation matrix	You can apply the transformation on the entire image, single ROI or multiple ROIs.
Multiple transformation matrices	• You can apply multiple transformation matrices on one ROI or on the entire image. The transformations are done in the order they are entered in the **TForm**. • You can apply multiple transformation matrices on multiple ROIs. Each transformation matrix is applied to one ROI. The first transformation matrix specified is applied to the first ROI specified. The second transformation matrix is applied to the second ROI specified, and so on. The number of transformation matrices must be equal to the number of ROIs.

ROI source

Specify the source for the region of interest (ROI), either `Specify via dialog` or `Input port`. This appears when you set the **Process pixels in** parameter to either `Rectangle ROI`, or `Polygon ROI`.

Location and size of rectangle ROI [x y width height]

Specify a 4-*element* vector or an R-by-4 matrix , (where R represents the number of ROIs). This parameter appears when you set the **Process pixels in** parameter to `Rectangle ROI`.

Specify the rectangle by its top-left corner and size in width and height. If you specify one ROI, it must be a 4-*element* vector of format [x y width height]. If you specify more than one ROI, it must be an R-by-4 matrix, such that each row's format is [x y width height].

Vertices of polygon ROI [x1 y1 x2 y2 ... xL yL]

Specify a 2*L*-*element* vector or an R-by-2L matrix, (where R represents the number of ROIs and L is the number of vertices in a polygon). This parameter appears when you set the **Process pixels in** parameter to `Polygon ROI`.

Specify the polygon by its vertices in clockwise or counter-clockwise order, with at least three or more vertices. If you specify one ROI of *L* vertices, it must be a 2*L-element* vector of format

$\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & ... & x_L & y_L \end{bmatrix}$. If you specify more than one ROI, it must be an *R*-by-2*L* matrix, where *L* is the maximum number of vertices in the ROIs. For ROI with vertices fewer than *L*, its last vertex can be repeated to form a vector.

Output flag indicating if any part of ROI is outside input image

Select the **Output flag indicating if any part of ROI is outside input image** check box to enable this output port on the Apply Geometric Transformation block.

For projective transformation, use quadratic approximation to calculate pixel locations

Specify whether to use an exact computation or an approximation for the projective transformation. If you select this option, you can enter an error tolerance in the **Error tolerance (in pixels)** parameter.

Error tolerance (in pixels)

Specify the maximum error tolerance in pixels. This appears when you select the **For projective transformation, use quadratic approximation to calculate pixel locations** check box.

Output flag indicating if any transformed pixels were clipped

Enable output port for flag, which indicates clipping. Clipping occurs when any of the transformed pixels fall outside of the output image.

Examples   **Apply a Projective Transformation to an Image**

The Simple projective transformation model ex_visionApplyGeo_proj, uses the Apply Geometric Transformation block, two Constant blocks and twoVideo Viewer blocks to illustrate a basic model. The transformation matrix determines a projective transformation and is applied to the entire input image. The input image is a checker board. The steps taken to run this model were:

Apply Geometric Transformation

1 Add two Constant blocks for the input image and the transformation matrix. Set the **Constant value** parameters for the constant blocks as follows:

- for the input image, `"checker_board"`, and

- for the transformation matrix, `[1 0 0; .4 1 0; 0 0 1]`

2 Add two Video Viewer blocks, connecting one directly to the input image output port, and the other one to the Apply Geometric Transformation output port.

Create an Image of a Cube using Three Regions of Interest

This example shows how to apply affine transformation on multiple ROIs of an image. It also sets the background color of the output image to a solid color purple. The input image, transformation matrix, and ROI vertices are provided to the Apply Geometric Transformation block via constant blocks. Video viewers are used to view the original image and the output image created. Open this model by typing `ex_visionApplyGeo_roi` at the MATLAB command prompt. The steps taken to run this model was:

1 Change the **Process pixels in** parameter to `Polygon ROI`.

2 Change the **Background fill value** to `[0.5 0.5 0.75]`

3 Add three Constant blocks for the input image, transformation matrix, and ROI vertices. Set the **Constant value** parameters for the three blocks as follows:

- For the input image, `checker_board(20,10)`.

- For the transformation matrix, `[1 0 30 0 1 -30; 1.0204 -0.4082 70 0 0.4082 30; 0.4082 0 89.1836 -0.4082 1 10.8164]`.

- For the polygon ROI, `[1 101 99 101 99 199 1 199; 1 1 99 1 99 99 1 99; 101 101 199 101 199 199 101 199]`.

4 Add two Video Viewer blocks, connecting one directly to the Constant block containing the input image. The other, to the Apply Geometric Transformation output port.

References   [1] George Wolberg, "Digital Image Warping", IEEE Computer Society Press, 3rd edition, 1994.

Richard Hartley and Andrew Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, 2nd edition, 2003.

**Supported
Data
Types**

Port	Supported Data Types
Image	• Double-precision floating point • Single-precision floating point
TForm	• Double-precision floating point • Single-precision floating point
ROI	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	Same as input
Err_roi	Boolean
Err_clip	Boolean

Apply Geometric Transformation

See Also

imtransform	Image Processing Toolbox
Estimate Geometric Transformation	Computer Vision System Toolbox
Trace Boundary	Computer Vision System Toolbox
Blob Analysis	Computer Vision System Toolbox
Video and Image Processing Demos	Computer Vision System Toolbox

Apply Geometric Transformation (To Be Removed)

Purpose   Apply projective or affine transformation to an image

Library   Geometric Transformations

Description

> **Note** This Apply Geometric Transformation block will be removed
> in a future release. It uses the zero-based, [row column] coordinate
> system. It is recommended that you replace this block with the updated
> Apply Geometric Transformation block that uses the one-based, [x y]
> coordinate system.
>
> Refer to "Conventions Changed for Indexing, Spatial Coordinates, and
> Representation of Geometric Transforms" in the R2011b Release Notes
> for details related to these changes.
>
> See the Apply Geometric Transformation documentation to view the
> reference material related to the obsolete block.

Autothreshold

Purpose   Convert intensity image to binary image

Library   Conversions

visionconversions

Description   The Autothreshold block converts an intensity image to a binary image using a threshold value computed using Otsu's method.
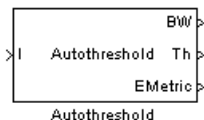
This block computes this threshold value by splitting the histogram of the input image such that the variance of each pixel group is minimized.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
BW	Scalar, vector, or matrix that represents a binary image	Boolean	No
Th	Threshold value	Same as I port	No
EMetric	Effectiveness metric	Same as I port	No

Use the **Thresholding operator** parameter to specify the condition the block places on the input values. If you select > and the input value is greater than the threshold value, the block outputs 1 at the BW port; otherwise, it outputs 0. If you select <= and the input value is less

than or equal to the threshold value, the block outputs 1; otherwise, it outputs 0.

Select the **Output threshold** check box to output the calculated threshold values at the Th port.

Select the **Output effectiveness metric** check box to output values that represent the effectiveness of the thresholding at the EMetric port. This metric ranges from 0 to 1. If every pixel has the same value, the effectiveness metric is 0. If the image has two pixel values or the histogram of the image pixels is symmetric, the effectiveness metric is 1.

If you clear the **Specify data range** check box, the block assumes that floating-point input values range from 0 to 1. To specify a different data range, select this check box. The **Minimum value of input** and **Maximum value of input** parameters appear in the dialog box. Use these parameters to enter the minimum and maximum values of your input signal.

Use the **When data range is exceeded** parameter to specify the block's behavior when the input values are outside the expected range. The following options are available:

- Ignore — Proceed with the computation and do not issue a warning message. If you choose this option, the block performs the most efficient computation. However, if the input values exceed the expected range, the block produces incorrect results.

- Saturate — Change any input values outside the range to the minimum or maximum value of the range and proceed with the computation.

- Warn and saturate — Display a warning message in the MATLAB Command Window, saturate values, and proceed with the computation.

- Error — Display an error dialog box and terminate the simulation.

If you clear the **Scale threshold** check box, the block uses the threshold value computed by Otsu's method to convert intensity images

into binary images. If you select the **Scale threshold** check box, the **Threshold scaling factor** appears in the dialog box. Enter a scalar value. The block multiplies this scalar value with the threshold value computed by Otsu's method and uses the result as the new threshold value.

Fixed-Point Data Types

The following diagram shows the data types used in the Autothreshold block for fixed-point signals. You can use the default fixed-point parameters if your input has a word length less than or equal to 16.

In this diagram, DT means data type. You can set the product, accumulator, quotient, and effectiveness metric data types in the block mask.
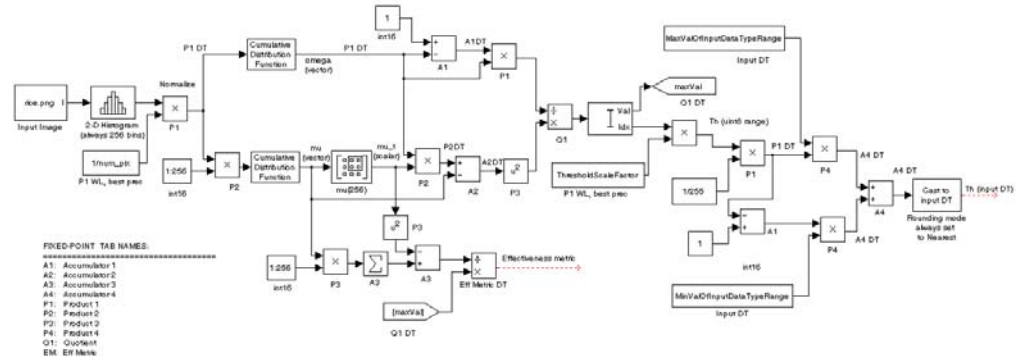
Dialog Box

The **Main** pane of the Autothreshold dialog box appears as shown in the following figure.
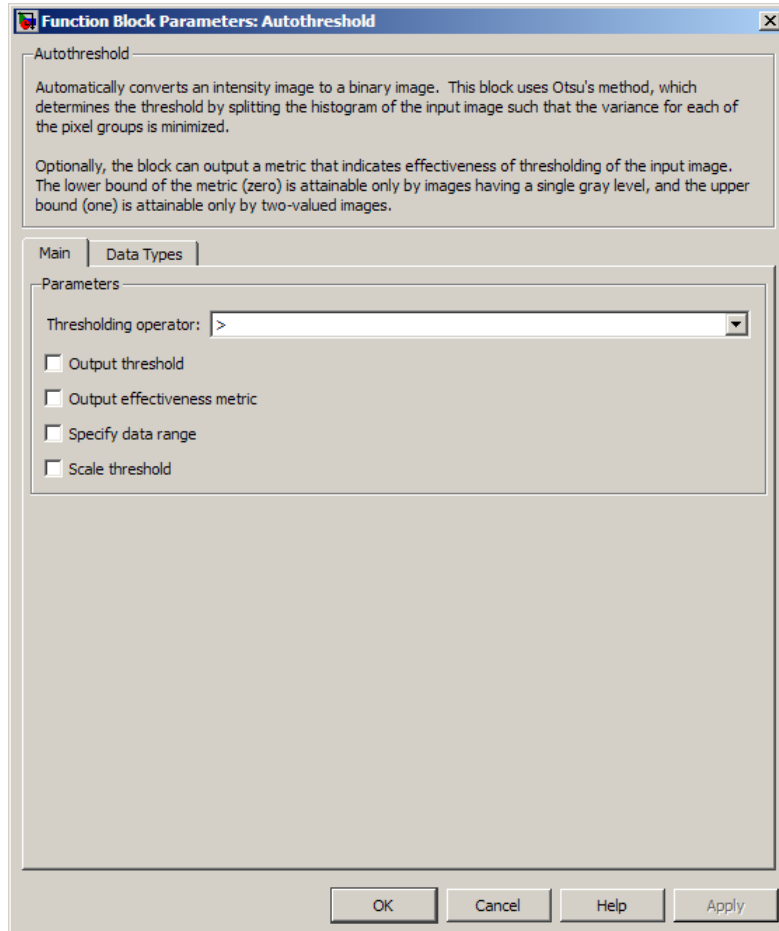
Thresholding operator

Specify the condition the block places on the input matrix values. If you select > or <=, the block outputs 0 or 1 depending on

whether the input matrix values are above, below, or equal to the threshold value.

Output threshold

Select this check box to output the calculated threshold values at the Th port.

Output effectiveness metric

Select this check box to output values that represent the effectiveness of the thresholding at the EMetric port.

Specify data range

If you clear this check box, the block assumes that floating-point input values range from 0 to 1. To specify a different data range, select this check box.

Minimum value of input

Enter the minimum value of your input data. This parameter is visible if you select the **Specify data range** check box.

Maximum value of input

Enter the maximum value of your input data. This parameter is visible if you select the **Specify data range** check box.

When data range is exceeded

Specify the block's behavior when the input values are outside the expected range. Your options are `Ignore`, `Saturate`, `Warn and saturate`, or `Error`. This parameter is visible if you select the **Specify data range** check box.

Scale threshold

Select this check box to scale the threshold value computed by Otsu's method.

Threshold scaling factor

Enter a scalar value. The block multiplies this scalar value with the threshold value computed by Otsu's method and uses the result as the new threshold value. This parameter is visible if you select the **Scale threshold** check box.

The **Data Types pane** pane of the Autothreshold dialog box appears as
follows. You can use the default fixed-point parameters if your input
has a word length less than or equal to 16.
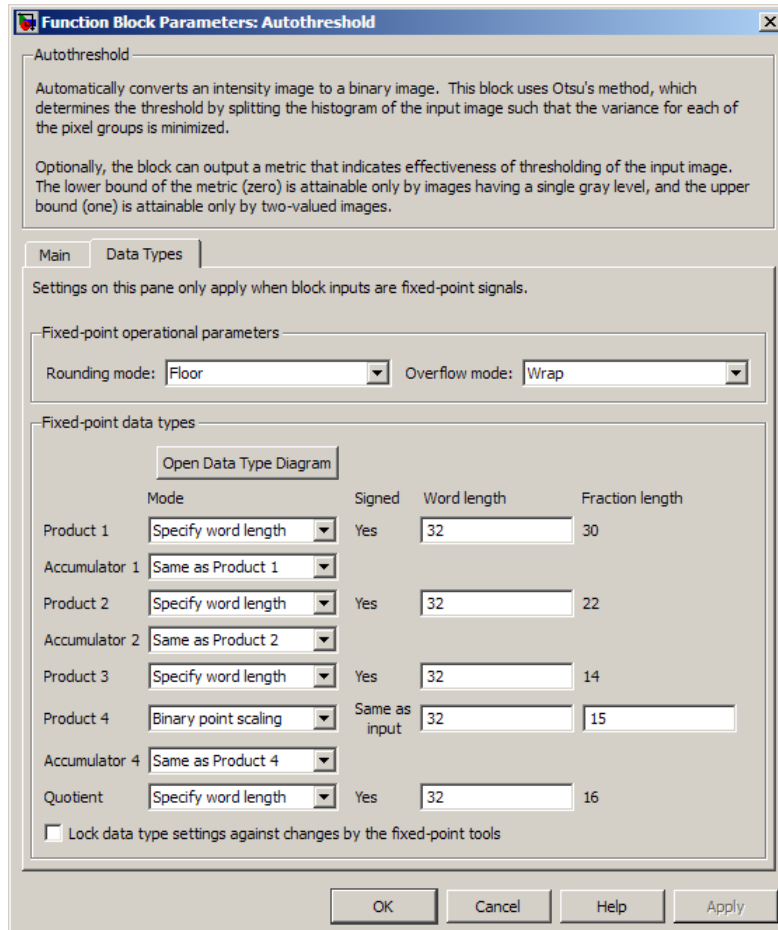
Rounding mode

> Select the rounding mode for fixed-point operations. This
> parameter does not apply to the Cast to input DT step shown in

"Fixed-Point Data Types" on page 1-204. For this step, **Rounding mode** is always set to Nearest.

Overflow mode

Select the overflow mode for fixed-point operations.

Product 1, 2, 3, 4

As shown previously, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate the product output word and fraction lengths.

- When you select Specify word length, you can enter the word length of the product values in bits. The block sets the fraction length to give you the best precision.

- When you select Same as input, the characteristics match those of the input to the block. This choice is only available for the **Product 4** parameter.

- When you select Binary point scaling, you can enter the word length and the fraction length of the product output in bits.

- When you select Slope and bias scaling, you can enter the word length in bits and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Accumulator 1, 2, 3, 4

The result of each addition remains
in the accumulator data type.

Input to adder -
product output
data type

CAST

Accumulator
data type

ADDER

Accumulator
data type
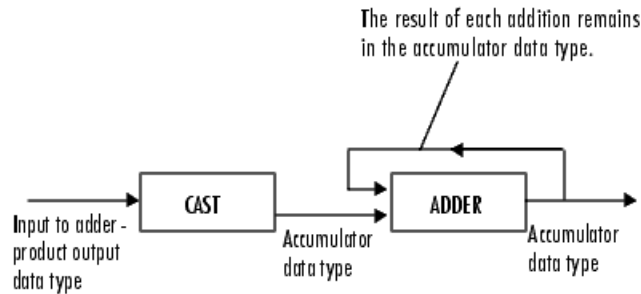
As shown previously, inputs to the accumulator are cast to the
accumulator data type. The output of the adder remains in the
accumulator data type as each element of the input is added to it.
Use this parameter to specify how to designate the accumulator
word and fraction lengths.

- When you select `Same as Product`, these characteristics match
  those of the product output.

- When you select `Specify word length`, you can enter the
  word length of the accumulator values in bits. The block sets
  the fraction length to give you the best precision. This choice is
  not available for the **Accumulator 4** parameter because it is
  dependent on the input data type.

- When you select `Binary point scaling`, you can enter the
  word length and the fraction length of the accumulator in bits.

- When you select `Slope and bias scaling`, you can enter the
  word length in bits and the slope of the accumulator. The bias
  of all signals in the Computer Vision System Toolbox software
  is 0.

The **Accumulator 3** parameter is only visible if, on the **Main**
pane, you select the **Output effectiveness metric** check box.

Quotient

Choose how to specify the word length and fraction length of the quotient data type:

- When you select `Specify word length`, you can enter the word length of the quotient values in bits. The block sets the fraction length to give you the best precision.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the quotient, in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the quotient. The bias of all signals in the Computer Vision System Toolbox software is 0.

Eff Metric

Choose how to specify the word length and fraction length of the effectiveness metric data type. This parameter is only visible if, on the **Main** tab, you select the **Output effectiveness metric** check box.

- When you select `Specify word length`, you can enter the word length of the effectiveness metric values, in bits. The block sets the fraction length to give you the best precision.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the effectiveness metric in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the effectiveness metric. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Example

Thresholding Intensity Images Using the Autothreshold Block

Convert an intensity image into a binary image. Use the Autothreshold block when lighting conditions vary and the threshold needs to change for each video frame.

You can open the example model by typing

```
ex_vision_autothreshold
```

on the MATLAB command line.

See Also

Compare To Constant	Simulink
Relational Operator	Simulink
graythresh	Image Processing Toolbox

Blob Analysis

Purpose   Compute statistics for labeled regions

Library   Statistics

                     visionstatistics

Description   Use the Blob Analysis block to calculate statistics for labeled regions
in a binary image. The block returns quantities such as the centroid,
bounding box, label matrix, and blob count. The Blob Analysis block
supports input and output variable size signals.
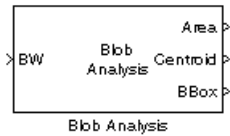
For information on pixel and spatial coordinate system definitions, see
"Expressing Image Locations" in the Image Processing Toolbox User
Guide. Use the Selector block from the Simulink, to select certain blobs
based on their statistics.

Port Descriptions

Port	Input/Output	Supported Data Types
BW	Vector or matrix that represents a binary image	Boolean
Area	Vector that represents the number of pixels in labeled regions	32-bit signed integer
Centroid	*M*-by-2 matrix of centroid coordinates, where *M* represents the number of blobs	• Double-precision floating point • Single-precision floating point • Fixed point
BBox	*M*-by-4 matrix of [x y width height] bounding box coordinates, where *M* represents the number of blobs and [x y] represents the upper left corner of the bounding box.	32-bit signed integer

Port	Input/Output	Supported Data Types
MajorAxis	Vector that represents the lengths of major axes of ellipses	• Double-precision floating point • Single-precision floating point
MinorAxis	Vector that represents the lengths of minor axes of ellipses	Same as MajorAxis port
Orientation	Vector that represents the angles between the major axes of the ellipses and the *x*-axis.	Same as MajorAxis port
Eccentricity	Vector that represents the eccentricities of the ellipses	Same as MajorAxis port
Diameter ^2	Vector that represents the equivalent diameters squared	Same as Centroid port
Extent	Vector that represents the results of dividing the areas of the blobs by the area of their bounding boxes	Same as Centroid port
Perimeter	Vector containing an estimate of the perimeter length, in pixels, for each blob	Same as Centroid port
Label	Label matrix	8-, 16-, or 32-bit unsigned integer
Count	Scalar value that represents the actual number of labeled regions in each image	Same as Label port

Dialog Box

The **Main** pane of the Blob Analysis dialog box appears as shown in the following figure. Use the check boxes to specify the statistics values you want the block to output. For a full description of each of these statistics, see the `regionprops` function reference page in the Image Processing Toolbox documentation.

Blob Analysis
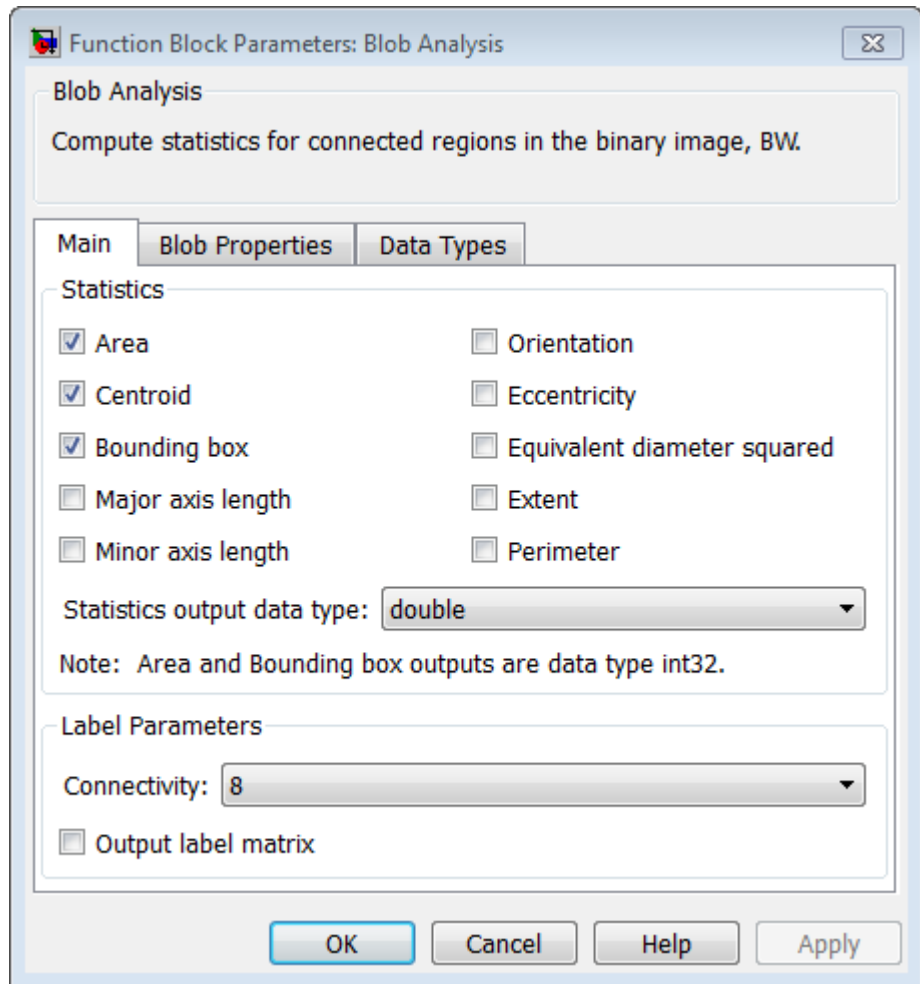
Area

Select this check box to output a vector that represents the number of pixels in labeled regions

Centroid

Select this check box to output an *M*-by-2 matrix of [x y] centroid coordinates. The rows represent the coordinates of the centroid of each region, where *M* represents the number of blobs.

Example: Suppose there are two blobs, where the row and column coordinates of their centroids are x_1, y_1 and x_2, y_2, respectively. The block outputs:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

at the Centroid port.

Bounding box

Select this check box to output an *M*-by-4 matrix of [x y width height] bounding boxes. The rows represent the coordinates of each bounding box, where *M* represents the number of blobs.

Example: Suppose there are two blobs, where *x* and *y* define the location of the upper-left corner of the bounding box, and *w*, *h* define the width and height of the bounding box. The block outputs

$$\begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \end{bmatrix}$$

at the BBox port.

Major axis length

Select this check box to output a vector with the following characteristics:

- Represents the lengths of the major axes of ellipses

- Has the same normalized second central moments as the labeled regions

Blob Analysis

Minor axis length

Select this check box to output a vector with the following characteristics:

- Represents the lengths of the minor axes of ellipses

- Has the same normalized second central moments as the labeled regions

Orientation

Select this check box to output a vector that represents the angles between the major axes of the ellipses and the *x*-axis. The angle values are in radians and range between:

$-\dfrac{\pi}{2}$ and $\dfrac{\pi}{2}$

Eccentricity

Select this check box to output a vector that represents the eccentricities of ellipses that have the same second moments as the region

Equivalent diameter squared

Select this check box to output a vector that represents the equivalent diameters squared

Extent

Select this check box to output a vector that represents the results of dividing the areas of the blobs by the area of their bounding boxes

Perimeter

Select this check box to output an *N-by-1* vector of the perimeter lengths, in pixels, of each blob, where *N* is the number of blobs.

Statistics output data type

Specify the data type of the outputs at the **Centroid**, **MajorAxis**, **MinorAxis**, **Orientation**, **Eccentricity**, **Equivalent diameter squared**, and **Extent** ports. If you select Fixed-point, the block cannot calculate the major axis, minor axis, orientation, or eccentricity and the associated check boxes become unavailable.

Connectivity

Define which pixels connect to each other. If you want to connect pixels located on the top, bottom, left, and right, select 4. If you want to connect pixels to the other pixels on the top, bottom, left, right, and diagonally, select 8. For more information about this parameter, see the Label block reference page.

The **Connectivity** parameter also affects how the block calculates the perimeter of a blob. For example:

The following figure illustrates how the block calculates the perimeter when you set the **Connectivity** parameter to 4.
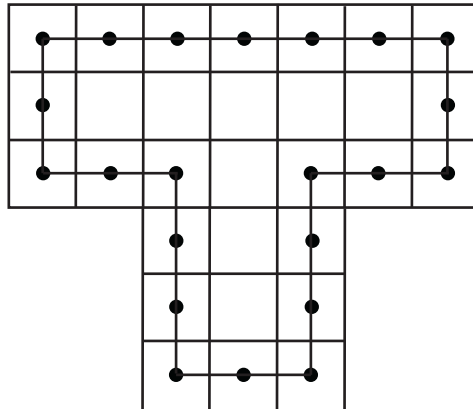
The block calculates the distance between the center of each pixel (marked by the black dots) and estimates the perimeter to be 22.

The next figure illustrates how the block calculates the perimeter of a blob when you set the **Connectivity** parameter to 8.

Blob Analysis
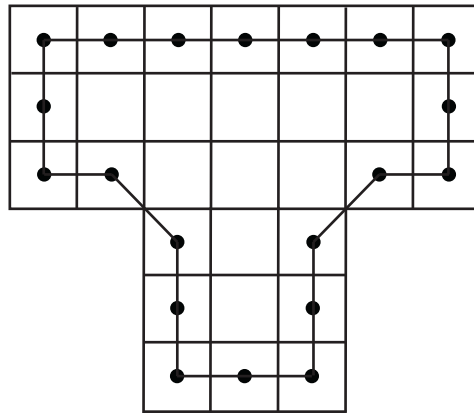
The block takes a different path around the blob and estimates

the perimeter to be $18 + 2\sqrt{2}$.

Output label matrix

Select this check box, to output the label matrix at the **Label** port. The pixels equal to 0 represent the background. The pixels equal to 1 represent the first object. The pixels equal to 2 represent the second object, and so on.

The **Blob Properties** pane of the Blob Analysis dialog box appears as shown in the following figure.
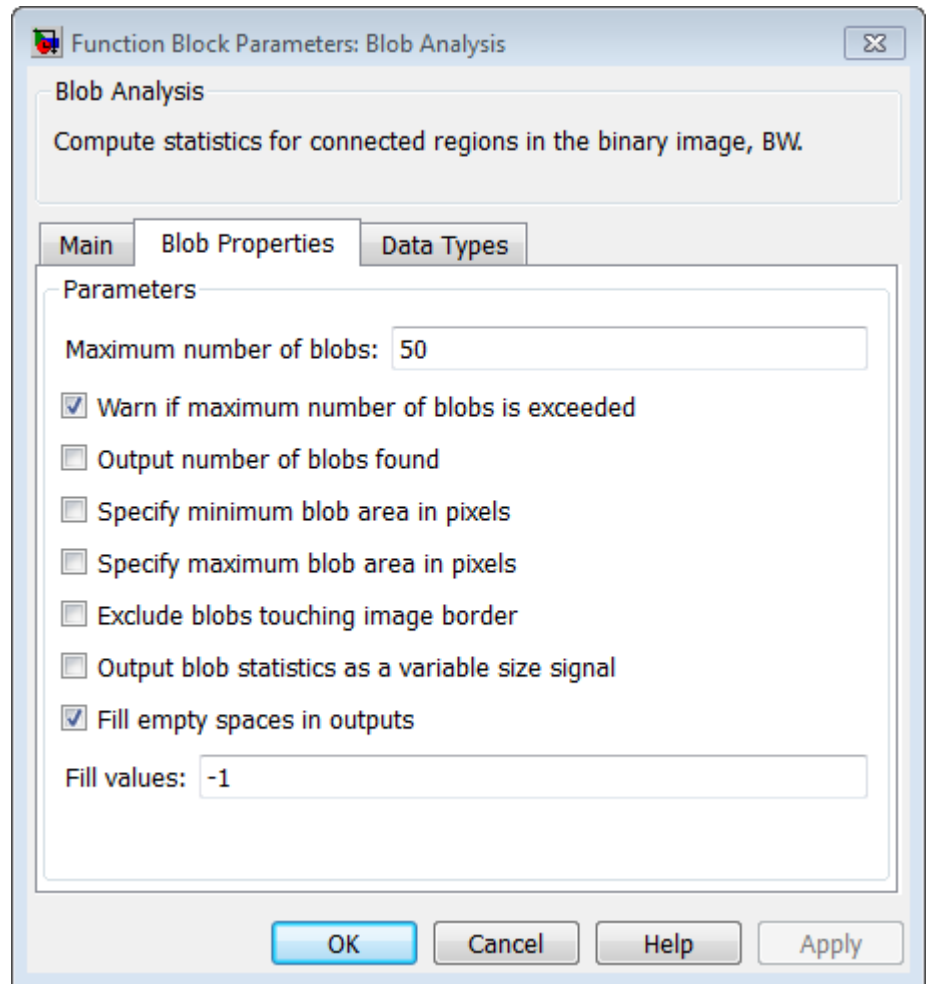
Maximum number of blobs

Specify the maximum number of labeled regions in each input image. The block uses this value to preallocate vectors and matrices to ensure that they are long enough to hold the statistical values. The maximum number of blobs the block outputs depends

Blob Analysis

on both the value of this parameter, and on the size of the input image. The number of blobs the block outputs may be limited by the input image size.

Warn if maximum number of blobs is exceeded
Select this check box to output a warning when the number of blobs in an image is greater than the value of **Maximum number of blobs** parameter.

Output number of blobs found
Select this check box to output a scalar value that represents the actual number of connected regions in each image at the **Count** port.

Specify maximum blob area in pixels
Select this check box to enter the minimum blob area in the edit box that appears beside the check box. The blob gets a label if the number of pixels meets the minimum size specified. The maximum allowable value is the maximum of `uint32` data type. This parameter is tunable.

Exclude blobs touching image border
Select this check box to exclude a labeled blob that contains at least one border pixel.

Output blob statistics as a variable-size signal
Select this check box to output blob statistics as a variable-size signal. Selecting this check box means that you do not need to specify fill values.

Fill empty spaces in outputs outputs
Select this check box to fill empty spaces in the statistical vectors with the values you specify in the **Fill values** parameter.

The **Fill empty spaces in outputs** check box does not appear when you select the **Output blob statistics as a variable-size signal** check box.

Fill values

If you enter a scalar value, the block fills all the empty spaces in the statistical vectors with this value. If you enter a vector, it must have the same length as the number of selected statistics. The block uses each vector element to fill a different statistics vector. If the empty spaces do not affect your computation, you can deselect the **Fill empty spaces in outputs** check box. As a best practice, leave this check box selected.

The **Fill values** parameter is not visible when you select the **Output blob statistics as a variable-size signal** check box.

The **Fixed-Point Data Types** pane of the Blob Analysis dialog box appears as shown in the following figure.

The result of each addition remains in the accumulator data type.

MULTIPLIER → CAST → ADDER → CAST

Input data type(s)   Product output data type   Accumulator data type   Accumulator data type   Output data type
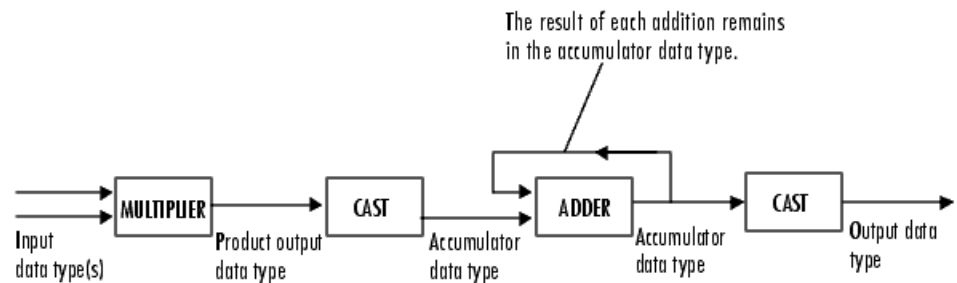
The parameters on the **Fixed-point** tab apply only when you set the **Statistics output data type** parameter to Specify via Fixed-point tab.

Blob Analysis
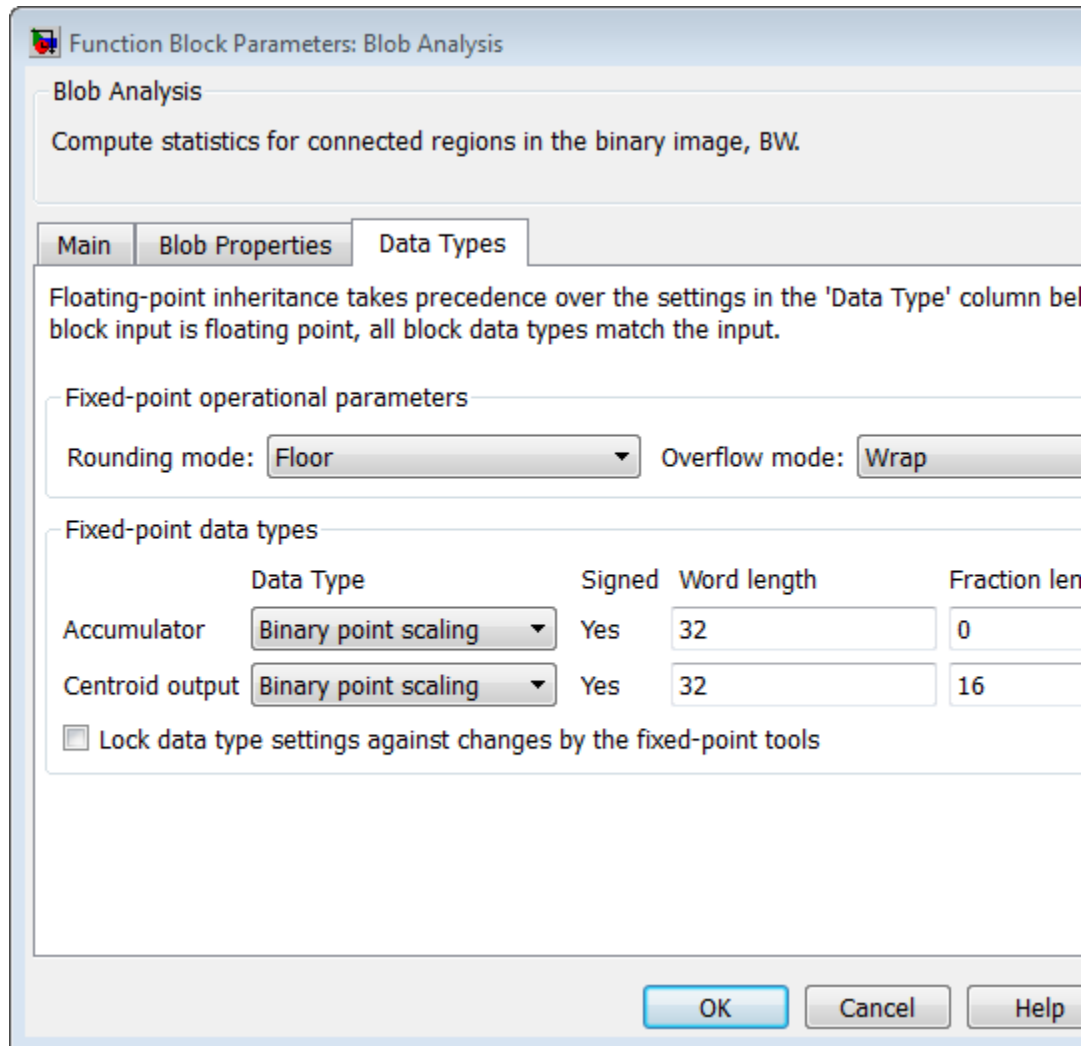
Rounding mode

Select the rounding mode `Floor`, `Ceiling`, `Nearest` or `Zero` for fixed-point operations.

Overflow mode

Select the overflow mode, Wrap or Saturate for fixed-point operations.

Product output

When you select Binary point scaling, you can enter the **Word length** and the **Fraction length** of the product output, in bits.

When you select Slope and bias scaling, you can enter the **Word length** in bits, and the **Slope** of the product output. All signals in the Computer Vision System Toolbox software have a bias of 0.

The block places the output of the multiplier into the **Product output** data type and scaling. The computation of the equivalent diameter squared uses the product output data type. During this computation, the block multiplies the blob area (stored in the accumulator) by the 4/pi factor. This factor has a word length that equals the value of **Equivalent diameter squared** output data type **Word length**. The value of the **Fraction length** equals its word length minus two. Use this parameter to specify how to designate this product output word and fraction lengths.

Accumulator

When you select Same as product output the characteristics match the characteristics of the product output.

When you select Binary point scaling, you can enter the **Word length** and the **Fraction length** of the accumulator, in bits.

When you select Slope and bias scaling, you can enter the **Word length**, in bits, and the **Slope** of the **Accumulator**. All

signals in the Computer Vision System Toolbox software have a bias of 0.

The result of each addition remains in the accumulator data type.
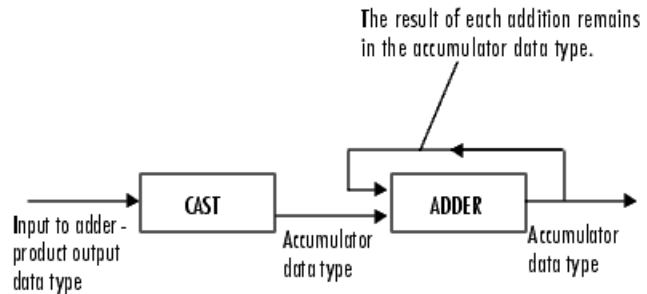
Inputs to the **Accumulator** get cast to the accumulator data type. Each element of the input gets added to the output of the adder, which remains in the accumulator data type. Use this parameter to specify how to designate this accumulator word and fraction lengths.

Centroid output

Choose how to specify the **Word length** and **Fraction length** of the output at the **Centroid** port:

- When you select Same as accumulator, these characteristics match the characteristics of the accumulator.

- When you select Binary point scaling, you can enter the **Word length** and **Fraction length** of the output, in bits.

- When you select Slope and bias scaling, you can enter the **Word length**, in bits, and the **Slope** of the output. All signals in the Computer Vision System Toolbox software have a bias of 0.

Equiv Diam^2 output

Choose how to specify the **Word length** and **Fraction length** of the output at the **Diameter ^2** port:

- When you select `Same as accumulator`, these characteristics match the characteristics of the **Accumulator**.

- When you select `Same as product output`, these characteristics match the characteristics of the **Product output**.

- When you select `Binary point scaling`, you can enter the **Word length** and **Fraction length** of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the **Word length**, in bits, and the **Slope** of the output. All signals in the Computer Vision System Toolbox software have a bias of 0.

Extent output

Choose how to specify the **Word length** and **Fraction length** of the output at the **Extent** port:

- When you select `Same as accumulator`, these characteristics match the characteristics of the accumulator.

- When you select `Binary point scaling`, you can enter the **Word length** and **Fraction length** of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the **Word length**, in bits, and the **Slope** of the output. All signals in the Computer Vision System Toolbox software have a bias of 0.

Fill in empty spaces outputs

Select this check box to fill empty spaces in the statistical vectors with the value you specify in the **Fill values** parameter.

The **Fill empty spaces in outputs** check box is not visible when you select the **Output blob statistics as a variable-size signal** check box.

Perimeter output

Choose how to specify the **Word length** and **Fraction length** of the output at the **Perimeter** port:

Blob Analysis

- When you select `Same as accumulator`, these characteristics match the characteristics of the accumulator.

-

    When you select `Binary point scaling`, you can enter the **Word length** and **Fraction length** of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the **Slope** of the output. All signals in the Computer Vision System Toolbox software have a bias of 0.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent the autoscaling tool in the Fixed-Point Tool overriding any fixed-point scaling you specify in this block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also

Label	Computer Vision System Toolbox
regionprops	Image Processing Toolbox

Purpose   Compute statistics for labeled regions

Library   Statistics

Description
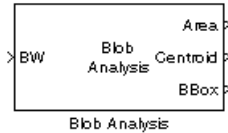
Note This Blob Analysis block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Blob Analysis block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Blob Analysis documentation to view the reference material related to the obsolete block.

Block Matching

Purpose   Estimate motion between images or video frames

Library   Analysis & Enhancement

visionanalysis

Description   The Block Matching block estimates motion between two images or two video frames using "blocks" of pixels. The Block Matching block matches the block of pixels in frame k to a block of pixels in frame k+1 by moving the block of pixels over a search region.

Suppose the input to the block is frame k. The Block Matching block performs the following steps:

1 The block subdivides this frame using the values you enter for the **Block size [height width]** and **Overlap [r c]** parameters. In the following example, the **Overlap [r c]** parameter is [0 0].

2 For each subdivision or block in frame k+1, the Block Matching block establishes a search region based on the value you enter for the **Maximum displacement [r c]** parameter.

3 The block searches for the new block location using either the Exhaustive or Three-step search method.

Input image = frame k

STEP 1: Subdivide the image in frame k.

Center pixel
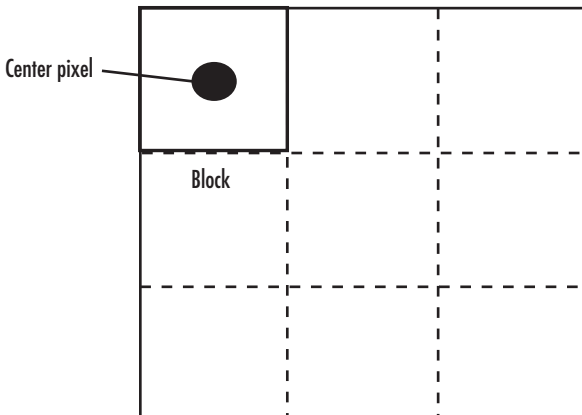
Block

STEP 2: Establish the search region in frame k+1.
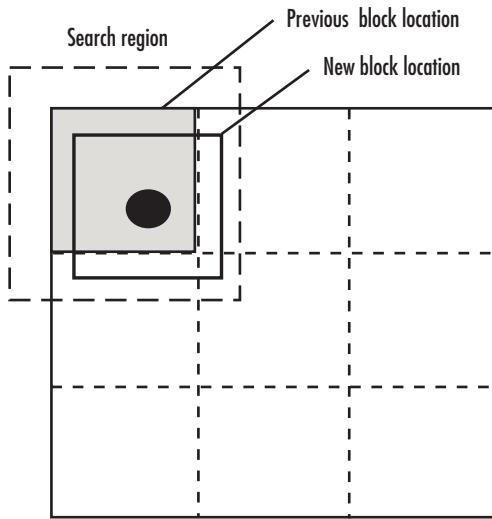
STEP 3: Search for the new block location in frame k+1.

Search region

Previous block location

New block location

Search region
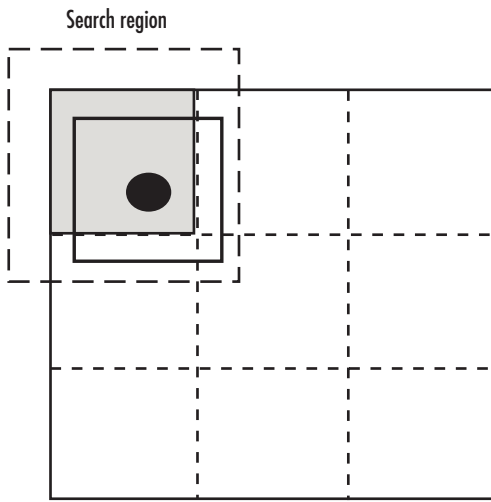
Block Matching

Port	Output	Supported Data Types	Complex Values Supported
I/I1	Scalar, vector, or matrix of intensity values	- Double-precision floating point - Single-precision floating point - Fixed point 8-, 16-, and 32-bit signed integer 8-, 16-, and 32-bit unsigned integer	No
I2	Scalar, vector, or matrix of intensity values	Same as I port	No
|V|^2	Matrix of velocity magnitudes	Same as I port	No
V	Matrix of velocity components in complex form	Same as I port	Yes

Use the **Estimate motion between** parameter to specify whether to estimate the motion between two images or two video frames. If you select Current frame and N-th frame back, the **N** parameter appears in the dialog box. Enter a scalar value that represents the number of frames between the reference frame and the current frame.

Use the **Search method** parameter to specify how the block locates the block of pixels in frame k+1 that best matches the block of pixels in frame k.

- If you select Exhaustive, the block selects the location of the block of pixels in frame k+1 by moving the block over the search region 1 pixel at a time. This process is computationally expensive.

- If you select Three-step, the block searches for the block of pixels in frame k+1 that best matches the block of pixels in frame k using a steadily decreasing step size. The block begins with a step size

approximately equal to half the maximum search range. In each step, the block compares the central point of the search region to eight search points located on the boundaries of the region and moves the central point to the search point whose values is the closest to that of the central point. The block then reduces the step size by half, and begins the process again. This option is less computationally expensive, though it might not find the optimal solution.

Use the **Block matching criteria** parameter to specify how the block measures the similarity of the block of pixels in frame k to the block of pixels in frame k+1. If you select Mean square error (MSE), the Block Matching block estimates the displacement of the center pixel of the block as the (d_1, d_2) values that minimize the following MSE equation:

$$MSE(d_1, d_2) = \frac{1}{N_1 \times N_2} \sum_{(n_1, n_2), \in B} \sum [s(n_1, n_2, k) - s(n_1 + d_1, n_2 + d_2, k+1)]^2$$

In the previous equation, B is an $N_1 \times N_2$ block of pixels, and $s(x, y, k)$ denotes a pixel location at (x, y) in frame k. If you select Mean absolute difference (MAD), the Block Matching block estimates the displacement of the center pixel of the block as the (d_1, d_2) values that minimize the following MAD equation:

$$MAD(d_1, d_2) = \frac{1}{N_1 \times N_2} \sum_{(n_1, n_2), \in B} \sum | s(n_1, n_2, k) - s(n_1 + d_1, n_2 + d_2, k+1) |$$

Use the **Block size [height width]** and **Overlap [r c]** parameters to specify how the block subdivides the input image. For a graphical description of these parameters, see the first figure in this reference page. If the **Overlap [r c]** parameter is not [0 0], the blocks would overlap each other by the number of pixels you specify.

Use the **Maximum displacement [r c]** parameter to specify the maximum number of pixels any center pixel in a block of pixels might

move from image to image or frame to frame. The block uses this value to determine the size of the search region.

Use the **Velocity output** parameter to specify the block's output. If you select `Magnitude-squared`, the block outputs the optical flow matrix where each element is of the form u^2+v^2. If you select `Horizontal and vertical components in complex form`, the block outputs the optical

flow matrix where each element is of the form $u + jv$. The real part of each value is the horizontal velocity component and the imaginary part of each value is the vertical velocity component.

Fixed-Point Data Types

The following diagram shows the data types used in the Block Matching block for fixed-point signals.

MSE Block Matching
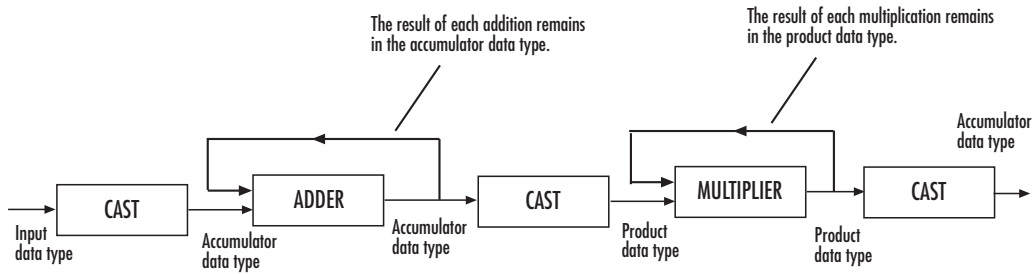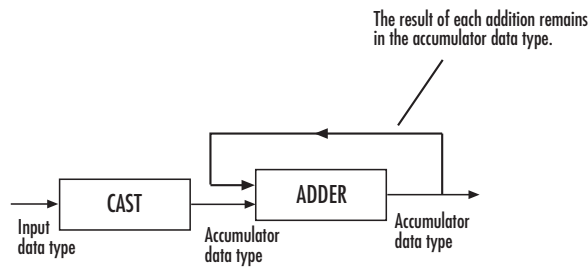
The result of each addition remains
in the accumulator data type.

The result of each multiplication remains
in the product data type.

Accumulator
data type

| CAST | ADDER | CAST | MULTIPLIER | CAST |

Input
data type

Accumulator
data type

Accumulator
data type

Product
data type

Product
data type

MAD Block Matching

The result of each addition remains
in the accumulator data type.

| CAST | ADDER |

Input
data type

Accumulator
data type

Accumulator
data type

You can set the accumulator and output data types in the block mask as discussed in the next section.

Block Matching

Dialog Box

The **Main** pane of the Block Matching dialog box appears as shown in the following figure.
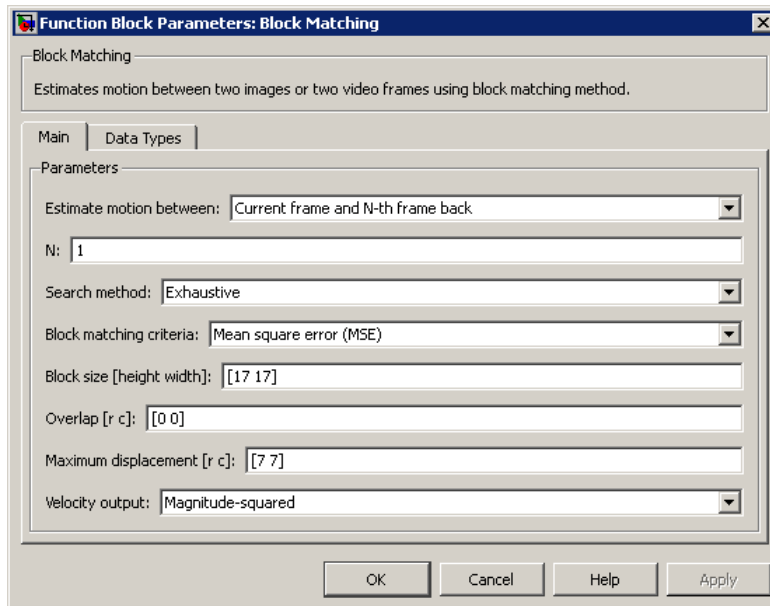
Estimate motion between

Select Two images to estimate the motion between two images. Select Current frame and N-th frame back to estimate the motion between two video frames that are N frames apart.

N

Enter a scalar value that represents the number of frames between the reference frame and the current frame. This parameter is only visible if, for the **Estimate motion between** parameter, you select Current frame and N-th frame back.

Search method

Specify how the block searches for the block of pixels in the next image or frame. Your choices are Exhaustive or Three-step.

Block matching criteria

Specify how the block measures the similarity of the block of pixels in frame k to the block of pixels in frame k+1. Your choices are Mean square error (MSE) or Mean absolute difference (MAD).

Block size [height width]

Specify the size of the block of pixels.

Overlap [r c]

Specify the overlap (in pixels) of two subdivisions of the input image.

Maximum displacement [r c]

Specify the maximum number of pixels any center pixel in a block of pixels might move from image to image or frame to frame. The block uses this value to determine the size of the search region.

Velocity output

If you select Magnitude-squared, the block outputs the optical flow matrix where each element is of the form $u^2 + v^2$. If you select Horizontal and vertical components in complex form, the block outputs the optical flow matrix where each element is of the form $u + jv$.

The **Data Types** pane of the Block Matching dialog box appears as shown in the following figure.

Block Matching
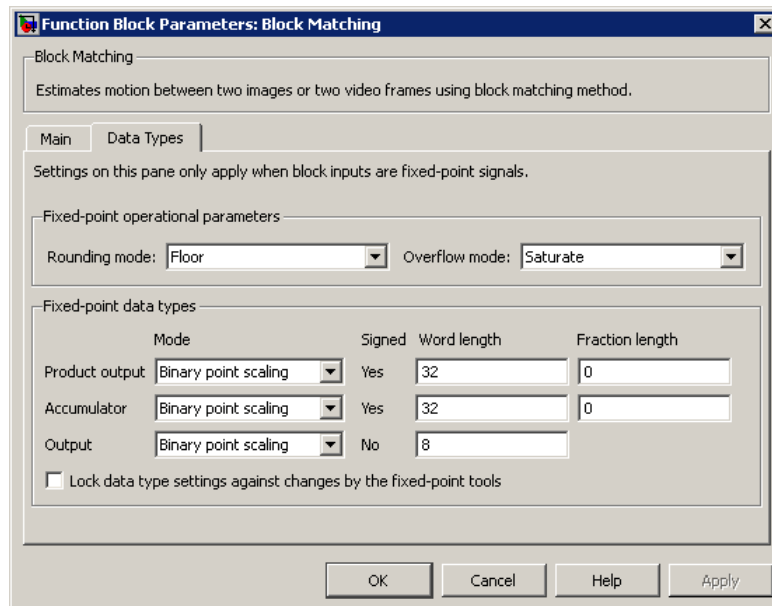
Rounding mode
>   Select the rounding mode for fixed-point operations.

Overflow mode
>   Select the overflow mode for fixed-point operations.

Product output

>   As shown previously, the output of the multiplier is placed into
>   the product output data type and scaling. Use this parameter to
>   specify how to designate the product output word and fraction
>   lengths.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.
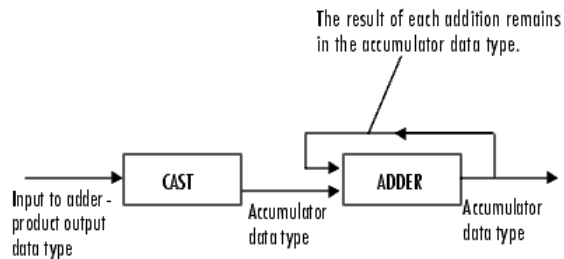
Accumulator

As depicted previously, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software is 0.

Block Matching

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Binary point scaling`, you can enter the word length of the output, in bits. The fractional length is always 0.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, of the output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also   Optical Flow   Computer Vision System Toolbox software

Purpose   Repeat user-specified operation on submatrices of input matrix

Library   Utilities

               visionutilities
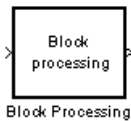
Description   The Block Processing block extracts submatrices of a user-specified
               size from each input matrix. It sends each submatrix to a subsystem
               for processing, and then reassembles each subsystem output into the
               output matrix.
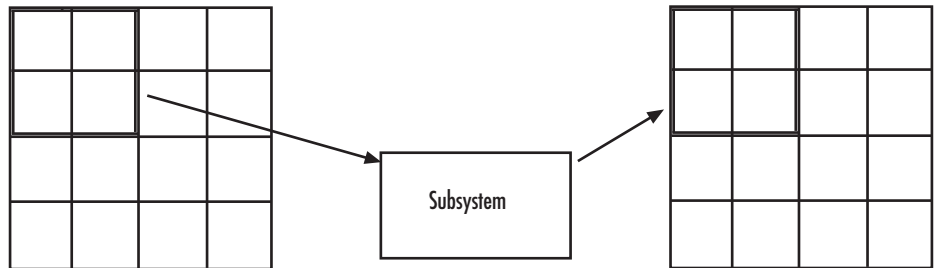
Note Because you modify the Block Processing block's subsystem,
the link between this block and the block library is broken when you
click-and-drag a Block Processing block into your model. As a result,
this block will not be automatically updated if you upgrade to a newer
version of the Computer Vision System Toolbox software. If you
right-click on the block and select **Mask>Look Under Mask**, you can
delete blocks from this subsystem without triggering a warning. Lastly,
if you search for library blocks in a model, this block will not be part of
the results.

The blocks inside the subsystem dictate the frame status of the input
and output signals, whether single channel or multichannel signals are
supported, and which data types are supported by this block.

Block Processing

Use the **Number of inputs** and **Number of outputs** parameters to specify the number of input and output ports on the Block Processing block.

Use the **Block size** parameter to specify the size of each submatrix in cell array format. Each vector in the cell array corresponds to one input; the block uses the vectors in the order you enter them. If you have one input port, enter one vector. If you have more than one input port, you can enter one vector that is used for all inputs or you can specify a different vector for each input. For example, if you want each submatrix to be 2-by-3, enter {[2 3]}.

Use the **Overlap** parameter to specify the overlap of each submatrix in cell array format. Each vector in the cell array corresponds to the overlap of one input; the block uses the vectors in the order they are specified. If you enter one vector, each overlap is the same size. For example, if you want each 3-by-3 submatrix to overlap by 1 row and 2 columns, enter {[1 2]}.

The **Traverse order** parameter determines how the block extracts submatrices from the input matrix. If you select Row-wise, the block extracts submatrices by moving across the rows. If you select Column-wise, the block extracts submatrices by moving down the columns.

Click the **Open Subsystem** button to open the block's subsystem. Click-and-drag blocks into this subsystem to define the processing operation(s) the block performs on the submatrices. The input to this subsystem are the submatrices whose size is determined by the **Block size** parameter.

Note When you place an Assignment block inside a Block Processing block's subsystem, the Assignment block behaves as though it is inside a For Iterator block. For a description of this behavior, see the "Iterated Assignment" section of the Assignment block reference page.

Dialog Box

The Block Processing dialog box appears as shown in the following figure.
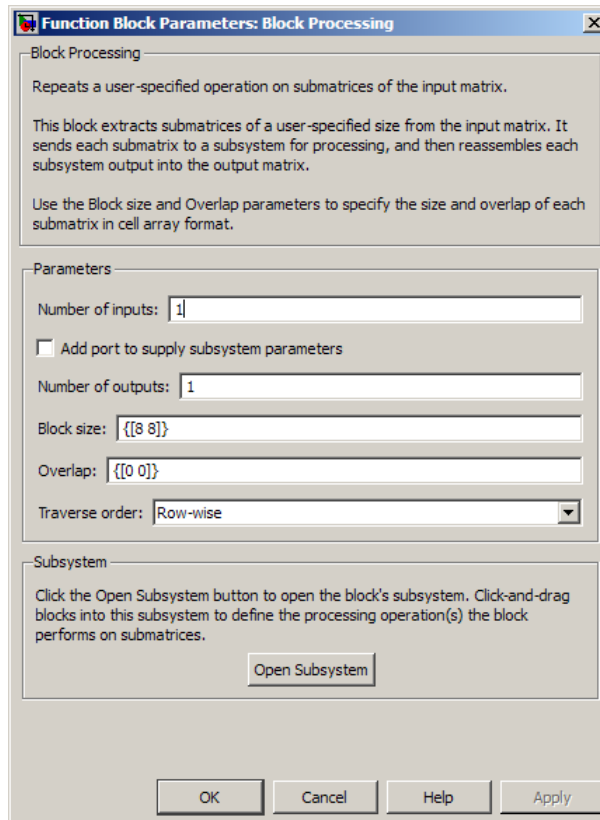
Number of inputs
   Enter the number of input ports on the Block Processing block.

Add port to supply subsystem parameters
   Add an input port to the block to supply subsystem parameters.

Number of outputs
   Enter the number of output ports on the Block Processing block.

Block Processing

Block size

Specify the size of each submatrix in cell array format. Each vector in the cell array corresponds to one input.

Overlap

Specify the overlap of each submatrix in cell array format. Each vector in the cell array corresponds to the overlap of one input.

Traverse order

Determines how the block extracts submatrices from the input matrix. If you select `Row-wise`, the block extracts submatrices by moving across the rows. If you select `Column-wise`, the block extracts submatrices by moving down the columns.

Open Subsystem

Click this button to open the block's subsystem. Click-and-drag blocks into this subsystem to define the processing operation(s) the block performs on the submatrices.

See Also

For Iterator	Simulink
`blockproc`	Image Processing Toolbox

Purpose   Perform bottom-hat filtering on intensity or binary images

Library   Morphological Operations

visionmorphops

Description   Use the Bottom-hat block to perform bottom-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element. Bottom-hat filtering is the equivalent of subtracting the input image from the result of performing a morphological closing operation on the input image. This block uses flat structuring elements only.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of ones and zeros that represents the neighborhood values	Boolean	No
Output	Scalar, vector, or matrix that represents the filtered image	Same as I port	No

If your input image is a binary image, for the **Input image type** parameter, select Binary. If your input image is an intensity image, select Intensity.

Bottom-hat

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. You can only specify a structuring element using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the region the block moves throughout the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the use of a more efficient algorithm.

Dialog Box

The Bottom-hat dialog box appears as shown in the following figure.

Input image type

If your input image is a binary image, select `Binary`. If your input image is an intensity image, select `Intensity`.

Neighborhood or structuring element source

Specify how to enter your neighborhood or structuring element values. Select `Specify via dialog` to enter the values in the dialog box. Select `Input port` to use the Nhood port to specify the neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the `strel` function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or structuring element source** parameter, you select `Specify via dialog`.

See Also

Closing	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Erosion	Video and Image Processing Blockset software
Label	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
imbothat	Image Processing Toolbox software
strel	Image Processing Toolbox software

Chroma Resampling

Purpose   Downsample or upsample chrominance components of images

Library   Conversions

visionconversions

Description   The Chroma Resampling block downsamples or upsamples chrominance components of pixels to reduce the bandwidth required for transmission or storage of a signal.

Port	Input/Output	Supported Data Types	Complex Values Supported
Cb	Matrix that represents one chrominance component of an image	• Double-precision floating point • Single-precision floating point • 8-bit unsigned integer	No
Cr	Matrix that represents one chrominance component of an image	Same as Cb port	No

The data type of the output signals is the same as the data type of the input signals.

Chroma Resampling Formats

The Chroma Resampling block supports the formats shown in the following diagram.

4:4:4   4:2:2   4:1:1

4:2:0 MPEG1   4:2:0 MPEG2

◯ Y pixel

△ Cb and Cr pixel
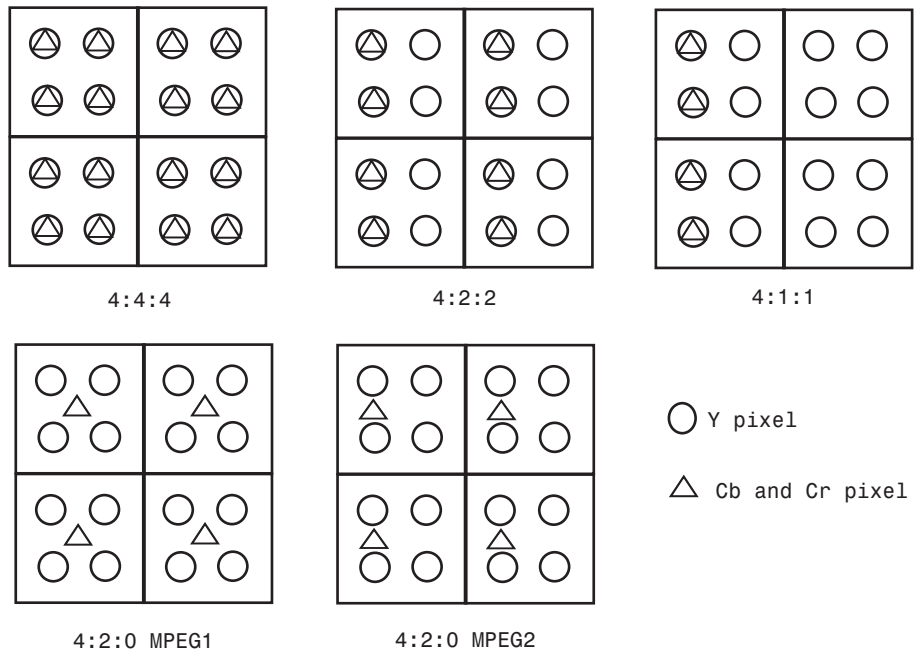
Downsampling

If, for the **Resampling** parameter, you select 4:4:4 to 4:2:2,
4:4:4 to 4:2:0 (MPEG1), 4:4:4 to 4:2:0 (MPEG2),
4:4:4 to 4:1:1, 4:2:2 to 4:2:0 (MPEG1), or
4:2:2 to 4:2:0 (MPEG2), the block performs a downsampling
operation. When the block downsamples from one format to another, it
can bandlimit the input signal by applying a lowpass filter to prevent
aliasing.

If, for the **Antialiasing filter** parameter, you select Default, the block
uses a built-in lowpass filter to prevent aliasing.

If, for the **Resampling** parameter, you select 4:4:4 to 4:2:2,
4:4:4 to 4:2:0 (MPEG1), 4:4:4 to 4:2:0 (MPEG2), or
4:4:4 to 4:1:1 and, for the **Antialiasing filter** parameter, you select

User-defined, the **Horizontal filter coefficients** parameter appears on the dialog box. Enter the filter coefficients to apply to your input.

If, for the **Resampling** parameter, you select 4:4:4 to 4:2:0 (MPEG1), 4:4:4 to 4:2:0 (MPEG2), 4:2:2 to 4:2:0 (MPEG1), or 4:2:2 to 4:2:0 (MPEG2) and, for the **Antialiasing filter** parameter, you select User-defined. **Vertical filter coefficients** parameters appear on the dialog box. Enter an even number of filter coefficients to apply to your input signal.

If, for the **Antialiasing filter** parameter, you select None, the block does not filter the input signal.

Upsampling

If, for the **Resampling** parameter, you select 4:2:2 to 4:4:4, 4:2:0 (MPEG1) to 4:2:2, 4:2:0 (MPEG1) to 4:4:4, 4:2:0 (MPEG2) to 4:2:2, 4:2:0 (MPEG2) to 4:4:4, or 4:1:1 to 4:4:4, the block performs an upsampling operation.

When the block upsamples from one format to another, it uses interpolation to approximate the missing chrominance values. If, for the **Interpolation** parameter, you select Linear, the block uses linear interpolation to calculate the missing values. If, for the **Interpolation** parameter, you select Pixel replication, the block replicates the chrominance values of the neighboring pixels to create the upsampled image.

Row-Major Data Format

The MATLAB environment and the Computer Vision System Toolbox software use column-major data organization. However, the Chroma Resampling block gives you the option to process data that is stored in row-major format. When you select the **Input image is transposed (data order is row major)** check box, the block assumes that the input buffer contains contiguous data elements from the first row first, then data elements from the second row second, and so on through the last row. Use this functionality only when you meet all the following criteria:

- You are developing algorithms to run on an embedded target that uses the row-major format.

- You want to limit the additional processing required to take the transpose of signals at the interfaces of the row-major and column-major systems.

When you use the row-major functionality, you must consider the following issues:

- When you select this check box, the signal dimensions of the Chroma Resampling block's input are swapped.

- All the Computer Vision System Toolbox blocks can be used to process data that is in the row-major format, but you need to know the image dimensions when you develop your algorithms.

  For example, if you use the 2-D FIR Filter block, you need to verify that your filter coefficients are transposed. If you are using the Rotate block, you need to use negative rotation angles, etc.

- Only three blocks have the **Input image is transposed (data order is row major)** check box. They are the Chroma Resampling, Deinterlacing, and Insert Text blocks. You need to select this check box to enable row-major functionality in these blocks. All other blocks must be properly configured to process data in row-major format.

Use the following two-step workflow to develop algorithms in row-major format to run on an embedded target.

Chroma Resampling

Step 1:
Create block diagram

Algorithm blocks

Video source block

Transpose block

Transpose block

Video sink block

Embedded target source block

Step 2:
Replace source, transpose, and sink blocks with target source and sink blocks that produce data in row-major format

Embedded target sink block
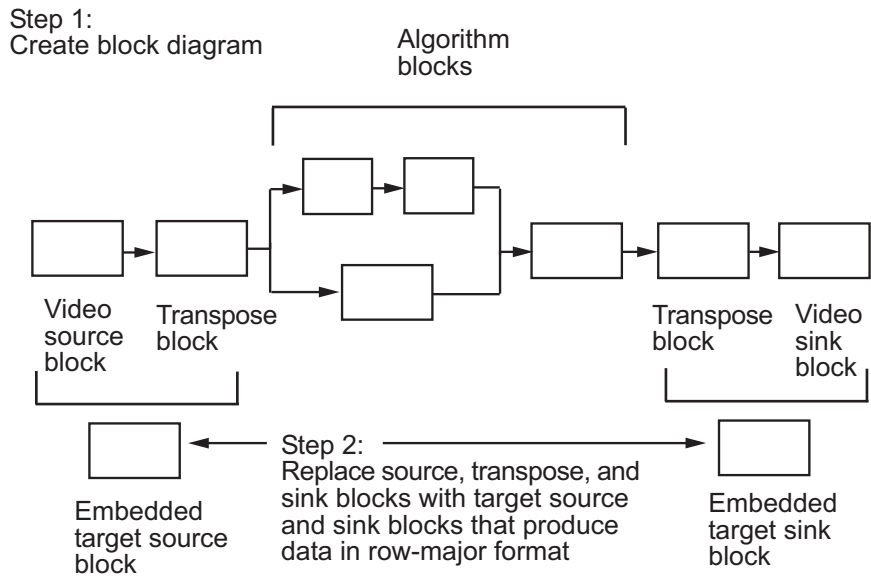
See the DM642 EVM Video ADC and DM642 EVM Video DAC reference pages.

Dialog Box

The Chroma Resampling dialog box appears as shown in the following figure.
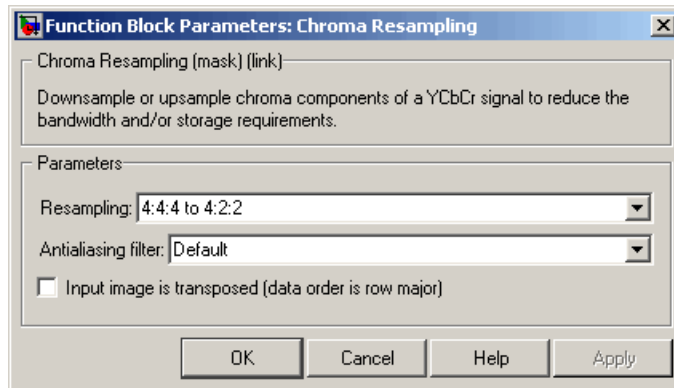
Resampling

Specify the resampling format.

Antialiasing filter

Specify the lowpass filter that the block uses to prevent aliasing. If you select Default, the block uses a built-in lowpass filter. If you select User-defined, the **Horizontal filter coefficients** and/or **Vertical filter coefficients** parameters appear on the dialog box. If you select None, the block does not filter the input signal. This parameter is visible when you are downsampling the chrominance values.

Horizontal filter coefficients

Enter the filter coefficients to apply to your input signal. This parameter is visible if, for the **Resampling** parameter, you select 4:4:4 to 4:2:2, 4:4:4 to 4:2:0 (MPEG1), 4:4:4 to 4:2:0 (MPEG2), or 4:4:4 to 4:1:1 and, for the **Antialiasing filter** parameter, you select User-defined.

Vertical filter coefficients

Enter the filter coefficients to apply to your input signal. This parameter is visible if, for the **Resampling** parameter, you

select `4:4:4 to 4:2:0 (MPEG1)`, `4:4:4 to 4:2:0 (MPEG2)`, `4:2:2 to 4:2:0 (MPEG1)`, or `4:2:2 to 4:2:0 (MPEG2)` and, for the **Antialiasing filter** parameter, you select `User-defined`.

Interpolation

Specify the interpolation method that the block uses to approximate the missing chrominance values. If you select `Linear`, the block uses linear interpolation to calculate the missing values. If you select `Pixel replication`, the block replicates the chrominance values of the neighboring pixels to create the upsampled image. This parameter is visible when you are upsampling the chrominance values. This parameter is visible if the **Resampling** parameter is set to `4:2:2 to 4:4:4`, `4:2:0 (MPEG1) to 4:4:4`, `4:2:0 (MPEG2) to 4:4:4`, `4:1:1 to 4:4:4`, `4:2:0 (MPEG1) to 4:2:2`, or `4:2:0 (MPEG2) to 4:2:2`.

Input image is transposed (data order is row major)

When you select this check box, the block assumes that the input buffer contains data elements from the first row first, then data elements from the second row second, and so on through the last row.

References

[1] Haskell, Barry G., Atul Puri, and Arun N. Netravali. *Digital Video: An Introduction to MPEG-2.* New York: Chapman & Hall, 1996.

[2] Recommendation ITU-R BT.601-5, Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide Screen 16:9 Aspect Ratios.

[3] Wang, Yao, Jorn Ostermann, Ya-Qin Zhang. *Video Processing and Communications.* Upper Saddle River, NJ: Prentice Hall, 2002.

See Also

Autothreshold	Computer Vision System Toolbox software
Color Space Conversion	Computer Vision System Toolbox software
Image Complement	Computer Vision System Toolbox software

Purpose	Perform morphological closing on binary or intensity images
Library	Morphological Operations
	`visionmorphops`

Description   The Closing block performs a dilation operation followed by an erosion operation using a predefined neighborhood or structuring element. This block uses flat structuring elements only.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of ones and zeros that represents the neighborhood values	Boolean	No
Output	Vector or matrix of intensity values that represents the closed image	Same as I port	No

The output signal has the same data type as the input to the I port.

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input

port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. You can only specify a structuring element using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the region the block moves throughout the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the use of a more efficient algorithm.

Dialog Box

The Closing dialog box appears as shown in the following figure.

Neighborhood or structuring element source

> Specify how to enter your neighborhood or structuring element values. Select Specify via dialog to enter the values in the dialog box. Select Input port to use the Nhood port to specify the

neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the `strel` function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or structuring element source** parameter, you select `Specify via dialog`.

References   [1] Soille, Pierre. *Morphological Image Analysis*. 2nd ed. New York: Springer, 2003.

See Also

Bottom-hat	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Erosion	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
imclose	Image Processing Toolbox software
strel	Image Processing Toolbox software

Purpose   Convert color information between color spaces

Library   Conversions

visionconversions

Description   The Color Space Conversion block converts color information between
color spaces. Use the **Conversion** parameter to specify the color spaces
you are converting between. Your choices are R'G'B' to Y'CbCr,
Y'CbCr to R'G'B', R'G'B' to intensity, R'G'B' to HSV, HSV to
R'G'B', sR'G'B' to XYZ, XYZ to sR'G'B', sR'G'B' to L*a*b*, and
L*a*b* to sR'G'B'.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input / Output	M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • 8-bit unsigned integer	No
R'	Matrix that represents one plane of the input RGB video stream	Same as the Input port	No
G'	Matrix that represents one plane of the input RGB video stream	Same as the Input port	No
B'	Matrix that represents one plane of the input RGB video stream	Same as the Input port	No
Y'	Matrix that represents the luma portion of an image	Same as the Input port	No

Color Space Conversion

Port	Input/Output	Supported Data Types	Complex Values Supported
Cb	Matrix that represents one chrominance component of an image	Same as the Input port	No
Cr	Matrix that represents one chrominance component of an image	Same as the Input port	No
I'	Matrix of intensity values	Same as the Input port	No
H	Matrix that represents the hue component of an image	• Double-precision floating point • Single-precision floating point	No
S	Matrix that represents represent the saturation component of an image	Same as the H port	No
V	Matrix that represents the value (brightness) component of an image	Same as the H port	No
X	Matrix that represents the X component of an image	Same as the H port	No
Y	Matrix that represents the Y component of an image	Same as the H port	No
Z	Matrix that represents the Z component of an image	Same as the H port	No
L*	Matrix that represents the luminance portion of an image	Same as the H port	No
a*	Matrix that represents the a* component of an image	Same as the H port	No
b*	Matrix that represents the b* component of an image	Same as the H port	No

The data type of the output signal is the same as the data type of the input signal.

Use the **Image signal** parameter to specify how to input and output a color video signal. If you select `One multidimensional signal`, the block accepts an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port accepts one M-by-N plane of an RGB video stream.

Note The prime notation indicates that the signals are gamma corrected.

Conversion Between R'G'B' and Y'CbCr Color Spaces

The R'G'B' to Y'CbCr conversion and the Y'CbCr to R'G'B' conversion are defined by the following equations:

$$
\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + A \times \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}
$$

$$
\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = B \times \left(\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right)
$$

The values in the A and B matrices are based on your choices for the **Use conversion specified by** and **Scanning standard** parameters. The following table summarizes the possible values:

Color Space Conversion

Matrix	Use conversion specified by = Rec. 601 (SDTV)	Use conversion specified by = Rec. 709 (HDTV)	
		Scanning standard = 1125/60/2:1	Scanning standard = 1250/50/2:1
A	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$	$\begin{bmatrix} 0.18258588 & 0.61423059 & 0.06200706 \\ -0.10064373 & -0.33857195 & 0.43921569 \\ 0.43921569 & -0.39894216 & -0.04027352 \end{bmatrix}$	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$
B	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 0.16438356 & 2.0172321 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.16438356 & 0 & 1.79274107 \\ 1.16438356 & -0.21324861 & -0.53290933 \\ 1.16438356 & 2.11240179 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 0.16438356 & 2.0172321 & 0 \end{bmatrix}$

Conversion R'B'G' to Intensity

The conversion from the R'B'G' color space to intensity is defined by the following equation:

$$\text{intensity} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Conversion Between R'G'B' and HSV Color Spaces

The R'G'B' to HSV conversion is defined by the following equations. In these equations, *MAX* and *MIN* represent the maximum and minimum values of each R'G'B' triplet, respectively. *H*, *S*, and *V* vary from 0 to 1, where 1 represents the greatest saturation and value.

$$H = \begin{cases} \left(\dfrac{G' - B'}{MAX - MIN} \right)/6, & \text{if} \quad R' = MAX \\[3ex] \left(2 + \dfrac{B' - R'}{MAX - MIN} \right)/6, & \text{if} \quad G' = MAX \\[3ex] \left(4 + \dfrac{R' - G'}{MAX - MIN} \right)/6, & \text{if} \quad B' = MAX \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

The HSV to R'G'B' conversion is defined by the following equations:

$$H_i = \lfloor 6H \rfloor$$
$$f = 6H - H_i$$
$$p = 1 - S$$
$$q = 1 - fS$$
$$t = 1 - (1 - f)S$$

$$\text{if} \quad H_i = 0, \quad R_{tmp} = 1, \quad G_{tmp} = t, \quad B_{tmp} = p$$
$$\text{if} \quad H_i = 1, \quad R_{tmp} = q, \quad G_{tmp} = 1, \quad B_{tmp} = p$$
$$\text{if} \quad H_i = 2, \quad R_{tmp} = p, \quad G_{tmp} = 1, \quad B_{tmp} = t$$
$$\text{if} \quad H_i = 3, \quad R_{tmp} = p, \quad G_{tmp} = q, \quad B_{tmp} = 1$$
$$\text{if} \quad H_i = 4, \quad R_{tmp} = t, \quad G_{tmp} = p, \quad B_{tmp} = 1$$
$$\text{if} \quad H_i = 5, \quad R_{tmp} = 1, \quad G_{tmp} = p, \quad B_{tmp} = q$$

$$u = V / \max(R_{tmp}, G_{tmp}, B_{tmp})$$
$$R' = uR_{tmp}$$
$$G' = uG_{tmp}$$
$$B' = uB_{tmp}$$

For more information about the HSV color space, see "HSV Color Space" in the Image Processing Toolbox documentation.

Color Space Conversion

Conversion Between sR'G'B' and XYZ Color Spaces

The sR'G'B' to XYZ conversion is a two-step process. First, the block converts the gamma-corrected sR'G'B' values to linear sRGB values using the following equations:

$$\text{If} \quad R'_{sRGB}, G'_{sRGB}, B'_{sRGB} \leq 0.03928$$

$$R_{sRGB} = R'_{sRGB} / 12.92$$

$$G_{sRGB} = G'_{sRGB} / 12.92$$

$$B_{sRGB} = B'_{sRGB} / 12.92$$

$$\text{otherwise,} \quad \text{if} \quad R'_{sRGB}, G'_{sRGB}, B'_{sRGB} > 0.03928$$

$$R_{sRGB} = \left[\left(R'_{sRGB} + 0.055 \right) \Big/ 1.055 \right]^{2.4}$$

$$G_{sRGB} = \left[\left(G'_{sRGB} + 0.055 \right) \Big/ 1.055 \right]^{2.4}$$

$$B_{sRGB} = \left[\left(B'_{sRGB} + 0.055 \right) \Big/ 1.055 \right]^{2.4}$$

Then the block converts the sRGB values to XYZ values using the following equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41239079926596 & 0.35758433938388 & 0.18048078840183 \\ 0.21263900587151 & 0.71516867876776 & 0.07219231536073 \\ 0.01933081871559 & 0.11919477979463 & 0.95053215224966 \end{bmatrix} \times \begin{bmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{bmatrix}$$

The XYZ to sR'G'B' conversion is also a two-step process. First, the block converts the XYZ values to linear sRGB values using the following equation:

$$\begin{bmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{bmatrix} = \begin{bmatrix} 0.41239079926596 & 0.35758433938388 & 0.18048078840183 \\ 0.21263900587151 & 0.71516867876776 & 0.07219231536073 \\ 0.01933081871559 & 0.11919477979463 & 0.95053215224966 \end{bmatrix}^{-1} \times \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Then the block applies gamma correction to obtain the sR'G'B' values. This process is described by the following equations:

If $\quad R_{sRGB}, G_{sRGB}, B_{sRGB} \leq 0.00304$

$R'_{sRGB} = 12.92 R_{sRGB}$

$G'_{sRGB} = 12.92 G_{sRGB}$

$B'_{sRGB} = 12.92 B_{sRGB}$

otherwise, $\quad$ if $\quad R_{sRGB}, G_{sRGB}, B_{sRGB} > 0.00304$

$R'_{sRGB} = 1.055 R_{sRGB}^{(1.0/2.4)} - 0.055$

$G'_{sRGB} = 1.055 G_{sRGB}^{(1.0/2.4)} - 0.055$

$B'_{sRGB} = 1.055 B_{sRGB}^{(1.0/2.4)} - 0.055$

Note Computer Vision System Toolbox software uses a D65 white point, which is specified in Recommendation ITU-R BT.709, for this conversion. In contrast, the Image Processing Toolbox conversion is based on ICC profiles, and it uses a D65 to D50 Bradford adaptation transformation to the D50 white point. If you are using these two products and comparing results, you must account for this difference.

Conversion Between sR'G'B' and L*a*b* Color Spaces

The Color Space Conversion block converts sR'G'B' values to L*a*b* values in two steps. First it converts sR'G'B' to XYZ values using the equations described in "Conversion Between sR'G'B' and XYZ Color Spaces" on page 1-262. Then it uses the following equations to transform the XYZ values to L*a*b* values. Here, X_n, Y_n, and Z_n are the tristimulus values of the reference white point you specify using the **White point** parameter:

$$L* = 116(Y/Y_n)^{1/3} - 16, for \quad Y/Y_n > 0.008856$$
$$L* = 903.3\,Y/Y_n, \quad \text{otherwise}$$

$$a* = 500(f(X/X_n) - f(Y/Y_n))$$
$$b* = 200(f(Y/Y_n) - f(Z/Z_n)),$$
$$\text{where} \quad f(t) = t^{1/3}, for \quad t > 0.008856$$
$$f(t) = 7.787t + 16/166, \quad \text{otherwise}$$

The block converts L*a*b* values to sR'G'B' values in two steps as well. The block transforms the L*a*b* values to XYZ values using these equations:

For $\quad Y/Y_n > 0.008856$

$$X = X_n(P + a*/500)^3$$
$$Y = Y_n P^3$$
$$Z = Z_n(P - b*/200)^3,$$

where $\quad P = (L* + 16)/116$

Dialog Box

The Color Space Conversion dialog box appears as shown in the following figure.
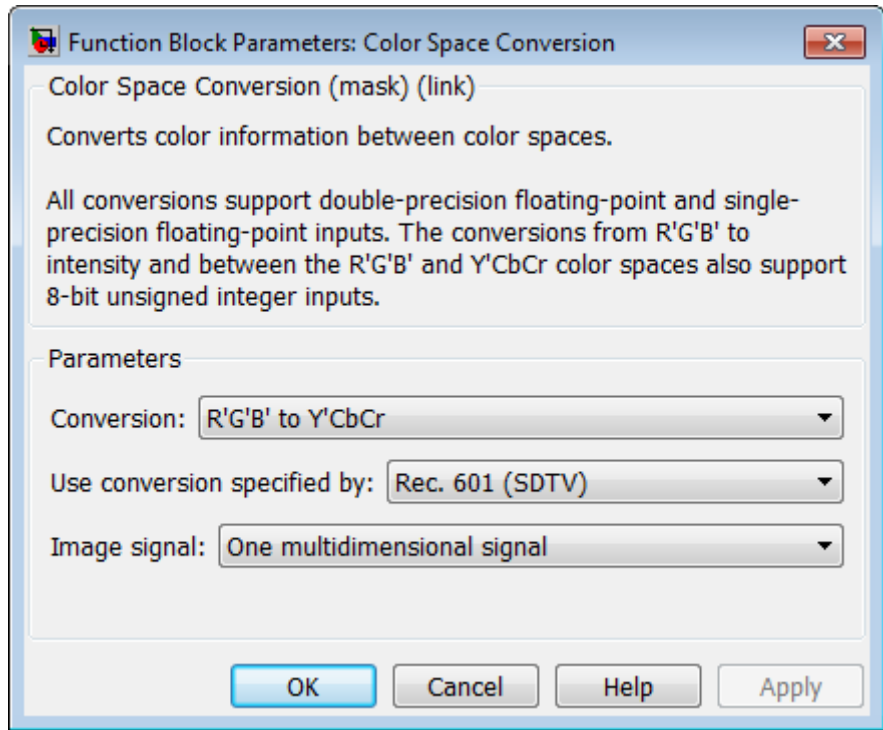
Conversion

Specify the color spaces you are converting between. Your choices are R'G'B' to Y'CbCr, Y'CbCr to R'G'B', R'G'B' to intensity, R'G'B' to HSV, HSV to R'G'B', sR'G'B' to XYZ, XYZ to sR'G'B', sR'G'B' to L*a*b*, and L*a*b* to sR'G'B'.

Use conversion specified by

Specify the standard to use to convert your values between the R'G'B' and Y'CbCr color spaces. Your choices are Rec. 601 (SDTV) or Rec. 709 (HDTV). This parameter is only available

if, for the **Conversion** parameter, you select `R'G'B' to Y'CbCr` or `Y'CbCr to R'G'B'`.

Scanning standard

Specify the scanning standard to use to convert your values between the R'G'B' and Y'CbCr color spaces. Your choices are `1125/60/2:1` or `1250/50/2:1`. This parameter is only available if, for the **Use conversion specified by** parameter, you select `Rec. 709 (HDTV)`.

White point

Specify the reference white point. This parameter is visible if, for the **Conversion** parameter, you select `sR'G'B' to L*a*b*` or `L*a*b* to sR'G'B'`.

Image signal

Specify how to input and output a color video signal. If you select `One multidimensional signal`, the block accepts an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port accepts one M-by-N plane of an RGB video stream.

References

[1] Poynton, Charles A. *A Technical Introduction to Digital Video*. New York: John Wiley & Sons, 1996.

[2] Recommendation ITU-R BT.601-5, Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide Screen 16:9 Aspect Ratios.

[3] Recommendation ITU-R BT.709-5. Parameter values for the HDTV standards for production and international programme exchange.

[4] Stokes, Michael, Matthew Anderson, Srinivasan Chandrasekar, and Ricardo Motta, "A Standard Default Color Space for the Internet - sRGB." November 5, 1996.

[5] Berns, Roy S. *Principles of Color Technology, 3rd ed.* New York: John Wiley & Sons, 2000.

See Also

Chroma Resampling	Computer Vision System Toolbox software
rgb2hsv	MATLAB software
hsv2rgb	MATLAB software
rgb2ycbcr	Image Processing Toolbox software
ycbcr2rgb	Image Processing Toolbox software
rgb2gray	Image Processing Toolbox software
makecform	Image Processing Toolbox software
applycform	Image Processing Toolbox software

Compositing

Purpose   Combine pixel values of two images, overlay one image over another, or highlight selected pixels

Library   Text & Graphics

visiontextngfix

Description

You can use the Compositing block to combine two images. Each pixel of the output image is a linear combination of the pixels in each input image. This process is defined by the following equation:

$$O(i, j) = (1 - X) * I1(i, j) + X * I2(i, j)$$

You can define the amount by which to scale each pixel value before combining them using the opacity factor, X, where , $0 \leq X \leq 1$.

You can use the Compositing block to overlay one image over another image. The masking factor and the location determine which pixels are overwritten. Masking factors can be 0 or 1, where 0 corresponds to not overwriting pixels and 1 corresponds to overwriting pixels.

You can also use this block to highlight selected pixels in the input image. The block uses a binary input image at the **Mask** port, to specify which pixels to highlight.

Note This block supports intensity and color images.

Fixed-Point Data Types

The following diagram shows the data types used in the Compositing
block for fixed-point signals. These data types applies when the
Operation parameter is set to Blend.
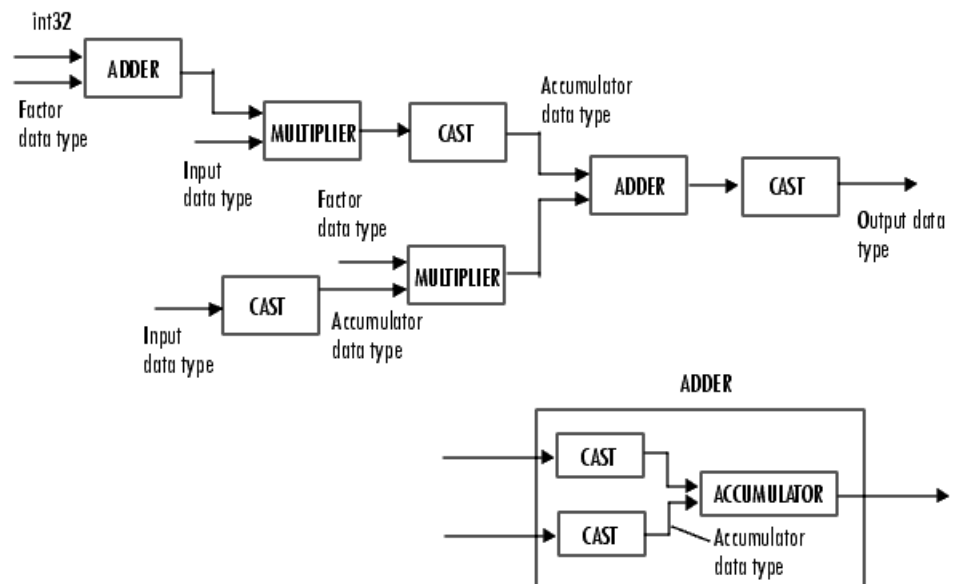
You can set the product output, accumulator, and output data types in
the block mask as discussed in the next section.

Compositing

The **Main** pane of the Compositing dialog box appears as shown in the following figure.

Function Block Parameters: Compositing

Compositing

Combines the pixel values of two images, overlays one image over another, or highlights s

If, for the Operation parameter, you choose Blend, you can use the Opacity factor(s) paran
amount by which the block scales the pixel values of Image2 before combining them with t
Image1. This parameter can be a scalar value that the block uses for all pixels or a matrix
defines the factor for each pixel.

If, for the Operation parameter, you choose Binary mask, you can use the Masking factor(s
define which pixels of Image1 are overwritten by the pixel values of Image2. You can enter
block uses for all pixels, or a matrix of 0's and 1's that defines the factor for each pixel.

If, for the Operation parameter, you choose Highlight selected pixel, the block uses the bina
the Mask port to determine which pixels are set to the maximum value supported by their

| Main | Data Types |

Parameters

Operation: | Blend |

Opacity factor(s) source: | Specify via dialog |

Opacity factor(s): 0.75

Location source: | Specify via dialog |

Location [x y]: [1 1]

OK   Cancel   He

Operation

Specify the operation you want the block to perform. If you choose Blend, the block linearly combines the pixels of one image with another image. If you choose Binary mask, the block overwrites the pixel values of one image with the pixel values of another image. If you choose Highlight selected pixels, the block uses the binary image input at the **Mask** port. Using this image, the block then determines which pixels are set to the maximum value supported by their data type.

Blend

If, for the **Operation** parameter, you choose Blend, the **Opacity factor(s) source** parameter appears on the dialog box. Use this parameter to indicate where to specify the opacity factor(s).

- If you choose Specify via dialog, the **Opacity factor(s)** parameter appears on the dialog box. Use this parameter to define the amount by which the block scales each pixel values for input image at the **Image2** port before combining them with the pixel values of the input image at **Image1** port. You can enter a scalar value used for all pixels or a matrix of values that is the same size as the input image at the **Image2** port.

- If you choose Input port, the **Factor** port appears on the block. The input to this port must be a scalar or matrix of values as described for the **Opacity factor(s)** parameter. If the input to the **Image1** and **Image2** ports is floating point, the input to this port must be the same floating-point data type.

Binary mask

If, for the **Operation** parameter, you choose Binary mask, the **Mask source** parameter appears on the dialog box. Use this parameter to indicate where to specify the masking factor(s).

- If you choose Specify via dialog, the **Mask** parameter appears on the dialog box. Use this parameter and the location source of the image to define which pixels are overwritten. You

can enter `0` or `1` to use for all pixels in the image, or a matrix of `0`s and `1`s that defines the factor for each pixel.

- If you choose `Input port`, the **Factor** port appears on the block. The input to this port must be a `0` or `1` whose data type is Boolean. Or, a matrix of `0`s or `1`s whose data type is Boolean, as described for the **Mask** parameter.

Highlight selected pixels

If, for the **Operation** parameter, you choose `Highlight selected pixels`, the block uses the binary image input at the **Mask** port to determine which pixels are set to the maximum value supported by their data type. For example, for every pixel value set to `1` in the binary image, the block sets the corresponding pixel in the input image to the maximum value supported by its data type. For every `0` in the binary image, the block leaves the corresponding pixel value alone.

Opacity factor(s) source

Indicate where to specify any opacity factors. Your choices are `Specify via dialog` and `Input port`. This parameter is visible if, for the **Operation** parameter, you choose `Blend`.

Opacity factor(s)

Define the amount by which the block scales each pixel value before combining them. You can enter a scalar value used for all pixels or a matrix of values that defines the factor for each pixel. This parameter is visible if, for the **Opacity factor(s) source** parameter, you choose `Specify via dialog`. Tunable.

Mask source

Indicate where to specify any masking factors. Your choices are `Specify via dialog` and `Input port`. This parameter is visible if, for the **Operation** parameter, you choose `Binary mask`.

Mask

Define which pixels are overwritten. You can enter 0 or 1, which is used for all pixels, or a matrix of 0s and 1s that defines the

factor for each pixel. This parameter is visible if, for the **Mask source** parameter, you choose `Specify via dialog`. Tunable.

Location source

Use this parameter to specify where to enter the location of the upper-left corner of the image input at input port **Image2**. You can choose either `Specify via dialog` or `Input port`.

When you choose `Specify via dialog`, you can set the **Location [x y]** parameter.

When you choose `Input port`, the **Location** port appears on the block. The input to this port must be a two-element vector as described for the **Location [x y]** parameter.

Location [x y]

Enter a two-element vector that specifies the row and column position of the upper-left corner of the image input at **Image2** port. The position is relative to the upper-left corner of the image input at **Image1** port. This parameter is visible if, for the **Location source** parameter, you choose `Specify via dialog`. Tunable.

Positive values move the image down and to the right; negative values move the image up and to the left. If the first element is greater than the number of rows in the **Image1** matrix, the value is clipped to the total number of rows. If the second element is greater than the number of columns in the input **Image1** matrix, the value is clipped to the total number of columns.

The **Data Types** pane of the Compositing dialog box appears as follows. These parameters apply only when the **Operation** parameter is set to `Blend`.

Compositing

Function Block Parameters: Compositing

Compositing

Combines the pixel values of two images, overlays one image over another, or highlights s

If, for the Operation parameter, you choose Blend, you can use the Opacity factor(s) param
amount by which the block scales the pixel values of Image2 before combining them with t
Image1. This parameter can be a scalar value that the block uses for all pixels or a matrix
defines the factor for each pixel.

If, for the Operation parameter, you choose Binary mask, you can use the Masking factor(s
define which pixels of Image1 are overwritten by the pixel values of Image2. You can enter
block uses for all pixels, or a matrix of 0's and 1's that defines the factor for each pixel.

If, for the Operation parameter, you choose Highlight selected pixel, the block uses the bina
the Mask port to determine which pixels are set to the maximum value supported by their

Main	Data Types

Floating-point inheritance takes precedence over the settings in the 'Data Type' column bel
block input is floating point, all block data types match the input.

Fixed-point operational parameters

Rounding mode: Floor ▼   Overflow mode: Wrap ▼

Fixed-point data types

	Data Type	Signed	Word length	Fractio
Opacity factor	Same word length as input ▼			
Product output	Binary point scaling ▼	Yes	32	10
Accumulator	Same as product output ▼			
Output	Same as first input ▼			

☐ Lock data type settings against changes by the fixed-point tools

OK   Cancel   He

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Opacity factor

Choose how to specify the word length and fraction length of the opacity factor:

- When you select Same word length as input, these characteristics match those of the input to the block.

- When you select Specify word length, enter the word length of the opacity factor.

- When you select Binary point scaling, you can enter the word length of the opacity factor, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, of the opacity factor. The bias of all signals in the Computer Vision System Toolbox software is 0.

Product output

As the previous figure shows, the block places the output of the multiplier into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

- When you select Same as first input, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Accumulator

The result of each addition remains in the accumulator data type.

Input to adder - product output data type → CAST → Accumulator data type → ADDER → Accumulator data type

As the previous figure shows, the block takes inputs to the accumulator and casts them to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software software is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Supported Data Types

Port	Input/Output	Supported Data Types	Complex Values Supported
Image 1	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color video signal where *P* is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer	No

Port	Input/Output	Supported Data Types	Complex Values Supported
		• 8-, 16-, and 32-bit unsigned integer	
Image 2	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color video signal where *P* is the number of color planes	Same as Image 1 port	No
Factor	Scalar or matrix of opacity or masking factor	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Mask	Binary image that specifies which pixels to highlight	Same as Factor port When the **Operation** parameter is set to `Highlight selected pixel`, the input to the Mask port must be a Boolean data type.	No

Port	Input/Output	Supported Data Types	Complex Values Supported
Location	Two-element vector [*x y*], that specifies the position of the upper-left corner of the image input at port I2	• Double-precision floating point. (Only supported if the input to the Image 1 and Image 2 ports is a floating-point data type.) • Single-precision floating point. (Only supported if the input to the Image 1 and Image 2 ports is a floating-point data type.) • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Output	Vector or matrix of intensity or color values	Same as Image 1 port	No

See Also

Compositing (To Be Removed)

Purpose   Combine pixel values of two images, overlay one image over another, or highlight selected pixels

Library   Text & Graphics

viptextngfix

Description
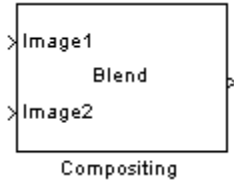
Note This Compositing block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Compositing block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Compositing documentation to view the reference material related to the obsolete block.

Purpose   Adjust image contrast by linearly scaling pixel values

Library   Analysis & Enhancement

                   visionanalysis

Description   The Contrast Adjustment block adjusts the contrast of an image by
                   linearly scaling the pixel values between upper and lower limits. Pixel
                   values that are above or below this range are saturated to the upper
                   or lower limit value, respectively.
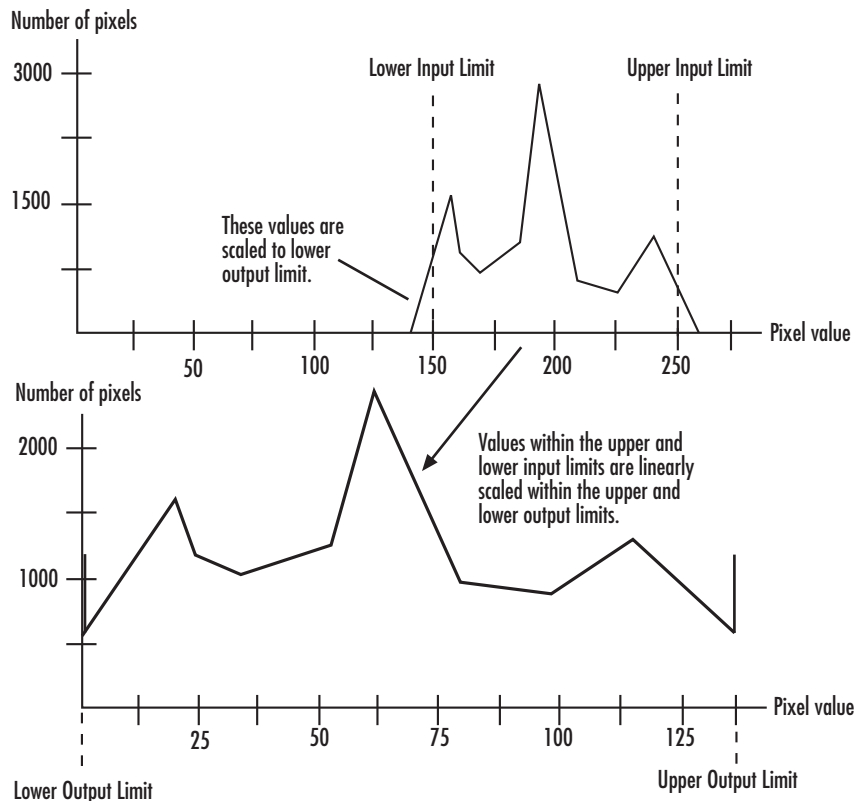
Contrast Adjustment

Mathematically, the contrast adjustment operation is described by the following equation, where the input limits are [*low_in high_in*] and the output limits are [*low_out high_out*]:

$$Output = \begin{cases} low\_out, & Input \leq low\_in \\ low\_out + (Input - low\_in)\dfrac{high\_out - low\_out}{high\_in - low\_in}, & low\_in < Input < high\_in \\ high\_out, & Input \geq high\_in \end{cases}$$

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Output	Scalar, vector, or matrix of intensity values or a scalar, vector, or matrix that represents one plane of the RGB video stream	Same as I port	No

Specifying upper and lower limits

Use the **Adjust pixel values from** and **Adjust pixel values to** parameters to specify the upper and lower input and output limits. All options are described below.

Input limits

Use the **Adjust pixel values from** parameter to specify the upper and lower input limits.

If you select `Full input data range [min max]`, uses the minimum input value as the lower input limit and the maximum input value as the upper input limit.

If you select `User-defined`, the **Range [low high]** parameter associated with this option appears. Enter a two-element vector of scalar values, where the first element corresponds to the lower input limit and the second element corresponds to the upper input limit.

If you select `Range determined by saturating outlier pixels`, the **Percentage of pixels to saturate [low high] (in %)**, **Specify number of histogram bins (used to calculate the range when outliers are eliminated)**, and **Number of histogram bins** parameters appear on the block. The block uses these parameter values to calculate the input limits in this three-step process:

1 Find the minimum and maximum input values, [*min_in max_in*].

2 Scale the pixel values from [**min_in max_in**] to [0 *num_bins*-1], where *num_bins* is the scalar value you specify in the **Number of histogram bins** parameter. This parameter always displays the value used by the block. Then the block calculates the histogram of the scaled input. For additional information about histograms, see the 2D-Histogram block reference page.

3 Find the lower input limit such that the percentage of pixels with values smaller than the lower limit is at most the value of the first element of the **Percentage of pixels to saturate [low high] (in %)** parameter. Similarly, find the upper input limit such that the percentage of pixels with values greater than the upper limit is at least the value of the second element of the parameter.

Output limits

Use the **Adjust pixel values to** parameter to specify the upper and lower output limits.

If you select `Full data type range`, the block uses the minimum value of the input data type as the lower output limit and the maximum value of the input data type as the upper out

Contrast Adjustment

If you select User-defined range, the **Range [low high]** parameter appears on the block. Enter a two-element vector of scalar values, where the first element corresponds to the lower output limit and the second element corresponds to the upper output limit.

For INF, -INF and NAN Input Values

If any input pixel value is either INF or -INF, the Contrast Adjustment block will change the pixel value according to how the parameters are set. The following table shows how the block handles these pixel values.

If Adjust pixel values from parameter is set to...	Contrast Adjustment block will:
Full data range [min,max]	Set the entire output image to the lower limit of the **Adjust pixel values to** parameter setting.
Range determined by saturating outlier pixels	
User defined range	Lower and higher limits of the **Adjust pixel values to** parameter set to -INF and INF, respectively.

If any input pixel has a NAN value, the block maps the pixels with valid numerical values according to the user-specified method. It maps the NAN pixels to the lower limit of the **Adjust pixels values to** parameter.

Examples

See "Adjust the Contrast of Intensity Images" in the *Computer Vision System Toolbox User's Guide*.

Fixed-Point Data Types

The following diagram shows the data types used in the Contrast Adjustment block for fixed-point signals:

FIXED-POINT TAB NAMES:
======================================
IN_DT: Datatype of the input image
EXD_DT: IN_DT with extra bit for sign
P1_DT: Datatype of product 1
P2_DT: Datatype of product 2

SUB-SYSTEMS:
======================================
Search:
Calculate the pixel value that corresponds
to Thr in the cumulative histogram (CDF).
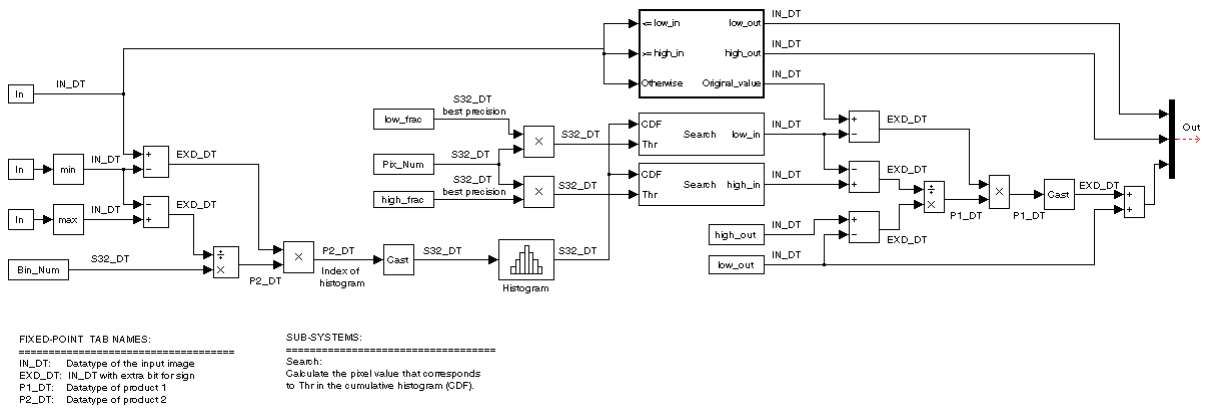
Dialog Box

The Contrast Adjustment dialog box appears as shown in the following figure.
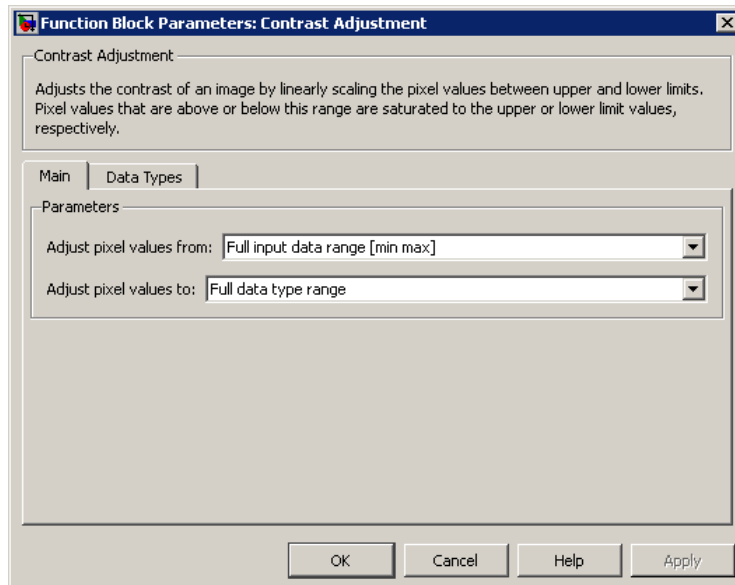
Contrast Adjustment

Adjust pixel values from

Specify how to enter the upper and lower input limits. Your choices are `Full input data range [min max]`, `User-defined`, and `Range determined by saturating outlier pixels`.

Range [low high]

Enter a two-element vector of scalar values. The first element corresponds to the lower input limit, and the second element corresponds to the upper input limit. This parameter is visible if, for the **Adjust pixel values from** parameter, you select `User-defined`.

Percentage of pixels to saturate [low high] (in %)

Enter a two-element vector. The block calculates the lower input limit such that the percentage of pixels with values smaller than the lower limit is at most the value of the first element. It calculates the upper input limit similarly. This parameter is visible if, for the **Adjust pixel values from** parameter, you select `Range determined by saturating outlier pixels`.

Specify number of histogram bins (used to calculate the range when outliers are eliminated)

Select this check box to change the number of histogram bins. This parameter is editable if, for the **Adjust pixel values from** parameter, you select `Range determined by saturating outlier pixels`.

Number of histogram bins

Enter the number of histogram bins to use to calculate the scaled input values. This parameter is available if you select the **Specify number of histogram bins (used to calculate the range when outliers are eliminated)** check box.

Adjust pixel values to

Specify the upper and lower output limits. If you select `Full data type range`, the block uses the minimum value of the input data type as the lower output limit and the maximum value of the input data type as the upper output limit. If you select `User-defined range`, the **Range [low high]** parameter appears on the block.

Range [low high]

> Enter a two-element vector of scalar values. The first element corresponds to the lower output limit and the second element corresponds to the upper output limit. This parameter is visible if, for the **Adjust pixel values to** parameter, you select User-defined range

The **Data Types** pane of the Contrast Adjustment dialog box appears as shown in the following figure.
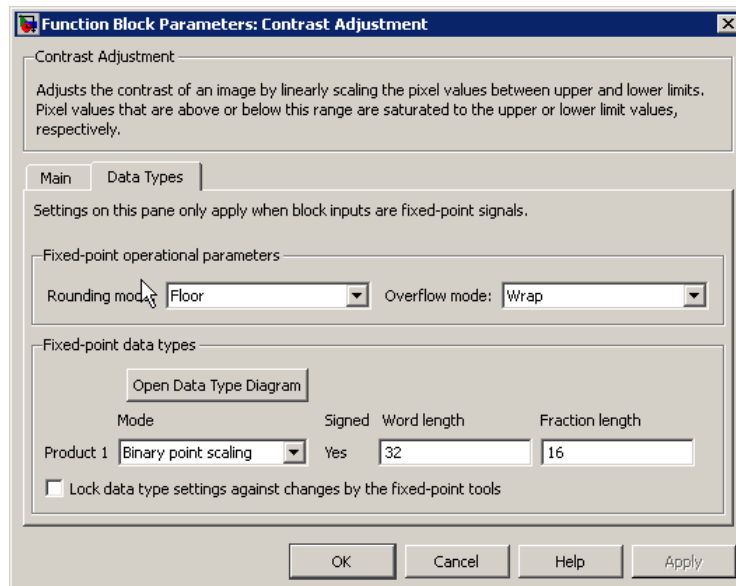
Rounding mode

> Select the rounding mode for fixed-point operations.

Overflow mode

> Select the overflow mode for fixed-point operations.

Product 1

The product output type when the block calculates the ratio between the input data range and the number of histogram bins.
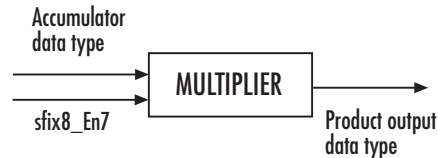
As shown in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths:

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Product 2

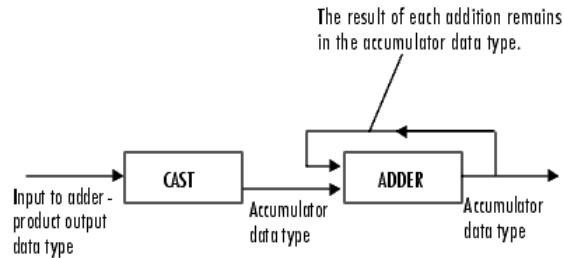The product output type when the block calculates the bin location of each input value.

As shown in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths:

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

This parameter is visible if, for the **Adjust pixel values from** parameter, you select `Range determined by saturating outlier pixels`.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also

2D-Histogram	Computer Vision System Toolbox software
Histogram Equalization	Computer Vision System Toolbox software

Corner Detection

Purpose   Calculate corner metric matrix and find corners in images

Library   Analysis & Enhancement

visionanalysis

Description
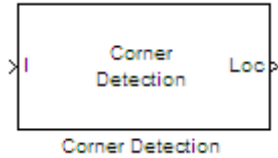
Corner Detection

The Corner Detection block finds corners in an image using the Harris corner detection (by Harris & Stephens), minimum eigenvalue (by Shi & Tomasi), or local intensity comparison (Features from Accelerated Segment Test, FAST by Rosten & Drummond) method. The block finds the corners in the image based on the pixels that have the largest corner metric values.

For the most accurate results, use the "Minimum Eigenvalue Method" on page 1-291. For the fastest computation, use the "Local Intensity Comparison" on page 1-292. For the trade-off between accuracy and computation, use the "Harris Corner Detection Method" on page 1-292.

Port Description

Port	Description	Supported Data Types
I	Matrix of intensity values	• Double-precision floating point
		• Single-precision floating point
		• Fixed point
		• 8-, 16-, and 32-bit signed integer
		• 8-, 16-, and 32-bit unsigned integer
Loc	*M*-by-2 matrix of [x y] coordinates, that represents	32-bit unsigned integer

Port	Description	Supported Data Types
	the locations of the corners. *M* represents the number of corners and is less than or equal to the **Maximum number of corners** parameter	
Count	Scalar value that represents the number of detected corners	32-bit unsigned integer
Metric	Matrix of corner metric values that is the same size as the input image	Same as I port

Minimum Eigenvalue Method

This method is more computationally expensive than the Harris corner detection algorithm because it directly calculates the eigenvalues of the sum of the squared difference matrix, *M*.

The sum of the squared difference matrix, *M*, is defined as follows:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

The previous equation is based on the following values:

$$A = (I_x)^2 \otimes w$$
$$B = (I_y)^2 \otimes w$$
$$C = (I_x I_y)^2 \otimes w$$

where I_x and I_y are the gradients of the input image, *I*, in the *x* and *y* direction, respectively. The $\otimes$ symbol denotes a convolution operation.

Corner Detection

Use the **Coefficients for separable smoothing filter** parameter to define a vector of filter coefficients. The block multiplies this vector of coefficients by its transpose to create a matrix of filter coefficients, *w*.

The block calculates the smaller eigenvalue of the sum of the squared difference matrix. This minimum eigenvalue corresponds to the corner metric matrix.

Harris Corner Detection Method

The Harris corner detection method avoids the explicit computation of the eigenvalues of the sum of squared differences matrix by solving for the following corner metric matrix, *R*:

$$R = AB - C^2 - k(A + B)^2$$

A, *B*, *C* are defined in the previous section, "Minimum Eigenvalue Method" on page 1-291.

The variable *k* corresponds to the sensitivity factor. You can specify its value using the **Sensitivity factor (0<k<0.25)** parameter. The smaller the value of *k*, the more likely it is that the algorithm can detect sharp corners.

Use the **Coefficients for separable smoothing filter** parameter to define a vector of filter coefficients. The block multiplies this vector of coefficients by its transpose to create a matrix of filter coefficients, *w*.

Local Intensity Comparison

This method determines that a pixel is a possible corner if it has either, *N* contiguous valid bright surrounding pixels, or *N* contiguous dark surrounding pixels. Specifying the value of *N* is discussed later in this section. The next section explains how the block finds these surrounding pixels.

Suppose that *p* is the pixel under consideration and *j* is one of the pixels surrounding *p*. The locations of the other surrounding pixels are denoted by the shaded areas in the following figure.
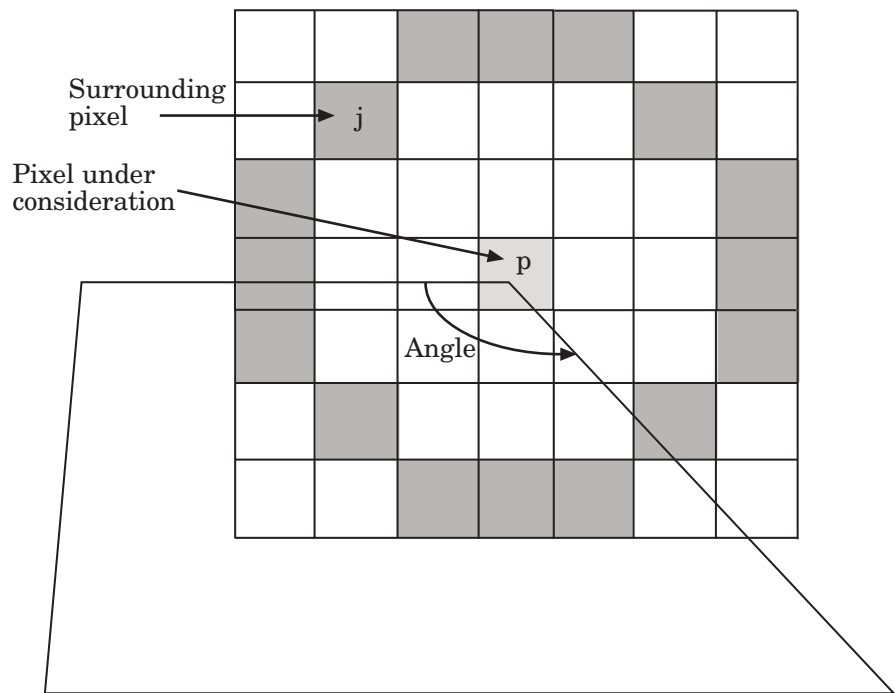
I_p and I_j are the intensities of pixels p and j, respectively. Pixel j is a valid bright surrounding pixel if $I_j - I_p \geq T$. Similarly, pixel j is a valid dark surrounding pixel if $I_p - I_j \geq T$. In these equations, T is the value you specified for the **Intensity comparison threshold** parameter.

The block repeats this process to determine whether the block has N contiguous valid surrounding pixels. The value of N is related to the value you specify for the **Maximum angle to be considered a corner (in degrees)**, as shown in the following table.

Corner Detection

Number of Valid Surrounding Pixels, N	Angle (degrees)
15	22.5
14	45
13	67.5
12	90
11	112.5
10	135
9	157.5

After the block determines that a pixel is a possible corner, it computes its corner metric using the following equation:

$$R = \max\left(\sum_{j:I_j \geq I_p + T} \left| I_p - I_j \right| - T, \sum_{j:I_j \leq I_p - T} \left| I_p - I_j \right| - T, \right)$$

Fixed-Point Data Types

The following diagram shows the data types used in the Corner Detection block for fixed-point signals. These diagrams apply to the Harris corner detection and minimum eigenvalue methods only.
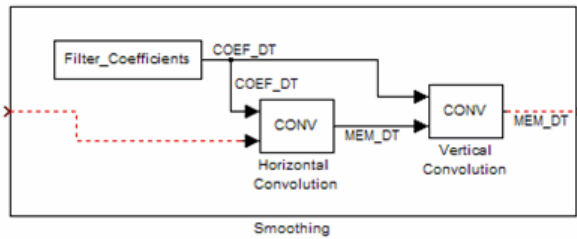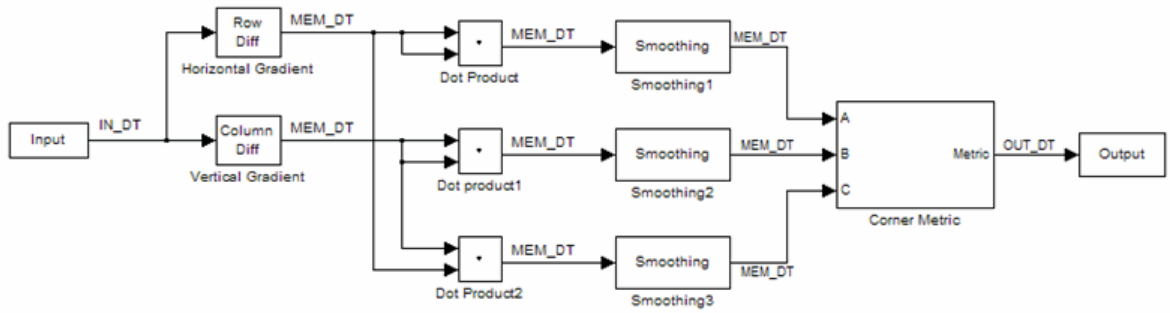
Corner Detection
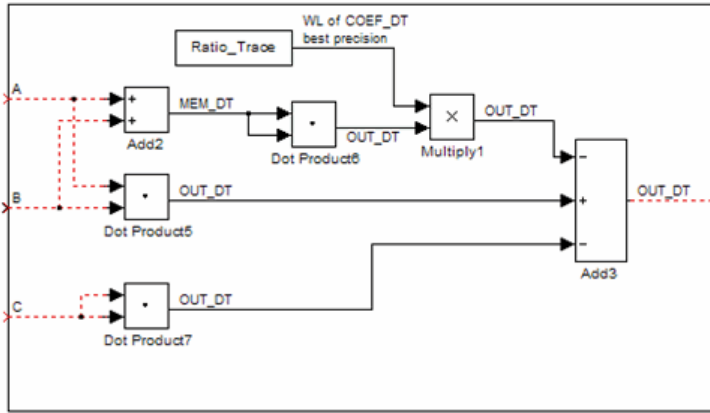
Corner Metric by Harris Algorithm
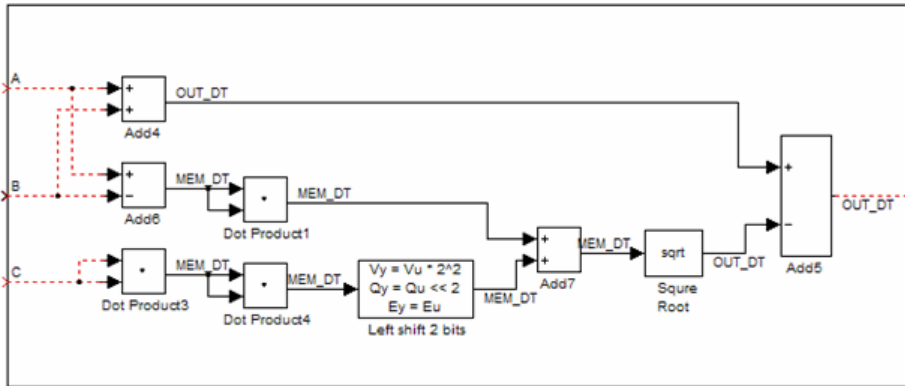
Corner Metric by Minimum Eigenvalue Algorithm

The following table summarizes the variables used in the previous diagrams.

Variable Name	Definition
IN_DT	Input data type
MEM_DT	Memory data type
OUT_DT	Metric output data type
COEF_DT	Coefficients data type

Dialog Box

The Corner Detection dialog box appears as shown in the following figure.

Function Block Parameters: Corner Detection

Corner Detection

Find corners in the image.

You can use this block to find the location of the corners, and the corner metric values returned as M-by-2 matrix of one-based [x y] coordinates, where M is the number of c

| Main | Data Types |

Parameters

Method: Harris corner detection (Harris & Stephens)

Sensitivity factor (0<k<0.25): 0.04

Coefficients for separable smoothing filter: fspecial('gaussian', [1 5], 1.5)

Output: Corner location

Maximum number of corners: 200

Minimum metric value that indicates a corner: 0.0005

Neighborhood size (suppress region around detected corners): [11 11]

☑ Output variable size signal

OK   Cancel

Corner Detection

Method

> Specify the method to use to find the corner values. Your
> choices are `Harris corner detection (Harris & Stephens)`,
> `Minimum eigenvalue (Shi & Tomasi)`, and `Local intensity`
> `comparison (Rosten & Drummond)`.

Sensitivity factor (0<k<0.25)

> Specify the sensitivity factor, k. The smaller the value of k
> the more likely the algorithm is to detect sharp corners. This
> parameter is visible if you set the **Method** parameter to `Harris`
> `corner detection (Harris & Stephens)`. This parameter is
> tunable.

Coefficients for separable smoothing filter

> Specify a vector of filter coefficients for the smoothing filter.
> This parameter is visible if you set the **Method** parameter to
> `Harris corner detection (Harris & Stephens)` or `Minimum`
> `eigenvalue (Shi & Tomasi)`.

Intensity comparison threshold

> Specify the threshold value used to find valid surrounding pixels.
> This parameter is visible if you set the **Method** parameter to
> `Local intensity comparison (Rosten & Drummond)`. This
> parameter is tunable.

Maximum angle to be considered a corner (in degrees)

> Specify the maximum corner angle. This parameter is visible if
> you set the **Method** parameter to `Local intensity comparison`
> `(Rosten & Drummond)`. This parameter is tunable for Simulation
> only.

Output

> Specify the block output. Your choices are `Corner location`,
> `Corner location and metric matrix`, and `Metric matrix`.
> The block outputs the corner locations in an M-by-2 matrix of [x
> y] coordinates, where M represents the number of corners. The
> block outputs the corner metric value in a matrix, the same size
> as the input image.

When you set the this parameter to `Corner location` or `Corner location and metric matrix`, the **Maximum number of corners**, **Minimum metric value that indicates a corner**, and **Neighborhood size (suppress region around detected corners)** parameters appear on the block.

To determine the final corner values, the block follows this process:

1 Find the pixel with the largest corner metric value.

2 Verify that the metric value is greater than or equal to the value you specified for the **Minimum metric value that indicates a corner** parameter.

3 Suppress the region around the corner value by the size defined in the **Neighborhood size (suppress region around detected corners)** parameter.

The block repeats this process until it finds all the corners in the image or it finds the number of corners you specified in the **Maximum number of corners** parameter.

The corner metric values computed by the `Minimum eigenvalue` and `Local intensity comparison` methods are always non-negative. The corner metric values computed by the `Harris corner detection` method can be negative.

Maximum number of corners
Enter the maximum number of corners you want the block to find. This parameter is visible if you set the **Output** parameter to `Corner location` or `Corner location and metric matrix`.

Minimum metric value that indicates a corner
Specify the minimum corner metric value. This parameter is visible if you set the **Output** parameter to `Corner location` or `Corner location and metric matrix`. This parameter is tunable.

Neighborhood size (suppress region around detected corners)
Specify the size of the neighborhood around the corner metric value over which the block zeros out the values. Enter a two-element vector of positive odd integers, [r c]. Here, r is the number of rows in the neighborhood and c is the number of columns. This parameter is visible if you set the **Output** parameter to `Corner location` or `Corner location and metric matrix`.

The **Data Types** pane of the Corner Detection dialog box appears as shown in the following figure.

Function Block Parameters: Corner Detection

Corner Detection

Find corners in the image.

You can use this block to find the location of the corners, and the corner metric values
returned as M-by-2 matrix of one-based [x y] coordinates, where M is the number of c

| Main | Data Types |

Floating-point inheritance takes precedence over the settings in the 'Data Type' column
block input is floating point, all block data types match the input.

Fixed-point operational parameters

Rounding mode: Floor ▼   Overflow mode: Wrap

Fixed-point data types

Open Data Type Diagram

	Data Type	Signed	Word length	Fra
Coefficients	Specify word length ▼	Yes	16	16
Product output	Binary point scaling ▼	Yes	32	0
Accumulator	Binary point scaling ▼	Yes	32	0
Memory	Binary point scaling ▼	Yes	32	0
Metric output	Same as accumulator ▼			

☐ Lock data type settings against changes by the fixed-point tools

OK   Cancel

Rounding mode

    Select the rounding mode for fixed-point operations.

Overflow mode

    Select the overflow mode for fixed-point operations.

Coefficients

    Choose how to specify the word length and the fraction length of the coefficients:

- When you select `Same word length as input`, the word length of the coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. The block automatically sets the fraction length to give you the best precision.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. The bias of all signals in the Computer Vision System Toolbox software is 0.

Product output

    As shown in the following figure, the output of the multiplier is placed into the product output data type and scaling.

    Use this parameter to specify how to designate the product output word and fraction lengths.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox software is 0.

Accumulator

As shown in the following figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it.

The result of each addition remains in the accumulator data type.

Input to adder - input data type → CAST → Accumulator data type → ADDER → Accumulator data type

Use this parameter to specify how to designate this accumulator word and fraction lengths:

- When you select `Same as input`, these characteristics match those of the input.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox software is 0.

Memory

Choose how to specify the memory word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of 0.

Metric output

Choose how to specify the metric output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References

[1] C. Harris and M. Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference.* August 1988, pp. 147-151.

[2] J. Shi and C. Tomasi. "Good Features to Track." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* June 1994, pp. 593–600.

[3] E. Rosten and T. Drummond. "Fusing Points and Lines for High Performance Tracking." *Proceedings of the IEEE International Conference on Computer Vision* Vol. 2 (October 2005): pp. 1508–1511.

See Also

Find Local Maxima	Computer Vision System Toolbox software
Estimate Geometric Transformation	Computer Vision System Toolbox software
matchFeatures	Computer Vision System Toolbox software
extractFeatures	Computer Vision System Toolbox software
detectSURFFeatures	Computer Vision System Toolbox software

Corner Detection (To Be Removed)

Purpose   Calculate corner metric matrix and find corners in images

Library   Analysis & Enhancement

                     vipanalysis

Description
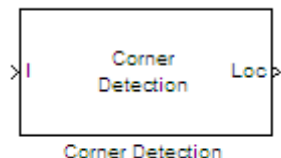
Note This Corner Detection block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Corner Detection block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Corner Detection documentation to view the reference material related to the obsolete block.

Purpose   Remove motion artifacts by deinterlacing input video signal

Library   Analysis & Enhancement

                  visionanalysis

Description   The Deinterlacing block takes the input signal, which is the combination
                  of the top and bottom fields of the interlaced video, and converts it into
                  deinterlaced video using line repetition, linear interpolation, or vertical
                  temporal median filtering.

> **Note** This block supports intensity and color images on its ports.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Combination of top and bottom fields of interlaced video	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Output	Frames of deinterlaced video	Same as Input port	No

Use the **Deinterlacing method** parameter to specify how the block
deinterlaces the video.

The following figure illustrates the block's behavior if you select Line
repetition.

Deinterlacing

Line Repetition

Original Interlaced Video

	Top Field				Bottom Field		
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

	Block Input				Block Output - Deinterlaced Video		
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	A	B	C
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	G	H	I
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

The following figure illustrates the block's behavior if you select Linear interpolation.

Linear Interpolation

Original Interlaced Video

	Top Field				Bottom Field		
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

	Block Input				Block Output - Deinterlaced Video		
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	(A+G)/2	(B+H)/2	(C+I)/2
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	(G+M)/2	(H+N)/2	(I+O)/2
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

The following figure illustrates the block's behavior if you select
`Vertical temporal median filtering`.

Deinterlacing

Vertical Temporal Median Filtering

Original Interlaced Video

	Top Field				Bottom Field		
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

	Block Input				Block Output - Deinterlaced Video		
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	median([A,D,G])	median([B,E,H])	median([C,F,I])
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	median([G,J,M])	median([H,K,N])	median([I,L,O])
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

Row-Major Data Format

The MATLAB enviroment and the Computer Vision System Toolbox software use column-major data organization. However, the Deinterlacing block gives you the option to process data that is stored in

row-major format. When you select the **Input image is transposed (data order is row major)** check box, the block assumes that the input buffer contains contiguous data elements from the first row first, then data elements from the second row second, and so on through the last row. Use this functionality only when you meet all the following criteria:

- You are developing algorithms to run on an embedded target that uses the row-major format.

- You want to limit the additional processing required to take the transpose of signals at the interfaces of the row-major and column-major systems.

When you use the row-major functionality, you must consider the following issues:

- When you select this check box, the first two signal dimensions of the Deinterlacing block's input are swapped.

- All the Computer Vision System Toolbox blocks can be used to process data that is in the row-major format, but you need to know the image dimensions when you develop your algorithms.

  For example, if you use the 2-D FIR Filter block, you need to verify that your filter coefficients are transposed. If you are using the Rotate block, you need to use negative rotation angles, etc.

- Only three blocks have the **Input image is transposed (data order is row major)** check box. They are the Chroma Resampling, Deinterlacing, and Insert Text blocks. You need to select this check box to enable row-major functionality in these blocks. All other blocks must be properly configured to process data in row-major format.

Use the following two-step workflow to develop algorithms in row-major format to run on an embedded target.

Deinterlacing

Step 1:
Create block diagram

Algorithm blocks

Video source block

Transpose block

Transpose block

Video sink block

Embedded target source block

Step 2:
Replace source, transpose, and sink blocks with target source and sink blocks that produce data in row-major format

Embedded target sink block
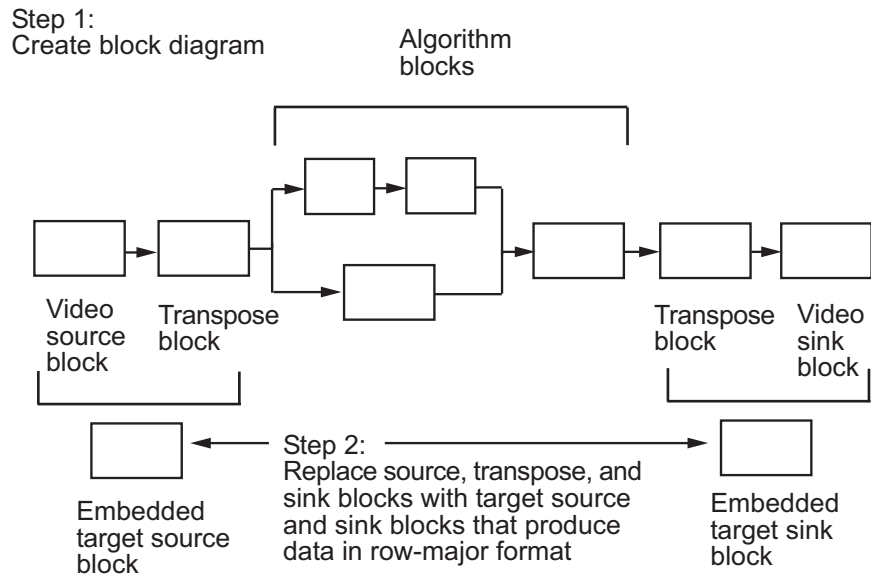
See the DM642 EVM Video ADC and DM642 EVM Video DAC reference pages.

Example

The following example shows you how to use the Deinterlacing block to remove motion artifacts from an image.

1 Open the example model by typing

```
ex_deinterlace
```

at the MATLAB command prompt.

2 Double-click the Deinterlacing block. The model uses this block to remove the motion artifacts from the input image. The **Deinterlacing method** parameter is set to `Vertical temporal median filtering`.
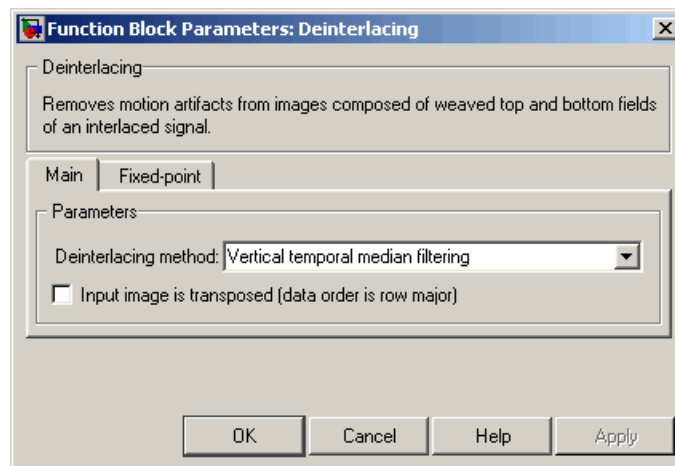
3 Run the model.

The original image that contains the motion artifacts appears in the Input Image window.
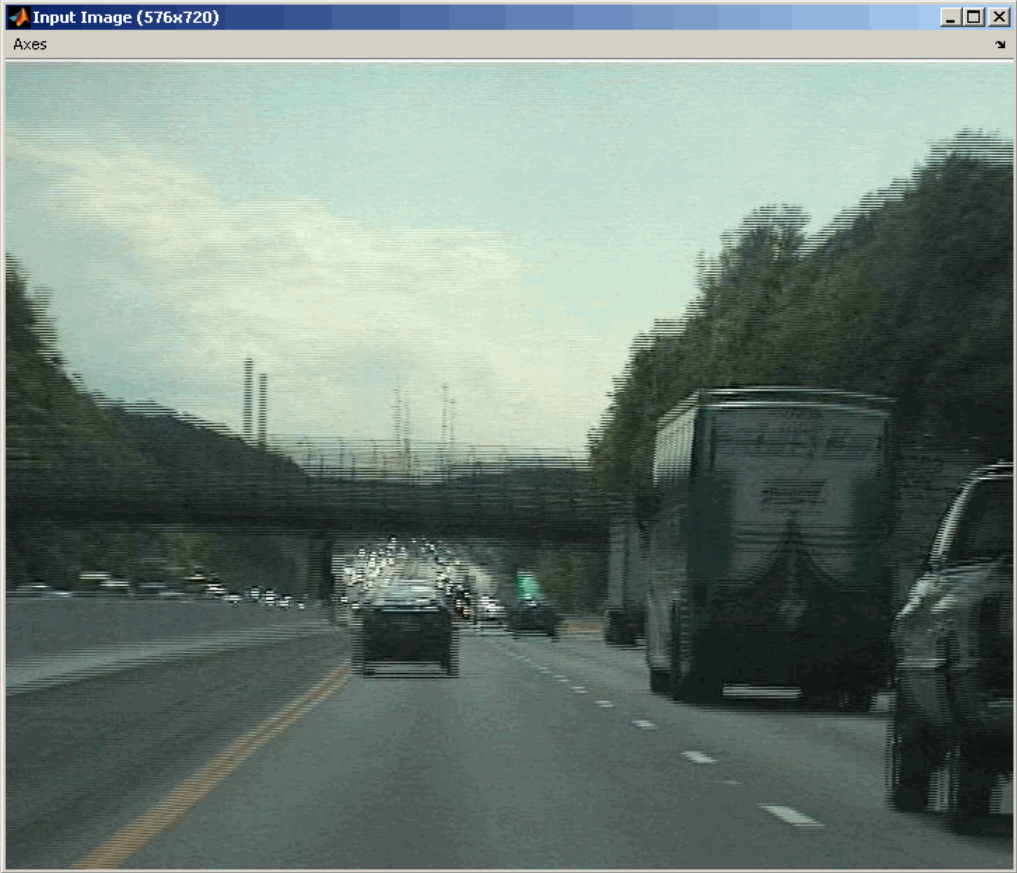
The clearer output image appears in the Output Image window.

Fixed-Point Data Types

The following diagram shows the data types used in the Deinterlacing block for fixed-point signals.

Deinterlacing
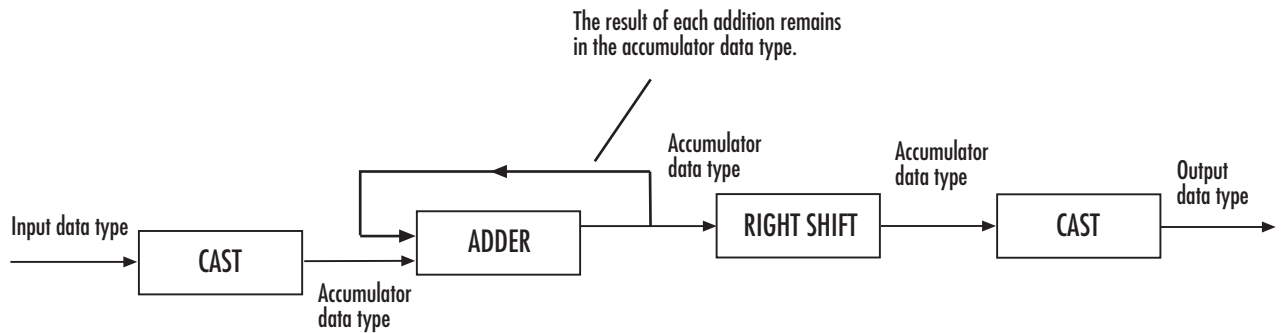
The result of each addition remains
in the accumulator data type.

Input data type → CAST → (Accumulator data type) → ADDER → (Accumulator data type) → RIGHT SHIFT → (Accumulator data type) → CAST → Output data type

You can set the product output, accumulator, and output data types in the block mask as discussed in the next section.

Dialog Box

The **Main** pane of the Deinterlacing dialog box appears as shown in the following figure.

Function Block Parameters: Deinterlacing

Deinterlacing

Removes motion artifacts from images composed of weaved top and bottom fields of an interlaced signal.

Main | Data Types

Parameters

Deinterlacing method: Line repetition

☐ Input image is transposed (data order is row major)

OK | Cancel | Help | Apply
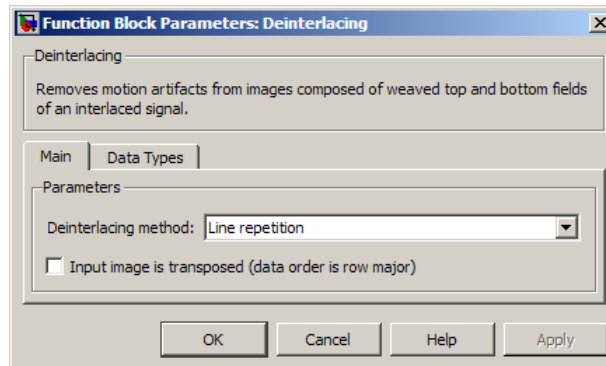
Deinterlacing method

Specify how the block deinterlaces the video. Your choices are Line repetition, Linear interpolation, or Vertical temporal median filtering.

Input image is transposed (data order is row major)

When you select this check box, the block assumes that the input buffer contains data elements from the first row first, then data

elements from the second row second, and so on through the last row.

The **Data Types** pane of the Deinterlacing dialog box appears as shown in the following figure.
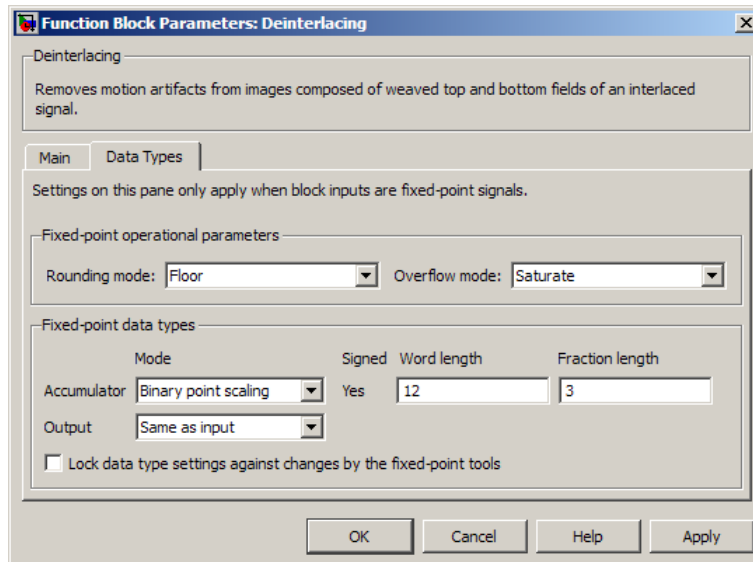
Note The parameters on the **Data Types** pane are only available if, for the **Deinterlacing method**, you select Linear interpolation.

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Accumulator

The result of each addition remains in the accumulator data type.

Input to adder - input data type

CAST

Accumulator data type

ADDER

Accumulator data type

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths:

- When you select Same as product output, these characteristics match those of the product output.

- When you select Same as input, these characteristics match those of the input.

- When you select Binary point scaling, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output

Choose how to specify the output word length and fraction length:

- When you select Same as input, these characteristics match those of the input to the block.

- When you select Binary point scaling, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of 0.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Demosaic

Purpose   Demosaic Bayer's format images

Library   Conversions

visionconversions

Description
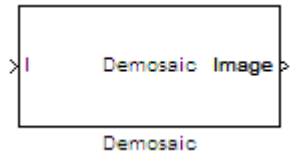
The following figure illustrates a 4-by-4 image in Bayer's format with each pixel labeled R, G, or B.

B	G	B	G
G	R	G	R
B	G	B	G
G	R	G	R

The Demosaic block takes in images in Bayer's format and outputs RGB images. The block performs this operation using a gradient-corrected linear interpolation algorithm or a bilinear interpolation algorithm.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Matrix of intensity values • If, for the **Interpolation algorithm** parameter, you select Bilinear, the number of rows and columns must be greater than or equal to 3. • If, for the **Interpolation algorithm** parameter, you select Gradient-corrected linear, the number of rows and columns must be greater than or equal to 5.	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
R, G, B	Matrix that represents one plane of the input RGB video stream. Outputs from the R, G, or B ports have the same data type.	Same as I port	No
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes.	Same as I port	No

Use the **Interpolation algorithm** parameter to specify the algorithm the block uses to calculate the missing color information. If you select Bilinear, the block spatially averages neighboring pixels to calculate the color information. If you select Gradient-corrected linear, the block uses a Weiner approach to minimize the mean-squared error in

Demosaic

the interpolation. This method performs well on the edges of objects in the image. For more information, see [1].

Use the **Sensor alignment** parameter to specify the alignment of the input image. Select the sequence of R, G and B pixels that correspond to the 2-by-2 block of pixels in the top-left corner of the image. You specify the sequence in left-to-right, top-to-bottom order. For example, for the image at the beginning of this reference page, you would select BGGR.

Both methods use symmetric padding at the image boundaries. For more information, see the Image Pad block reference page.

Use the **Output image signal** parameter to specify how to output a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Fixed-Point Data Types

The following diagram shows the data types used in the Demosaic block for fixed-point signals.
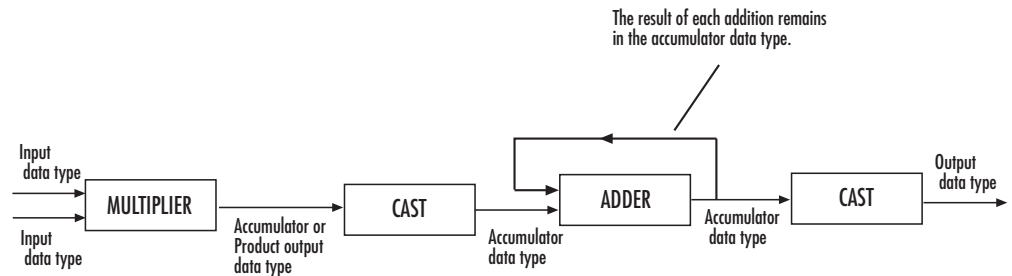
You can set the product output and accumulator data types in the block mask as discussed in the next section.

Dialog Box

The **Main** pane of the Demosaic dialog box appears as shown in the following figure.
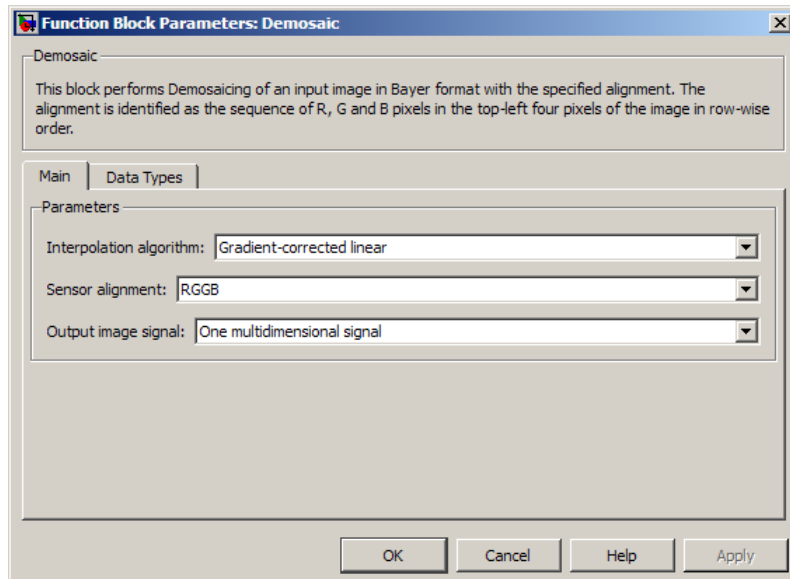
Interpolation algorithm

Specify the algorithm the block uses to calculate the missing color information. Your choices are Bilinear or Gradient-corrected linear.

Sensor alignment

Select the sequence of R, G and B pixels that correspond to the 2-by-2 block of pixels in the top left corner of the image. You specify the sequence in left-to-right, top-to-bottom order.

Output image signal

Specify how to output a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports

appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

The **Data Types** pane of the Demosaic dialog box appears as shown in the following figure.

Rounding mode
    Select the rounding mode for fixed-point operations.

Overflow mode
    Select the overflow mode for fixed-point operations.

Product output

Accumulator
data type

MULTIPLIER

sfix8_En7

Product output
data type

As depicted in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths:

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

The result of each addition remains in the accumulator data type.

CAST

ADDER

Input to adder -
input data type

Accumulator
data type

Accumulator
data type

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input

is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths:

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the input.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References [1] Malvar, Henrique S., Li-wei He, and Ross Cutler, "High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images," *Microsoft Research*, One Microsoft Way, Redmond, WA 98052

[2] Gunturk, Bahadir K., John Glotzbach, Yucel Altunbasak, Ronald W. Schafer, and Russel M. Mersereau, "Demosaicking: Color Filter Array Interpolation," *IEEE Signal Processing Magazine*, Vol. 22, Number 1, January 2005.

Purpose   Find local maxima in binary or intensity image

Library   Morphological Operations

visionmorphops

Description   The Dilation block rotates the neighborhood or structuring element 180 degrees. Then it slides the neighborhood or structuring element over an image, finds the local maxima, and creates the output matrix from these maximum values. If the neighborhood or structuring element has a center element, the block places the maxima there, as illustrated in the following figure.
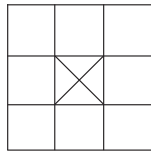
If the neighborhood or structuring element does not have an exact center, the block has a bias toward the lower-right corner, as a result of the rotation. The block places the maxima there, as illustrated in the following figure.
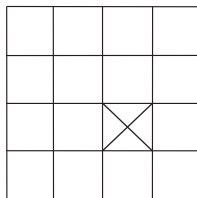
This block uses flat structuring elements only.

Dilation

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of ones and zeros that represents the neighborhood values	Boolean	No
Output	Vector or matrix of intensity values that represents the dilated image	Same as I port	No

The output signal has the same data type as the input to the I port.

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. You can only specify a structuring element using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the neighborhood or structuring element that the block applies to the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the

use of a more efficient algorithm. If you enter an array of STREL objects, the block applies each object to the entire matrix in turn.

Dialog Box

The Dilation dialog box appears as shown in the following figure.

Neighborhood or structuring element source

Specify how to enter your neighborhood or structuring element values. Select Specify via dialog to enter the values in the dialog box. Select Input port to use the Nhood port to specify the neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the strel function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or structuring element source** parameter, you select Specify via dialog.

Dilation

References   [1] Soille, Pierre. *Morphological Image Analysis. 2nd ed.* New York: Springer, 2003.

[1] Soille, Pierre. *Morphological Image Analysis. 2nd ed.* New York: Springer, 2003.

See Also

Bottom-hat	Computer Vision System Toolbox software
Closing	Computer Vision System Toolbox software
Erosion	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
imdilate	Image Processing Toolbox software
strel	Image Processing Toolbox software

Purpose   Draw markers by embedding predefined shapes on output image

Library   Text & Graphics

visiontextngfix

Description   The Draw Markers block can draw multiple circles, x-marks, plus signs, stars, or squares on images by overwriting pixel values. Overwriting the pixel values embeds the shapes.
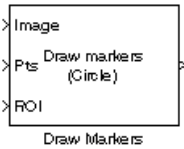
This block uses Bresenham's circle drawing algorithm to draw circles and Bresenham's line drawing algorithm to draw all other markers.

Port Description

Port	Input/Output	Supported Data Types	Complex Values Supported
Image	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color values where *P* is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
R, G, B	Scalar, vector, or matrix that represents one plane of the input RGB video stream. Inputs to the R, G, and B ports must have the same dimensions and data type.	Same as Image port	No

Draw Markers

Port	Input/Output	Supported Data Types	Complex Values Supported
Pts	*M*-by-2 matrix of [x y] coordinates, $$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_M & y_M \end{bmatrix}$$ where *M* is the total number of markers and each [x y] pair defines the center of a marker.	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer If the input to the Image port is an integer, fixed point, or boolean data type, the input to the Pts port must also be an integer data type.	No
ROI	Four-element vector of integers [x y width height] that define a rectangular area in which to draw the markers. The first two elements represent the one-based [x y] coordinates of the upper-left corner of the area. The second two elements represent the width and height of the area.	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Clr	*P*-element vector or *M*-by-*P* matrix where *P* is the number of color planes.	Same as Image port	No
Output	Scalar, vector, or matrix of pixel values that contain the marker(s)	Same as Image port	No

The output signal is the same size and data type as the inputs to the Image, R, G, and B ports.

Draw Markers

Dialog Box

The Draw Markers dialog box appears as shown in the following figure.

Function Block Parameters: Draw Markers

Draw Markers

Mark locations on images by drawing circles, x-marks, plus signs, stars, or squares.

Use the Pts port to specify an M-by-2 matrix of x and y pairs, where M is the total number of markers and each x and y pair defines a marker's center. Use the Marker size parameter to define the size of the marker, in pixels. Use the Draw markers in parameter to define the area in which to draw the markers.

Main	Data Types

Parameters

Marker shape: Circle

Marker size: 3

☐ Filled

Border color source: Specify via dialog

Border color: Black

Draw Markers in: Entire image

☐ Use antialiasing

Image signal: One multidimensional signal

OK   Cancel   Help   Apply

Marker shape

Specify the type of marker(s) to draw. Your choices are Circle, X-mark, Plus, Star, or Square.

When you select Circle, X-mark, or Star, and you select the **Use antialiasing** check box, the block performs a smoothing algorithm. The algorithm is similar to the poly2mask function to determine which subpixels to draw.

Marker size

Enter a scalar value that represents the size of the marker, in pixels.

Enter a scalar value, M, that defines a $(2M+1)$-by-$(2M+1)$ pixel square into which the marker fits. M must be greater than or equal to 1.

Filled

Select this check box to fill the marker with an intensity value or a color. This parameter is visible if, for the **Marker shape** parameter, you choose Circle or Square.

When you select the **Filled** check box, the **Fill color source**, **Fill color** and **Opacity factor (between 0 and 1)** parameters appear in the dialog box.

Fill color source

Specify source for fill color value. You can select Specify via dialog or Input port. This parameter appears when you select the **Filled** check box. When you select Input port, the color input port **clr** appears on the block.

Fill color

If you select Black, the marker is black. If you select White, the marker is white. If you select User-specified value, the **Color value(s)** parameter appears in the dialog box. This parameter is visible if you select the **Filled** check box.

Draw Markers

Border color source

Specify source for the border color value to either `Specify via dialog` or `Input port`. Border color options are visible when the fill shapes options are not selected. This parameter is visible if you select the **Filled** check box. When you select `Input port`, the color input port **clr** appears on the block.

Border color

Specify the appearance of the shape's border. If you select `Black`, the border is black. If you select `White`, the border is white. If you select `User-specified value`, the **Color value(s)** parameter appears in the dialog box. This parameter is visible if you clear the **Fill shapes** check box.

Color value(s)

Specify an intensity or color value for the marker's border or fill. This parameter appears when you set the **Border color** or **Fill color** parameters, to `User-specified value`. Tunable.

The following table describes what to enter for the color value based on the block input and the number of shapes you are drawing.

Block Input	Color Value(s) for Drawing One Marker or Multiple Markers with the Same Color	Color Value(s) for Drawing Multiple Markers with Unique Color
Intensity image	Scalar intensity value	R-element vector where R is the number of markers
Color image	P-element vector where P is the number of color planes	P-by-R matrix where P is the number of color planes and R is the number of markers

For each value in the parameter, enter a number between the minimum and maximum values that can be represented by the data type of the input image. If you enter a value outside this range, the block produces an error message.

Opacity factor (between 0 and 1)

Specify the opacity of the shading inside the marker, where 0 indicates transparent and 1 indicates opaque. This parameter appears when you select the **Filled** check box. This parameter is tunable.

The following table describes what to enter for the **Opacity factor(s) (between 0 and 1)** parameter based on the block input and the number of markers you are drawing.

Opacity Factor value for Drawing One Marker or Multiple Markers with the Same Color	Oopacity Factor value for Drawing Multiple Marker with Unique Color
Scalar intensity value	R-element vector where R is the number of markers

Draw markers in

Specify the area in which to draw the markers. When you select Entire image, you can draw markers in the entire image. When you select Specify region of interest via port, the ROI port appears on the block. Enter a four-element vector, [x y width height], where [x y] are the coordinates of the upper-left corner of the area.

Use antialiasing

Perform a smoothing algorithm on the marker. This parameter is visible if, for the **Marker shape** parameter, you select Circle, X-mark, or Star.

Image signal

Specify how to input and output a color video signal. When you select One multidimensional signal, the block accepts an

Draw Markers

M-by-N-by-P color video signal, where P is the number of color planes, at one port. When you select `Separate color signals`, additional ports appear on the block. Each port accepts one M-by-N plane of an RGB video stream.

See Also

|	
Draw Shapes	Computer Vision System Toolbox software
Insert Text	Computer Vision System Toolbox software

Purpose   Draw markers by embedding predefined shapes on output image

Library   Text & Graphics

Description

Note This Draw Markers block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Draw Markers block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Draw Markers documentation to view the reference material related to the obsolete block.

Draw Shapes

Purpose   Draw rectangles, lines, polygons, or circles on images

Library   Text & Graphics

visiontextngfix

Description The Draw Shapes block draws multiple rectangles, lines, polygons, or circles on images by overwriting pixel values. As a result, the shapes are embedded on the output image.

This block uses Bresenham's line drawing algorithm to draw lines, polygons, and rectangles. It uses Bresenham's circle drawing algorithm to draw circles.

The output signal is the same size and data type as the inputs to the Image, R, G, and B ports.

You can set the shape fill or border color via the input port or via the input dialog. Use the color input or color parameter to determine the appearance of the rectangle(s), line(s), polygon(s), or circle(s).

Port Description

Port	Input/Output	Supported Data Types	Complex Values Supported
Image	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color values where *P* is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer	No

Port	Input/Output	Supported Data Types	Complex Values Supported
		• 8-, 16-, and 32-bit unsigned integer	
R, G, B	Scalar, vector, or matrix that is one plane of the input RGB video stream. Inputs to the R, G, and B ports must have the same dimensions and data type.	Same as Image port	No
Pts	Use integer values to define one-based shape coordinates. If you enter noninteger values, the block rounds them to the nearest integer.	• Double-precision floating point (only supported if the input to the I or R, G, and B ports is floating point) • Single-precision floating point (only supported if the input to the I or R, G, and B ports is floating point) • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No

Draw Shapes

Port	Input/Output	Supported Data Types	Complex Values Supported
ROI	*4*-element vector of integers [x y width height], that define a rectangular area in which to draw the shapes. The first two elements represent the one-based coordinates of the upper-left corner of the area. The second two elements represent the width and height of the area.	- Double-precision floating point - Single-precision floating point 8-, 16-, and 32-bit signed integer 8-, 16-, and 32-bit unsigned integer	No
Clr	This port can be used to dynamically specify shape color. *P*-element vector or an *M*-by-*P* matrix, where *M* is the number of shapes, and *P*, the number of color planes. You can specify a color (RGB), for each shape, or specify one color for all shapes.	Same as Image port	No
Output	Scalar, vector, or matrix of pixel values that contain the shape(s)	Same as Image port	No

Drawing Shapes and Lines
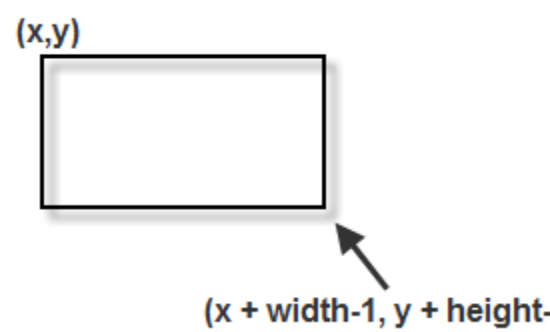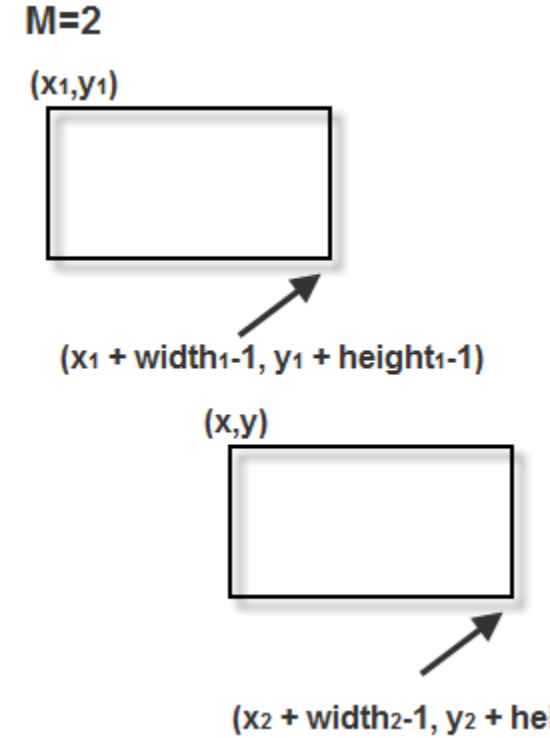
Use the **Shape** parameter and **Pts** port to draw the following shapes or lines:

- Drawing Rectangles on page 343
- Drawing Lines and Polylines on page 345
- Drawing Polygons on page 347
- Drawing Circles on page 349

Drawing Rectangles

The Draw Shapes block lets you draw one or more rectangles. Set the **Shape** parameter to `Rectangles`, and then follow the instructions in the table to specify the input to the Pts port to obtain the desired number of rectangles.
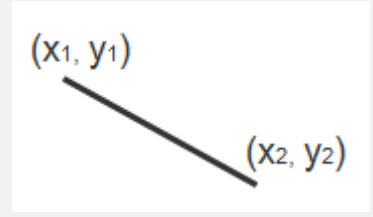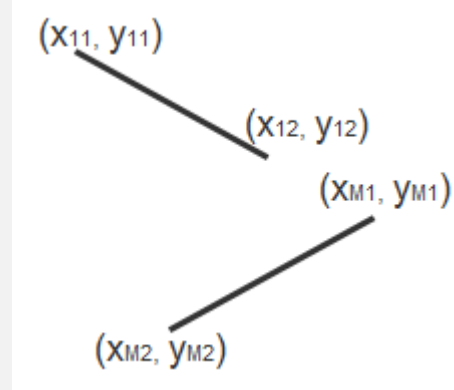
Draw Shapes

Shape	Input to the Pts Port	Drawn Shape
Single Rectangle	Four-element row vector `[x y width height]` where • `x` and `y` are the one-based coordinates of the upper-left corner of the rectangle. • `width` and `height` are the width, in pixels, and height, in pixels, of the rectangle. The values of `width` and `height` must be greater than 0.	
M Rectangles	M-by-4 matrix $$\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$$ where each row of the matrix corresponds to a different rectangle and is of the same form as the vector for a single rectangle.	
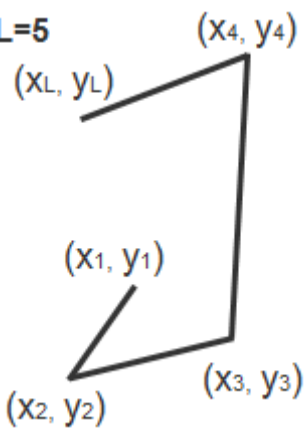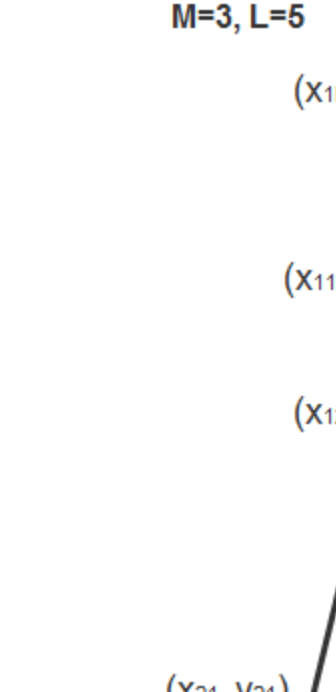
For an example of how to use the Draw Shapes block to draw a rectangle, see "Track an Object Using Correlation".

Drawing Lines and Polylines

The Draw Shapes block lets you draw either a single line, or one or more polylines. You can draw a polyline with a series of connected line segments. Set the **Shape** parameter to Lines, and then follow the instructions in the table to specify the input to the Pts port to obtain the desired shape.

Shape	Input to the Pts Port	Drawn Shape
Single Line	Four-element row vector $[x_1 \ y_1 \ x_2 \ y_2]$ where • x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.	
M Lines	M-by-4 matrix $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$$ where each row of the matrix corresponds to a different line and is of the same form as the vector for a single line.	
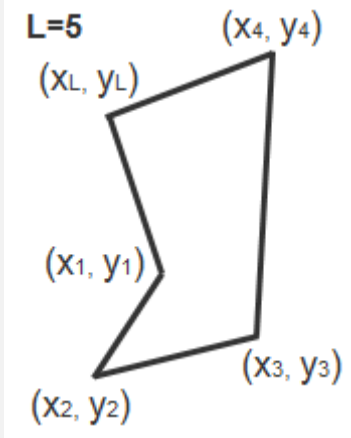
Shape	Input to the Pts Port	Drawn Shape
Single Polyline with (L-1) Segments	Vector of size $2L$, where L is the number of vertices, with format, [x$_1$, y$_1$, x$_2$, y$_2$, ..., x$_L$, y$_L$]. • x$_1$ and y$_1$ are the coordinates of the beginning of the first line segment. • x$_2$ and y$_2$ are the coordinates of the end of the first line segment and the beginning of the second line segment. • x$_L$ and y$_L$ are the coordinates of the end of the (L-1)th line segment. The polyline always contains (L-1) number of segments because the first and last vertex points do not connect. The block produces an error message when the number of rows is less than two or not a multiple of two.	
M Polylines with (L-1) Segments	M-by-$2L$ matrix $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} & y_1 \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} & y_M \end{bmatrix}$$ where each row of the matrix corresponds to a different polyline and is of the same form as the vector for a single polyline. When you require one polyline to contain less than (L–1) number of segments, fill the matrix by repeating the coordinates of the last vertex. The block produces an error message if the number of rows is less than two or not a multiple of two.	

If you select the **Use antialiasing** check box, the block applies an edge smoothing algorithm.

For examples of how to use the Draw Shapes block to draw a line, see "Detect Lines in Images" and "Measure Angle Between Lines".

Drawing Polygons

The Draw Shapes block lets you draw one or more polygons. Set the **Shape** parameter to Polygons, and then follow the instructions in the table to specify the input to the Pts port to obtain the desired number of polygons.

Shape	Input to the Pts Port	Drawn Shape
Single Polygon with L line segments	Row vector of size $2L$, where L is the number of vertices, with format, $[x_1 \; y_1 \; x_2 \; y_2 \; \ldots \; x_L \; y_L]$ where • x_1 and y_1 are the coordinates of the beginning of the first line segment. • x_2 and y_2 are the coordinates of the end of the first line segment and the beginning of the second line segment. • x_L and y_L are the coordinates of the end of the $(L\text{-}1)^{\text{th}}$ line segment and the beginning of the Lth line segment. The block connects $[x_1 \; y_1]$ to $[x_L \; y_L]$ to complete the polygon. The block	
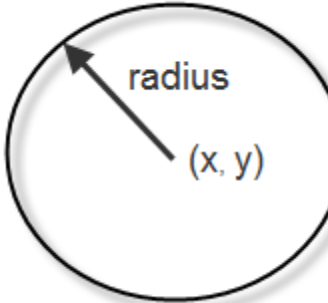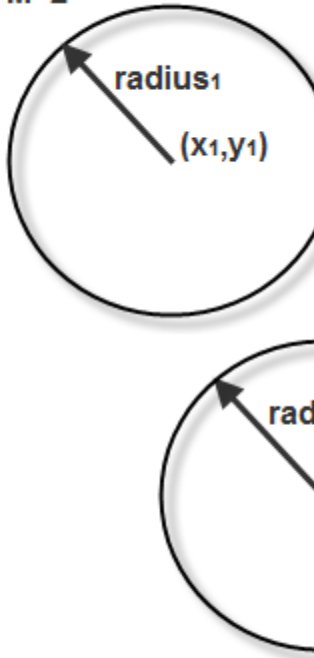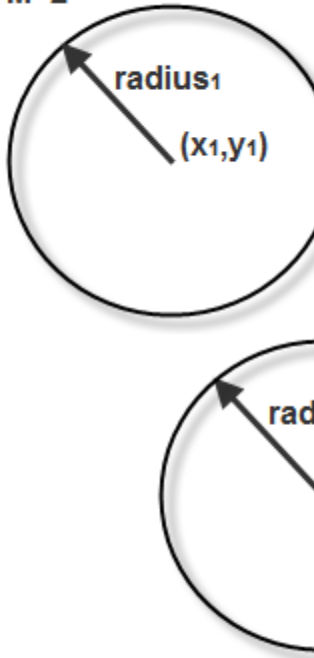
Shape	Input to the Pts Port	Drawn Shape
	produces an error if the number of rows is negative or not a multiple of two.	
M Polygons with the largest number of line segments in any line being *L*	*M*-by-2*L* matrix $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} & y_1 \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} & y_M \end{bmatrix}$$ where each row of the matrix corresponds to a different polygon and is of the same form as the vector for a single polygon. If some polygons are shorter than others, repeat the ending coordinates to fill the polygon matrix. The block produces an error message if the number of rows is less than two or is not a multiple of two.	$M=3, L=5$ (X_{1L}, Y_{1L}) (X_{11}, Y_{11}) (X_{12}, Y_{12}) $(X_{13}$ (X_{24}, Y_{24}) (X_{21}, Y_{21}) (X_{22}, Y_{22}) (X_{ML}, Y_{ML}) $(X_{M2}, Y_{M2}$ (X_{M1}, Y_{M1})

Drawing Circles

The Draw Shapes block lets you draw one or more circles. Set the **Shape** parameter to Circles, and then follow the instructions in the table to specify the input to the Pts port to obtain the desired number of circles.

Draw Shapes

Shape	Input to the Pts Port	Drawn Shape
Single Circle	Three-element row vector `[x y radius]` where • x and y are coordinates for the center of the circle. • radius is the radius of the circle, which must be greater than 0.	
M Circles	*M-by-3* matrix $$\begin{bmatrix} x_1 & y_1 & radius_1 \\ x_2 & y_2 & radius_2 \\ \vdots & \vdots & \vdots \\ x_M & y_M & radius_M \end{bmatrix}$$ where each row of the matrix corresponds to a different circle and is of the same form as the vector for a single circle.	

**Dialog
Box**
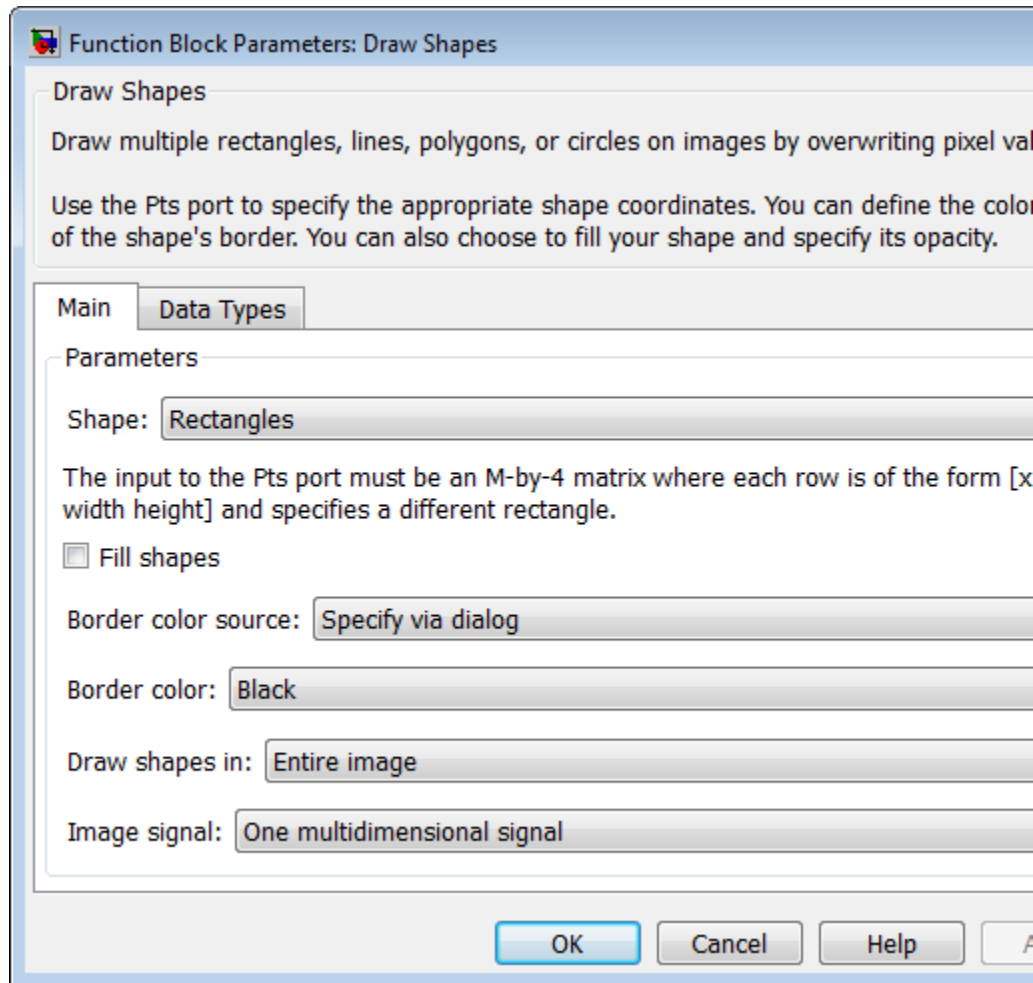
Shape

Specify the type of shape(s) to draw. Your choices are `Rectangles`,
`Lines`, `Polygons`, or `Circles`.

Draw Shapes

The block performs a smoothing algorithm when you select the **Use antialiasing** check box with either `Lines`, `Polygons`, or `Circles`. The block uses an algorithm similar to the `poly2mask` function to determine which subpixels to draw.

Fill shapes

Fill the shape with an intensity value or a color.

When you select this check box, the **Fill color source**, **Fill color** and **Opacity factor (between 0 and 1)** parameters appear in the dialog box.

> **Note** If you are generating code and you select the **Fill shapes** check box, the word length of the block input(s) cannot exceed 16 bits.

When you do not select the **Fill shapes** check box, the **Border color source**, and **Border color** parameters are available.

Fill color source

Specify source for fill color value to either `Specify via dialog` or `Input port`. This parameter appears when you select the **Fill shapes** check box. When you select `Input port`, the color input port **clr** appears on the block.

Fill color

Specify the fill color for shape. You can specify either `Black`, `White`, or `User-specified value`. When you select `User-specified value`, the **Color value(s)** parameter appears in the dialog box. This parameter is visible if you select the **Fill shapes** check box.

Border color source

Specify source for the border color value to either `Specify via dialog` or `Input port`. Border color options are visible when the fill shapes options are not selected. This appears when you select the **Fill shapes** check box. When you select `Input port`, the color input port **clr** appears on the block.

Border color

Specify the appearance of the shape's border. You can specify either `Black`, `White`, or `User-specified value`. If you select `User-specified value`, the **Color value(s)** parameter appears in the dialog box. This parameter appears when you clear the **Fill shapes** check box.

Color value(s)

Specify an intensity or color value for the shape's border or fill. This parameter applies when you set the **Border color** or **Fill color** parameter to `User-specified value`. This parameter is tunable.

The following table describes what to enter for the color value based on the block input and the number of shapes you are drawing.

Block Input	Color Value(s) for Drawing One Shape or Multiple Shapes with the Same Color	Color Value(s) for Drawing Multiple Shapes with Unique Color
Intensity image	Scalar intensity value	R-element vector where R is the number of shapes
Color image	P-element vector where P is the number of color planes	R-by-P matrix where P is the number of color planes and R is the number of shapes

For each value in the **Color Value(s)** parameter, enter a number between the minimum and maximum values that can be represented by the data type of the input image. If you enter a value outside this range, the block produces an error message.

Draw Shapes

Opacity factor (between 0 and 1)

Specify the opacity of the shading inside the shape, where 0 is transparent and 1 is opaque. This parameter is visible if you select the **Fill shapes** check box.

The following table describes what to enter for this parameter based on the block input and the number of shapes you are drawing. This parameter applies when you select the `Filled` check box.

Opacity Factor value for Drawing One Shape or Multiple Shapes with the Same Color	Oopacity Factor value for Drawing Multiple Shapes with Unique Color
Scalar intensity value	R-element vector where R is the number of shapes

Draw shapes in

Specify the type of area in which to draw shapes. You can define one of the following:

- `Entire image`, enables you to draw shapes in the entire image.

- `Specify region of interest via port`. When you select this option, the ROI port appears on the block. Enter a four-element vector of integer values, `[x y width height]`, where `[x y]` are the coordinates of the upper-left corner of the area.

Note If you specify values that are outside the image, the block sets the values to the image boundaries.

Use antialiasing

Perform a smoothing algorithm on the line, polygon, or circle. This parameter is visible if, for the **Shape** parameter, you select Lines, Polygons, or Circles.

Image signal

Specify how to input and output a color video signal. Select one of the following:

- One multidimensional signal, the block accepts an M-by-N-by-P color video signal, where P is the number of color planes, at one port.

- Separate color signals, additional ports appear on the block. Each port accepts one M-by-N plane of an RGB video stream.

See Also

Draw Markers	Computer Vision System Toolbox software
Insert Text	Computer Vision System Toolbox software

Draw Shapes (To Be Removed)

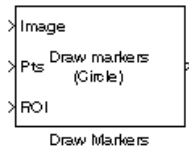**Purpose**   Draw rectangles, lines, polygons, or circles on images

Library   Text & Graphics

Description
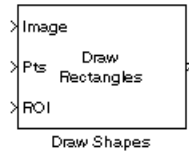
Note This Draw Shapes block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Draw Shapes block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Draw Shapes documentation to view the reference material related to the obsolete block.

Purpose	Find edges of objects in images using Sobel, Prewitt, Roberts, or Canny method
Library	Analysis & Enhancement
	visionanalysis

Description   If, for the **Method** parameter, you select Sobel, Prewitt, or Roberts, the Edge Detection block finds the edges in an input image by approximating the gradient magnitude of the image. The block convolves the input matrix with the Sobel, Prewitt, or Roberts kernel. The block outputs two gradient components of the image, which are the result of this convolution operation. Alternatively, the block can perform a thresholding operation on the gradient magnitudes and output a binary image, which is a matrix of Boolean values. If a pixel value is 1, it is an edge.

If, for the **Method** parameter, you select Canny, the Edge Detection block finds edges by looking for the local maxima of the gradient of the input image. It calculates the gradient using the derivative of the Gaussian filter. The Canny method uses two thresholds to detect strong and weak edges. It includes the weak edges in the output only if they are connected to strong edges. As a result, the method is more robust to noise, and more likely to detect true weak edges.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point (not supported for the Canny method) • 8-, 16-, 32-bit signed integer (not supported for the Canny method)	No

Port	Input/Output	Supported Data Types	Complex Values Supported
		• 8-, 16-, 32-bit unsigned integer (not supported for the Canny method)	
Th	Matrix of intensity values	Same as I port	No
Edge	Matrix that represents a binary image	Boolean	No
Gv	Matrix of gradient values in the vertical direction	Same as I port	No
Gh	Matrix of gradient values in the horizontal direction	Same as I port	No
G45	Matrix of gradient values	Same as I port	No
G135	Matrix of gradient values	Same as I port	No

The output of the Gv, Gh, G45, and G135 ports is the same data type as the input to the I port. The input to the Th port must be the same data type as the input to the I port.

Use the **Method** parameter to specify which algorithm to use to find edges. You can select Sobel, Prewitt, Roberts, or Canny to find edges using the Sobel, Prewitt, Roberts, or Canny method.

Sobel, Prewitt, and Roberts Methods

Use the **Output type** parameter to select the format of the output. If you select Binary image, the block outputs a Boolean matrix at the Edge port. The nonzero elements of this matrix correspond to the edge pixels and the zero elements correspond to the background pixels. If you select Gradient components and, for the **Method** parameter, you

select `Sobel` or `Prewitt`, the block outputs the gradient components that correspond to the horizontal and vertical edge responses at the Gh and Gv ports, respectively. If you select `Gradient components` and, for the **Method** parameter, you select `Roberts`, the block outputs the gradient components that correspond to the 45 and 135 degree edge responses at the G45 and G135 ports, respectively. If you select `Binary image and gradient components`, the block outputs both the binary image and the gradient components of the image.

Select the **User-defined threshold** check box to define a threshold values or values. If you clear this check box, the block computes the threshold for you.

Use the **Threshold source** parameter to specify how to enter your threshold value. If you select `Specify via dialog`, the **Threshold** parameter appears in the dialog box. Enter a threshold value that is within the range of your input data. If you choose `Input port`, use input port Th to specify a threshold value. This value must have the same data type as the input data. Gradient magnitudes above the threshold value correspond to edges.

The Edge Detection block computes the automatic threshold using the mean of the gradient magnitude squared image. However, you can adjust this threshold using the **Threshold scale factor (used to automatically calculate threshold value)** parameter. The block multiplies the value you enter with the automatic threshold value to determine a new threshold value.

Select the **Edge thinning** check box to reduce the thickness of the edges in your output image. This option requires additional processing time and memory resources.

Note This block is most efficient in terms of memory usage and processing time when you clear the **Edge thinning** check box and use the **Threshold** parameter to specify a threshold value.

Edge Detection

Canny Method

Select the **User-defined threshold** check box to define the low and high threshold values. If you clear this check box, the block computes the threshold values for you.

Use the **Threshold source** parameter to specify how to enter your threshold values. If you select `Specify via dialog`, the **Threshold [low high]** parameter appears in the dialog box. Enter the threshold values. If a pixel's magnitude in the gradient image, which is formed by convolving the input image with the derivative of the Gaussian filter, exceeds the high threshold, then the pixel corresponds to a strong edge. Any pixel connected to a strong edge and having a magnitude greater than the low threshold corresponds to a weak edge. If, for the **Threshold source** parameter, you choose `Input port`, use input port Th to specify a two-element vector of threshold values. These values must have the same data type as the input data.

The Edge Detection block computes the automatic threshold values using an approximation of the number of weak and nonedge image pixels. Enter this approximation for the **Approximate percentage of weak edge and nonedge pixels (used to automatically calculate threshold values)** parameter.

Use the **Standard deviation of Gaussian filter** parameter to define the Gaussian filter whose derivative is convolved with the input image.

Fixed-Point Data Types

The following diagram shows the data types used in the Edge Detection block for fixed-point signals.

The result of each addition remains
in the accumulator data type.

MULTIPLIER — Product output data type — CAST — Accumulator data type — ADDER — Accumulator data type — CAST — Output data type

Input data type
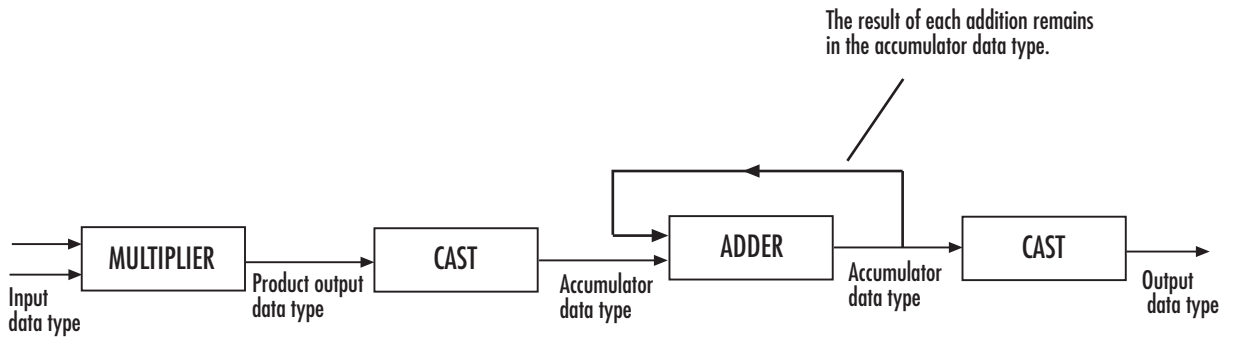
The block squares the threshold and compares it to the sum of the squared gradients to avoid using square roots.

Threshold:

MULTIPLIER — Product output data type — CAST — Accumulator data type

Input data type

Gradients:

MULTIPLIER — Product output data type — CAST — Accumulator data type

Accumulator data type

ADDER — Accumulator data type

MULTIPLIER — Product output data type — CAST — Accumulator data type

Accumulator data type
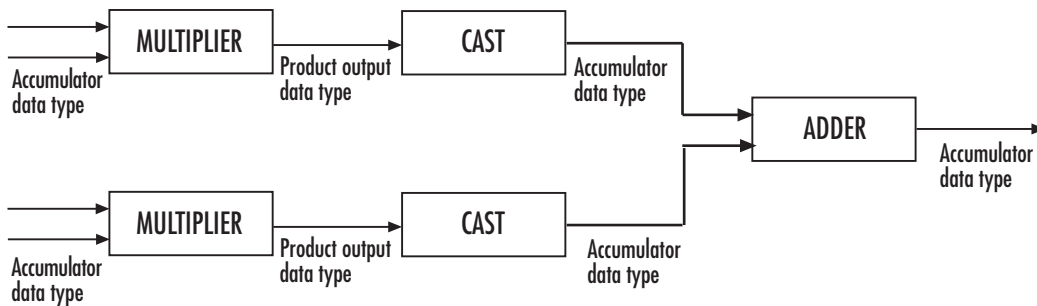
Edge Detection

You can set the product output and accumulator data types in the block mask as discussed in the next section.

Dialog Box

The **Main** pane of the Edge Detection dialog box appears as shown in the following figure.
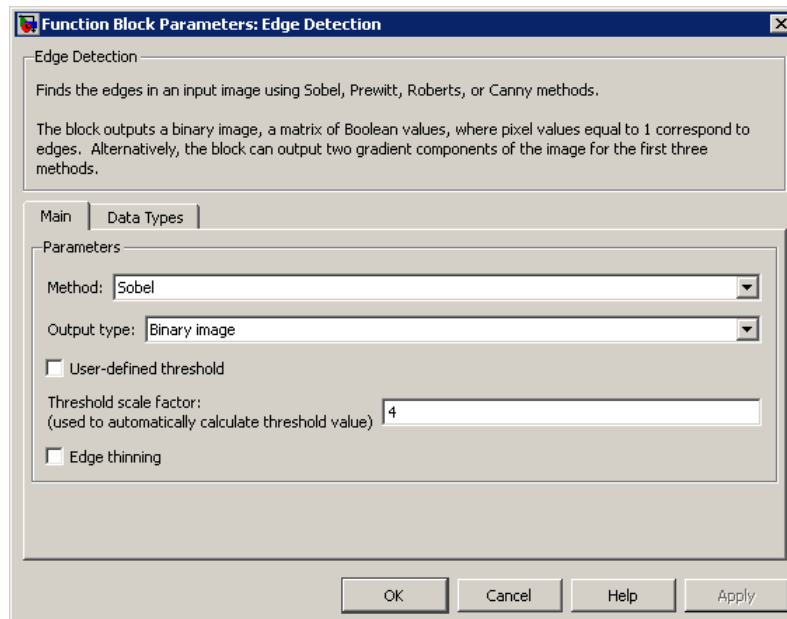
Method

> Select the method by which to perform edge detection. Your choices are Sobel, Prewitt, Roberts, or Canny.

Output type

> Select the desired form of the output. If you select Binary image, the block outputs a matrix that is filled with ones, which correspond to edges, and zeros, which correspond to the background. If you select Gradient components and, for the **Method** parameter, you select Sobel or Prewitt, the block outputs the gradient components that correspond to the horizontal

and vertical edge responses. If you select `Gradient components` and, for the **Method** parameter, you select `Roberts`, the block outputs the gradient components that correspond to the 45 and 135 degree edge responses. If you select `Binary image and gradient components`, the block outputs both the binary image and the gradient components of the image. This parameter is visible if, for the **Method** parameter, you select `Sobel`, `Prewitt`, or `Roberts`.

User-defined threshold

If you select this check box, you can enter a desired threshold value. If you clear this check box, the block computes the threshold for you. This parameter is visible if, for the **Method** parameter, you select `Sobel`, `Prewitt`, or `Roberts`, and, for the **Output type** parameter, you select `Binary image` or `Binary image and gradient components`. This parameter is also visible if, for the **Method** parameter, you select `Canny`.

Threshold source

If you select `Specify via dialog`, enter your threshold value in the dialog box. If you choose `Input port`, use the Th input port to specify a threshold value that is the same data type as the input data. This parameter is visible if you select the **User-defined threshold** check box.

Threshold

Enter a threshold value that is within the range of your input data. This parameter is visible if, for the **Method** parameter, you select `Sobel`, `Prewitt`, or `Roberts`, you select the **User-defined threshold** check box, and, for **Threshold source** parameter, you select `Specify via dialog`. .

Threshold [low high]

Enter the low and high threshold values that define the weak and strong edges. This parameter is visible if, for the **Method** parameter, you select `Canny`. Then you select the **User-defined threshold** check box, and, for **Threshold source** parameter, you select `Specify via dialog`. Tunable.

Edge Detection

Threshold scale factor (used to automatically calculate threshold value)

> Enter a multiplier that is used to adjust the calculation of the automatic threshold. This parameter is visible if, for the **Method** parameter, you select `Sobel`, `Prewitt`, or `Roberts`, and you clear the **User-defined threshold** check box. Tunable.

Edge thinning

> Select this check box if you want the block to perform edge thinning. This option requires additional processing time and memory resources. This parameter is visible if, for the **Method** parameter, you select `Sobel`, `Prewitt`, or `Roberts`, and for the **Output type** parameter, you select `Binary image` or `Binary image and gradient components`.

Approximate percentage of weak edge and nonedge pixels (used to automatically calculate threshold values)

> Enter the approximate percentage of weak edge and nonedge image pixels. The block computes the automatic threshold values using this approximation. This parameter is visible if, for the **Method** parameter, you select `Canny`. Tunable.

Standard deviation of Gaussian filter

> Enter the standard deviation of the Gaussian filter whose derivative is convolved with the input image. This parameter is visible if, for the **Method** parameter, you select `Canny`.

> The **Data Types** pane of the Edge Detection dialog box appears as shown in the following figure.

Edge Detection

Finds the edges in an input image using Sobel, Prewitt, Roberts, or Canny methods.

The block outputs a binary image, a matrix of Boolean values, where pixel values equal to 1 correspond to edges. Alternatively, the block can output two gradient components of the image for the first three methods.

Tabs: Main | Data Types

Settings on this pane only apply when block inputs are fixed-point signals.

Fixed-point operational parameters

Rounding mode: Floor   Overflow mode: Wrap

Fixed-point data types

	Mode	Signed	Word length	Fraction length
Product output	Binary point scaling	Yes	32	8
Accumulator	Same as product output			

☐ Lock data type settings against changes by the fixed-point tools

Buttons: OK | Cancel | Help | Apply

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output

Here, the internal coefficients are the Sobel, Prewitt, or Roberts masks. As depicted in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

The result of each addition remains in the accumulator data type.

Input to adder - product output data type | CAST | Accumulator data type | ADDER | Accumulator data type

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Gradients

Choose how to specify the word length and fraction length of the outputs of the Gv and Gh ports. This parameter is visible if, for the **Output type** parameter, you choose `Gradient components` or `Binary image and gradient components`:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

Edge Detection

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References

[1] Gonzales, Rafael C. and Richard E. Woods. *Digital Image Processing. 2nd ed*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[2] Pratt, William K. *Digital Image Processing, 2nd ed*. New York: John Wiley & Sons, 1991.

See Also   edge   Image Processing Toolbox

Purpose   Find local minima in binary or intensity images

Library   Morphological Operations

visionmorphops
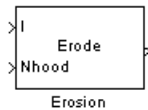
Description   The Erosion block slides the neighborhood or structuring element over an image, finds the local minima, and creates the output matrix from these minimum values. If the neighborhood or structuring element has a center element, the block places the minima there, as illustrated in the following figure.
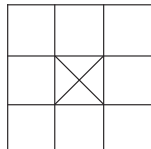
If the neighborhood or structuring element does not have an exact center, the block has a bias toward the upper-left corner and places the minima there, as illustrated in the following figure.
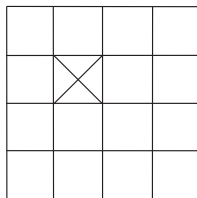
This block uses flat structuring elements only.

Erosion

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of 1s and 0s that represents the neighborhood values	Boolean	No
Output	Vector or matrix of intensity values that represents the eroded image	Same as I port	No

The output signal is the same data type as the input to the I port.

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. You can only specify a structuring element using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the neighborhood or structuring element that the block applies to the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the

use of a more efficient algorithm. If you enter an array of STREL objects, the block applies each object to the entire matrix in turn.

Dialog Box

The Erosion dialog box appears as shown in the following figure.

Neighborhood or structuring element source

Specify how to enter your neighborhood or structuring element values. Select Specify via dialog to enter the values in the dialog box. Select Input port to use the Nhood port to specify the neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the strel function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or structuring element source** parameter, you select Specify via dialog.

Erosion

References   [1] Soille, Pierre. *Morphological Image Analysis. 2nd ed.* New York: Springer, 2003.

See Also

Bottom-hat	Computer Vision System Toolbox software
Closing	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
imerode	Image Processing Toolbox software
strel	Image Processing Toolbox software

Purpose   Estimate geometric transformation from matching point pairs

Library   Geometric Transformations

visiongeotforms

Description

```
> Pts1
> Pts2    Projective TForm >
> Num
    Estimate Geometric
       Transformation
```
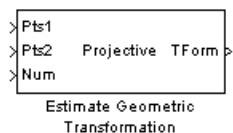
Use the Estimate Geometric Transformation block to find the transformation matrix which maps the greatest number of point pairs between two images. A *point pair* refers to a point in the input image and its related point on the image created using the transformation matrix. You can select to use the RANdom SAmple Consensus (RANSAC) or the Least Median Squares algorithm to exclude outliers and to calculate the transformation matrix. You can also use all input points to calculate the transformation matrix.

Port	Input/Output	Supported Data Types	Complex Values Supported
Pts1/Pts2	*M*-by-2 Matrix of one-based [x y] point coordinates, where *M* represents the number of points.	• Double • Single • 8, 16, 32-bit signed integer • 8, 16, 32-bit unsigned integer	No
Num	Scalar value that represents the number of valid points in **Pts1** and **Pts 2**.	• 8, 16, 32-bit signed integer • 8, 16, 32-bit unsigned integer	No

Estimate Geometric Transformation

Port	Input/Output	Supported Data Types	Complex Values Supported
TForm	3-by-2 or 3-by-3 transformation matrix.	• Double • Single	No
Inlier	*M*-by-1 vector indicating which points have been used to calculate TForm.	Boolean	No

Ports Pts1 and Pts2 are the points on two images that have the same data type. The block outputs the same data type for the transformation matrix

When Pts1 and Pts2 are single or double, the output transformation matrix will also have single or double data type. When Pts1 and Pts2 images are built-in integers, the option is available to set the transformation matrix data type to either Single or Double. The TForm output provides the transformation matrix. The Inlier output port provides the Inlier points on which the transformation matrix is based. This output appears when you select the **Output Boolean signal indicating which point pairs are inliers** checkbox.

RANSAC and Least Median Squares Algorithms

The RANSAC algorithm relies on a distance threshold. A pair of points, p_i^a (image *a*, Pts1) and p_i^b (image *b*, Pts 2) is an inlier only when the distance between p_i^b and the projection of p_i^a based on the transformation matrix falls within the specified threshold. The distance metric used in the RANSAC algorithm is as follows:

$$d = \sum_{i=1}^{Num} \min(D(p_i^b, \psi(p_i^a : H)), t)$$

The Least Median Squares algorithm assumes at least 50% of the point pairs can be mapped by a transformation matrix. The algorithm does not need to explicitly specify the distance threshold. Instead, it uses the median distance between all input point pairs. The distance metric used in the Least Median of Squares algorithm is as follows:

$$d = median(D(p_1^b, \psi(p_1^a : H)), D(p_2^b, \psi(p_2^a : H)), ..., D(p_{Num}^b, \psi(p_N^a : H)))$$

For both equations:

p_i^a is a point in image a (`Pts1`)

p_i^b is a point in image b (`Pts2`)

$\psi(p_i^a : H)$ is the projection of a point on image a based on transformation matrix H

$D(p_i^b, p_j^b)$ is the distance between two point pairs on image b

t is the threshold

Num is the number of points

The smaller the distance metric, the better the transformation matrix and therefore the more accurate the projection image.

Transformations

The Estimate Geometric Transformation block supports `Nonreflective similarity`, `affine`, and `projective` transformation types, which are described in this section.

Nonreflective similarity transformation supports translation, rotation, and isotropic scaling. It has four degrees of freedom and requires two pairs of points.

Estimate Geometric Transformation
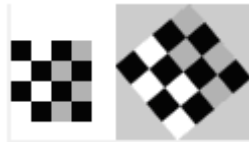
The transformation matrix is: $H = \begin{bmatrix} h_1 & -h_2 \\ h_2 & h_1 \\ h_3 & h_4 \end{bmatrix}$

The projection of a point $\begin{bmatrix} x & y \end{bmatrix}$ by H is: $\begin{bmatrix} \hat{x} & \hat{y} \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} H$

affine transformation supports nonisotropic scaling in addition to all transformations that the nonreflective similarity transformation supports. It has six degrees of freedom that can be determined from three pairs of noncollinear points.

The transformation matrix is: $H = \begin{bmatrix} h_1 & h_4 \\ h_2 & h_5 \\ h_3 & h_6 \end{bmatrix}$

The projection of a point $\begin{bmatrix} x & y \end{bmatrix}$ by H is: $\begin{bmatrix} \hat{x} & \hat{y} \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} H$

Projective transformation supports tilting in addition to all transformations that the affine transformation supports.

The transformation matrix is : $h = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$

The projection of a point $\begin{bmatrix} x & y \end{bmatrix}$ by H is represented by homogeneous coordinates as: $\begin{bmatrix} \hat{u} & \hat{v} & \hat{w} \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} H$

Distance Measurement

For computational simplicity and efficiency, this block uses algebraic distance. The algebraic distance for a pair of points, $\begin{bmatrix} x^a & y^a \end{bmatrix}^T$ on image a, and $\begin{bmatrix} x^b & y^b \end{bmatrix}$ on image b, according to transformation H, is defined as follows;

For projective transformation:

$$D(p_i^b, \psi(p_i^a : H)) = ((u^a - w^a x^b)^2 + (v^a - w^a y^b)^2)^{\frac{1}{2}}, \text{ where}$$

$$\begin{bmatrix} \hat{u}^a & \hat{v}^a & \hat{w}^a \end{bmatrix} = \begin{bmatrix} x^a & y^a & 1 \end{bmatrix} H$$

For Nonreflective similarity or affine transformation:

$$D(p_i^b, \psi(p_i^a : H)) = ((x^a - x^b)^2 + (y^a - y^b)^2)^{\frac{1}{2}},$$

where $\begin{bmatrix} \hat{x}^a & \hat{y}^a \end{bmatrix} = \begin{bmatrix} x^a & y^a & 1 \end{bmatrix} H$

Algorithm

The block performs a comparison and repeats it K number of times between successive transformation matrices. If you select the **Find and exclude outliers** option, the RANSAC and Least Median Squares (LMS) algorithms become available. These algorithms calculate and compare a distance metric. The transformation matrix that produces the smaller distance metric becomes the new transformation matrix that the next comparison uses. A final transformation matrix is resolved when either:

- *K* number of random samplings is performed

Estimate Geometric Transformation

- The RANSAC algorithm, when enough number of inlier point pairs can be mapped, (dynamically updating K)

The Estimate Geometric Transformation algorithm follows these steps:

1

A transformation matrix H is initialized to zeros

2

Set count = 0 (Randomly sampling).

3

While count < K, where K is total number of random samplings to perform, perform the following;

a

Increment the count; count = count + 1.

b

Randomly select pair of points from images a and b, (2 pairs for Nonreflective similarity, 3 pairs for affine, or 4 pairs for projective).

c

Calculate a transformation matrix H, from the selected points.

d

If H has a distance metric less than that of H, then replace H with H.

(Optional for RANSAC algorithm only)

   i.

   Update K dynamically.

   ii.

Exit out of sampling loop if enough number of point pairs can be mapped by H.

4

Use all point pairs in images a and b that can be mapped by H to calculate a refined transformation matrix H

5

Iterative Refinement, (Optional for RANSAC and LMS algorithms)

a

Denote all point pairs that can be mapped by H as inliers.

b

Use inlier point pairs to calculate a transformation matrix H.

c

If H has a distance metric less than that of H, then replace H with H, otherwise exit the loop.

Number of Random Samplings

The number of random samplings can be specified by the user for the RANSAC and Least Median Squares algorithms. You can use an additional option with the RANSAC algorithm, which calculates this number based on an accuracy requirement. The **Desired Confidence** level drives the accuracy.

The calculated number of random samplings, K used with the RANSAC algorithm, is as follows:

$$K = \frac{\log(1-p)}{\log(1-q^s)}$$

where

- p is the probability of independent point pairs belonging to the largest group that can be mapped by the same transformation. The

probability is dynamically calculated based on the number of inliers found versus the total number of points. As the probability increases, the number of samplings, K, decreases.

- q is the probability of finding the largest group that can be mapped by the same transformation.

- s is equal to the value 2, 3, or 4 for Nonreflective similarity, affine, and projective transformation, respectively.

Iterative Refinement of Transformation Matrix

The transformation matrix calculated from all inliers can be used to calculate a refined transformation matrix. The refined transformation matrix is then used to find a new set of inliers. This procedure can be repeated until the transformation matrix cannot be further improved. This iterative refinement is optional.

Dialog Box

Function Block Parameters: Estimate Geometric Transformation

Estimate Geometric Transformation

Find the transformation matrix that maps the largest number of points from Pts1 to Pts

The Pts1 and Pts2 inputs specify the location of points in two images. Each row in the
has the format [x y]. The points in the arrays must be ordered to form corresponding

When the Find and exclude outliers option is selected, the block randomly selects mat
it computes a transformation matrix that fits a specified number of inliers, or until a us
criterion is reached. It then refines this transformation matrix by using all inliers found

Parameters

Transformation type: | Projective

☑ Find and exclude outliers

Method: | Random Sample Consensus (RANSAC)

    Algebraic distance threshold for determining inliers: | 1.5

    Determine number of random samplings using: | Specified value

        Number of random samplings: | 100

        ☐ Stop sampling earlier when a specified percentage of point pairs are dete

☐ Perform additional iterative refinement of the transformation matrix

Outputs

☐ Output Boolean signal indicating which point pairs are inliers

When Pts1 and Pts2 are built-in integers, set transformation matrix data type to: | Sin

☑ Allow variable-size signal input

[OK]   [Cancel]

Estimate Geometric Transformation

Transformation Type

Specify transformation type, either `Nonreflective similarity`, `affine`, or `projective` transformation. If you select `projective` transformation, you can also specify a scalar algebraic distance threshold for determining inliers. If you select either `affine` or `projective` transformation, you can specify the distance threshold for determining inliers in pixels. See "Transformations" on page 1-375 for a more detailed discussion. The default value is `projective`.

Find and exclude outliers

When selected, the block finds and excludes outliers from the input points and uses only the inlier points to calculate the transformation matrix. When this option is not selected, all input points are used to calculate the transformation matrix.

Method

Select either the `RANdom SAmple Consensus (RANSAC)` or the `Least Median of Squares` algorithm to find outliers. See "RANSAC and Least Median Squares Algorithms" on page 1-374 for a more detailed discussion. This parameter appears when you select the **Find and exclude outliers** check box.

Algebraic distance threshold for determining inliers

Specify a scalar threshold value for determining inliers. The threshold controls the upper limit used to find the algebraic distance in the RANSAC algorithm. This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and the **Transformation type** parameter to `projective`. The default value is `1.5`.

Distance threshold for determining inliers (in pixels)

Specify the upper limit distance a point can differ from the projection location of its associating point. This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and you set the value of the **Transformation type** parameter to `Nonreflective similarity` or `affine`. The default value is `1.5`.

Determine number of random samplings using

    Select `Specified value` to enter a positive integer value for number of random samplings, or select `Desired confidence` to set the number of random samplings as a percentage and a maximum number. This parameter appears when you select `Find and exclude outliers parameter`, and you set the value of the **Method** parameter to `Random Sample Consensus (RANSAC)`.

Number of random samplings

    Specify the number of random samplings for the algorithm to perform. This parameter appears when you set the value of the **Determine number of random samplings using** parameter to `Specified value`.

Desired confidence (in %)

    Specify a percent by entering a number between `0` and `100`. The `Desired confidence` value represents the probability of the algorithm to find the largest group of points that can be mapped by a transformation matix. This parameter appears when you set the **Determine number of random samplings using** parameter to `Desired confidence`.

Maximum number of random samplings

    Specify an integer number for the maximum number of random samplings. This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and you set the value of the **Determine number of random samplings using** parameter to `Desired confidence.`

Stop sampling earlier when a specified percentage of point pairs are determined to be inlier

    Specify to stop random sampling when a percentage of input points have been found as inliers. This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC).`

Perform additional iterative refinement of the transformation matrix

Specify whether to perform refinement on the transformation matrix. This parameter appears when you select **Find and exclude outliers** check box.

Output Boolean signal indicating which point pairs are inliers

Select this option to output the inlier point pairs that were used to calculate the transformation matrix. This parameter appears when you select **Find and exclude outliers** check box. The block will not use this parameter with signed or double, data type points.

When Pts1 and Pts2 are built-in integers, set transformation matrix date type to

Specify transformation matrix data type as `Single` or `Double` when the input points are built-in integers. The block will not use this parameter with signed or double, data type points.

Examples

Calculate transformation matrix from largest group of point pairs

Examples of input data and application of the Estimate Geometric Transformation block appear in the following figures. Figures (a) and (b) show the point pairs. The points are denoted by stars or circles, and the numbers following them show how they are paired. Some point pairs can be mapped by the same transformation matrix. Other point pairs require a different transformation matrix. One matrix exists that maps the largest number of point pairs, the block calculates and returns this matrix. The block finds the point pairs in the largest group and uses them to calculate the transformation matrix. The point pairs connected by the magenta lines are the largest group.

The transformation matrix can then be used to stitch the images as shown in Figure (e).

(a)

(b)
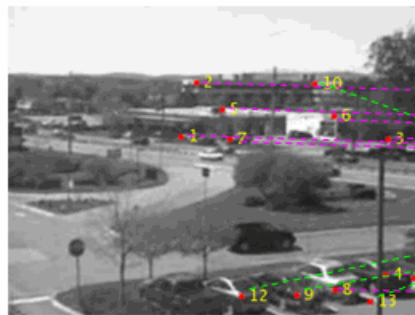
(c)
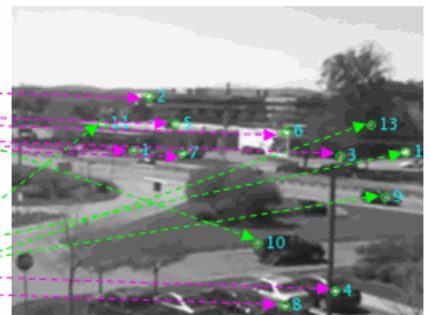
(d)

(e)

Estimate Geometric Transformation

Video Mosaicking

To see an example of the Estimate Geometric Transformation block used in a model with other blocks, see the "Video Mosaicking" example.

Troubleshooting The success of estimating the correct geometric transformation depends heavily on the quality of the input point pairs. If you chose the RANSAC or LMS algorithm, the block will randomly select point pairs to compute the transformation matrix and will use the transformation that best fits the input points. There is a chance that all of the randomly selected point pairs may contain outliers despite repeated samplings. In this case, the output transformation matrix, TForm, is invalid, indicated by a matrix of zeros.

To improve your results, try the following:

Increase the percentage of inliers in the input points.

Increase the number for random samplings.

For the RANSAC method, increase the desired confidence.

For the LMS method, make sure the input points have 50% or more inliers.

Use features appropriate for the image contents

Be aware that repeated patterns, for example, windows in office building, will cause false matches when you match the features. This increases the number of outliers.

Do not use this function if the images have significant parallax. You can use the estimateFundamentalMatrix function instead.

Choose the minimum transformation for your problem.

If a projective transformation produces the error message, "A portion of the input image was transformed to the location at infinity. Only transformation matrices that do not transform any part of the image to infinity are supported.", it is usually caused by a transformation matrix and an image that would result in an output distortion that does not fit physical reality. If the matrix was an output of the Estimate Geometric Transformation block, then most likely it could not find enough inliers.

References   R. Hartley and A. Ziserman, "Multiple View Geometry in Computer Vision," Second edition, Cambridge University Press, 2003

See Also

cp2tform	Image Processing Toolbox
vipmosaicking	Computer Vision System Toolbox

Estimate Geometric Transformation (To Be Removed)

Purpose   Estimate geometric transformation from matching point pairs

Library   Geometric Transformations

Description
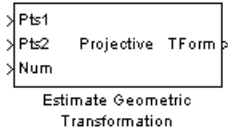
> **Note** This Estimate Geometric Transformation block will be removed
> in a future release. It uses the zero-based, [row column] coordinate
> system. It is recommended that you replace this block with the updated
> Estimate Geometric Transformation block that uses the one-based, [x y]
> coordinate system.
>
> Refer to "Conventions Changed for Indexing, Spatial Coordinates, and
> Representation of Geometric Transforms" in the R2011b Release Notes
> for details related to these changes.
>
> See the Estimate Geometric Transformation documentation to view the
> reference material related to the obsolete block.

Purpose   Find local maxima in matrices

Library   Statistics

visionstatistics

Description   The Find Local Maxima block finds the local maxima within an input matrix. It does so by comparing the maximum value in the matrix to a user-specified threshold. The block considers a value to be a valid local maximum when the maximum value is greater than or equal to the specified threshold. The determination of the local maxima is based on the *neighborhood*, an area around and including the maximum value. After finding the local maxima, the block sets all the matrix values in the neighborhood, including the maximum value, to 0. This step ensures that subsequent searches do not include this maximum. The size of the neighborhood must be appropriate for the data set. That is, the threshold must eliminate enough of the values around the maximum so that false peaks are not discovered. The process repeats until the block either finds all valid maximas or the number of local maximas equal the **Maximum number of local maxima** value. The block outputs one-based [x y] coordinates of the maxima. The data to all input ports must be the same data type.

If the input to this block is a Hough matrix output from the Hough Transform block, select the **Input is Hough matrix spanning full theta range** check box. If you select this check box, the block assumes that the Hough port input is antisymmetric about the rho axis and theta ranges from -pi/2 to pi/2 radians. If the block finds a local maxima near the boundary, and the neighborhood lies outside the Hough matrix, then the block detects only one local maximum. It ignores the corresponding antisymmetric maximum.

Find Local Maxima

Dialog Box

The Find Local Maxima dialog box appears as shown in the following figure.

Function Block Parameters: Find Local Maxima

Find Local Maxima

Finds local maxima in an input matrix.

The size of the search region can be specified using the Neighborhood size parameter. Idx output port has the dimension M-by-2, where M is the maximum number of local maxima. This output port holds one-based [x y] coordinates of the local maxima. Threshold value is applied on the input matrix to find the valid local maxima.

If the input is a Hough matrix generated from a Hough Transform block, the Input is Hough matrix spanning full theta range check box should be selected. In this case the block considers that the input Hough matrix is antisymmetric about the rho axis and theta spans from -pi/2 to pi/2 radians. If a local maximum is found near a boundary such that the neighboring window lies outside the Hough matrix, the block picks up only one peak and ignores the corresponding antisymmetric peak.

Parameters

Maximum number of local maxima: 2

Neighborhood size: [5 7]

Source of threshold value: Specify via dialog

Threshold: 10

☐ Input is Hough matrix spanning full theta range

Index output data type: uint32

☑ Output variable size signal

OK   Cancel   Help   Apply

Maximum number of local maxima

Specify the maximum number of maxima you want the block to find.

Neighborhood size

Specify the size of the neighborhood around the maxima over which the block zeros out the values. Enter a two-element vector of positive odd integers, [*rc*]. Here, *r* represents the number of rows in the neighborhood, and *c* represents the number of columns.

Source of threshold value

Specify how to enter the threshold value. If you select `Input port`, the **Th** port appears on the block. If you select `Specify via dialog`, the **Threshold** parameter appears in the dialog box. Enter a scalar value that represents the value all maxima should meet or exceed.

Threshold

Enter a scalar value that represents the value all maxima should meet or exceed. This parameter is visible if, for the **Source of threshold value** parameter, you choose `Specify via dialog`.

Input is Hough matrix spanning full theta range

If you select this check box, the block assumes that the Hough port input is antisymmetric about the rho axis and theta ranges from -pi/2 to pi/2 radians.

Index output data type

Specify the data type of the **Idx** port output. Your choices are `double`, `single`, `uint8`, `uint16`, or `uint32`.

Output variable size signal

Specify output data type. When you uncheck the **Output variable size signal**, the **Count output data type** parameter appears in the dialog box.

Count output data types

Specify the data type of the Count port output. Your choices are `double`, `single`, `uint8`, `uint16`, or `uint32`. This parameter

Find Local Maxima

applies when you clear the **Output variable size signal** check box.

Examples   See "Detect Lines in Images" and "Measure Angle Between Lines" in the *Computer Vision System Toolbox User's Guide*.

Supported Data Types

The block outputs the one-based [x y] coordinates of the maxima at the **Idx** port and the number of valid local maxima found at the **Count** port.

Port	Input/Output	Supported Data Types	Complex Values Supported
I/Hough	Matrix in which you want to find the maxima.	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Th	Scalar value that represents the value the maxima should meet or exceed.	Same as I/Hough port	No
Idx	An *M*-by-2 matrix of one-based [x y] coordinates, where *M* represents the number of local maximas found.	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit unsigned integer	No
Count	Scalar value that represents the number of maxima that meet or exceed the threshold value.	Same as Idx port	No

See Also

Hough Lines	Computer Vision System Toolbox
Hough Transform	Computer Vision System Toolbox
Corner Detection	Computer Vision System Toolbox
houghpeaks	Image Processing Toolbox
vision.CornerDetector	Computer Vision System Toolbox
vision.HoughLines	Computer Vision System Toolbox
vision.HoughTransform	Computer Vision System Toolbox

Find Local Maxima (To Be Removed)

Purpose   Find local maxima in matrices

Library   Statistics

Description
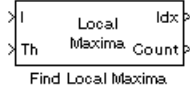
> **Note** This Find Local Maxima block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Find Local Maxima block that uses the one-based, [x y] coordinate system.
>
> Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.
>
> See the Find Local Maxima documentation to view the reference material related to the obsolete block.

Purpose   Calculate average update rate of input signal

Library   Sinks

                   visionsinks

Description   The Frame Rate Display block calculates and displays the average
                   update rate of the input signal. This rate is in relation to the wall clock
                   time. For example, if the block displays 30, the model is updating the
                   input signal 30 times every second. You can use this block to check the
                   video frame rate of your simulation. During code generation, Simulink
                   Coder does not generate code for this block.

Note This block supports intensity and color images on its port.

Port	Input	Supported Data Types	Complex Values Supported
Input	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No

Use the **Calculate and display rate every** parameter to control how
often the block updates the display. When this parameter is greater
than 1, the block displays the average update rate for the specified
number of video frames. For example, if you enter 10, the block
calculates the amount of time it takes for the model to pass 10 video
frames to the block. It divides this time by 10 and displays this average
video frame rate on the block.

Frame Rate Display

> **Note** If you do not connect the Frame Rate Display block to a signal line, the block displays the base (fastest) rate of the Simulink model.

Dialog Box

The Frame Rate Display dialog box appears as shown in the following figure.
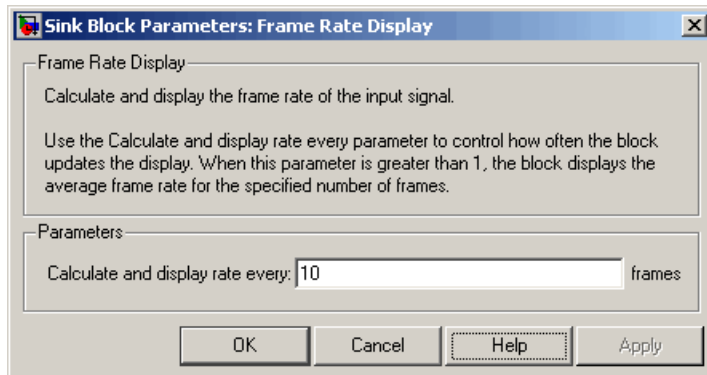
Calculate and display rate every

Use this parameter to control how often the block updates the display.

See Also

To Multimedia File	Computer Vision System Toolbox software
To Video Display	Computer Vision System Toolbox software
Video To Workspace	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software

Purpose   Read video frames and audio samples from compressed multimedia file

Library   Sources

visionsources

From Multimedia File
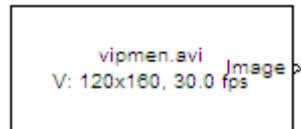
Description

The From Multimedia File block reads audio samples, video frames, or both from a multimedia file. The block imports data from the file into a Simulink model.

Note This block supports code generation for the host computer that has file I/O available. You cannot use this block with Real-Time Windows Target™ software because that product does not support file I/O.

The generated code for this block relies on prebuilt library files. You can run this code outside the MATLAB environment, or redeploy it, but be sure to account for these extra library files when doing so. The packNGo function creates a single zip file containing all of the pieces required to run or rebuild this code. See packNGo for more information.

To run an executable file that was generated from a model containing this block, you may need to add precompiled shared library files to your system path. See "Simulink Coder", "Shared Library Dependencies", and "Accelerating Simulink Models" for details.

This block allows you to read WMA/WMV streams to disk or across a network connection. Similarly, the To Multimedia File block allows you to write WMA/WMV streams to disk or across a network connection. If you want to play an MP3/MP4 file in Simulink, but you do not have the

From Multimedia File

codecs, you can re-encode the file as WMA/WMV, which are supported by the Computer Vision System Toolbox.

Supported Platforms and File Types

The supported file formats available to you depend on the codecs installed on your system.

Platform	Supported File Name Extensions
All Platforms	AVI (.avi)
Windows®	**Image**: .jpg,.bmp
	Video: MPEG (.mpeg) MPEG-2 (.mp2) MPEG-1 .mpg MPEG-4, including H.264 encoded video (.mp4, .m4v) Windows Media Video (.wmv,.asf, .asx, .asx) and any format supported by Microsoft DirectShow® 9.0 or higher.
	Audio: WAVE (.wav) Windows Media Audio File (.wma) Audio Interchange File Format (.aif, .aiff) Compressed Audio Interchange File Format(.aifc), MP3 (.mp3) Sun Audio (.au) Apple (.snd)

Platform	Supported File Name Extensions
Macintosh	**Video**: `.avi` Motion JPEG 2000 (`.mj2`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (`.mov`) and any format supported by QuickTime as listed on http://support.apple.com/kb/HT3775.
	Audio: Uncompressed `.avi`
Linux®	Any format supported by your installed plug-ins for GStreamer 0.10 or above, as listed on http://gstreamer.freedesktop.org/documentation/plugins.html, including Ogg Theora (.ogg).

Windows XP and Windows 7 x64 platform ships with a limited set of 64-bit video and audio codecs. If a compressed multimedia file fails to play, try one of the two alternatives:

- Run the 32-bit version of MATLAB on your Windows XP x64 platform. Windows XP x64 ships with many 32-bit codecs.

- Save the multimedia file to a supported file format listed in the table above.

If you use Windows, use Windows Media player Version 11 or later.

Ports   The output ports of the From Multimedia File block change according to the content of the multimedia file. If the file contains only video frames, the **Image**, intensity **I**, or **R**,**G**,**B** ports appear on the block. If the file contains only audio samples, the **Audio** port appears on the block. If the file contains both audio and video, you can select the data to emit. The following table describes available ports.

From Multimedia File

Port	Description
Image	*M*-by-*N*-by-*P* color video signal where *P* is the number of color planes.
I	*M*-by-*N* matrix of intensity values.
R, **G**, **B**	Matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports must have same dimensions.
Audio	Vector of audio data.
Y, Cb, Cr	Matrix that represents one frame of the YCbCr video stream. The Y, Cb, Cr ports produce the following outputs: Y: $M \times N$ Cb: $M \times \dfrac{N}{2}$ Cr: $M \times \dfrac{N}{2}$

Sample Rates

The sample rate that the block uses depends on the audio and video sample rate. While the FMMF block operates at a single rate in Simulink, the underlying audio and video streams can produce different rates. In some cases, when the block outputs both audio and video, makes a small adjustment to the video rate.

Sample Time Calculations Used for Video and Audio Files

Sample time = $\dfrac{ceil(AudioSampleRate/FPS)}{AudioSampleRate}$.

When audio sample time, $\dfrac{AudioSampleRate}{FPS}$ is noninteger, the equation cannot reduce to $\dfrac{1}{FPS}$.

In this case, to prevent synchronization problems, the block drops the corresponding video frame when the audio stream leads the video

stream by more than $\dfrac{1}{FPS}$.

In summary, the block outputs one video frame at each Simulink time step. To calculate the number of audio samples to output at each time step, the block divides the audio sample rate by the video frame rate (fps). If the audio sample rate does not divide evenly by the number of video frames per second, the block rounds the number of audio samples up to the nearest whole number. If necessary, the block periodically drops a video frame to maintain synchronization for large files.

From Multimedia File

Dialog Box

The **Main** pane of the From Multimedia File block dialog appears as follows.
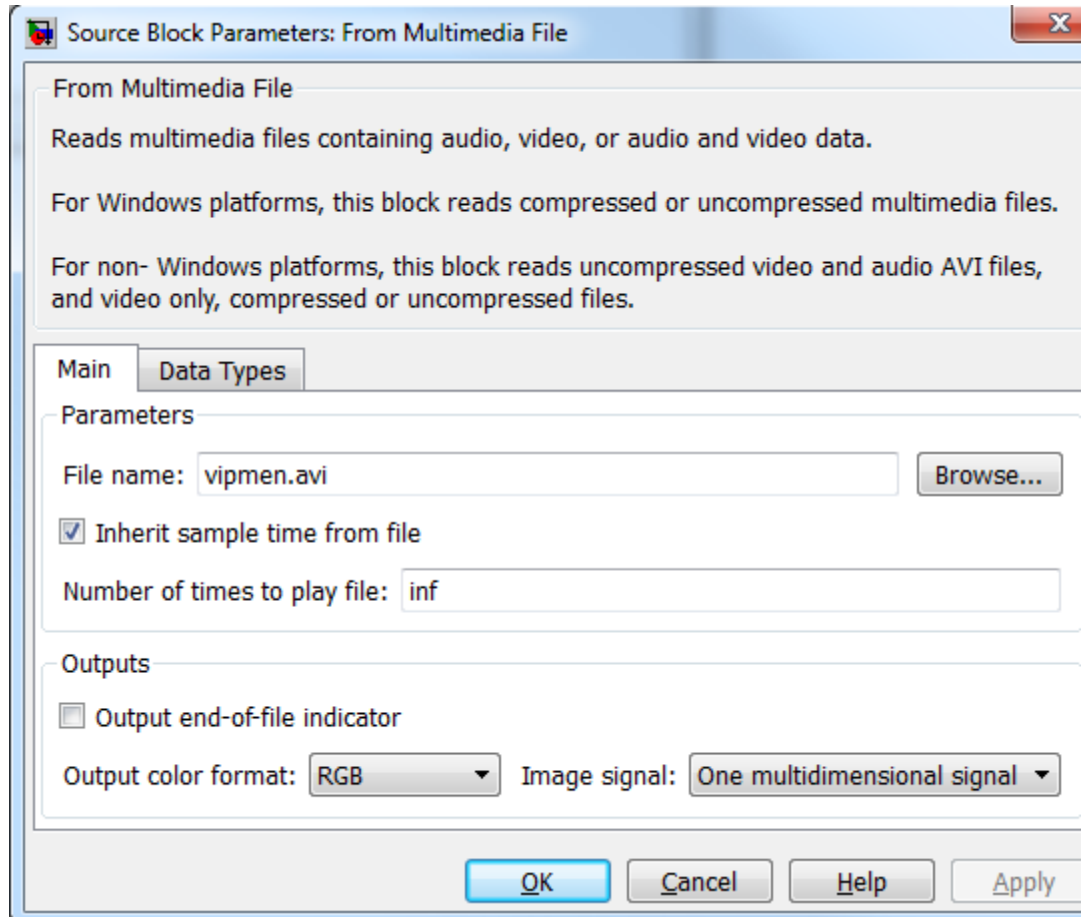
File name

> Specify the name of the multimedia file from which to read. The block determines the type of file (audio and video, audio only, or video only) and provides the associated parameters.

If the location of the file does not appear on your MATLAB path, use the **Browse** button to specify the full path. Otherwise, if the location of this file appears on your MATLAB path, enter only the file name. On Windows platforms, this parameter supports URLs that point to MMS (Microsoft Media Server) streams.

Inherit sample time from file

Select the **Inherit sample time from file** check box if you want the block sample time to be the same as the multimedia file. If you clear this check box, enter the block sample time in the **Desired sample time** parameter field. The file that the From Multimedia File block references, determines the block default sample time. You can also set the sample time for this block manually. If you do not know the intended sample rate of the video, let the block inherit the sample rate from the multimedia file.

Desired sample time

Specify the block sample time. This parameter becomes available if you clear the **Inherit sample time from file** check box.

Number of times to play file

Enter a positive integer or `inf` to represent the number of times to play the file.

Output end-of-file indicator

Use this check box to determine whether the output is the last video frame or audio sample in the multimedia file. When you select this check box, a Boolean output port labeled EOF appears on the block. The output from the EOF port defaults to 1 when the last video frame or audio sample is output from the block. Otherwise, the output from the EOF port defaults to 0.

Multimedia outputs

Specify `Video and audio`, `Video only`, or `Audio only` output file type. This parameter becomes available only when a video signal has both audio and video.

Samples per audio channel

Specify number of samples per audio channel. This parameter becomes available for files containing audio.

From Multimedia File

Output color format

Specify whether you want the block to output `RGB`, `Intensity`, or `YCbCr 4:2:2` video frames. This parameter becomes available only for a signal that contains video. If you select `RGB`, use the **Image signal** parameter to specify how to output a color signal.

Image signal

Specify how to output a color video signal. If you select `One multidimensional signal`, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream. This parameter becomes available only if you set the **Image color space** parameter to `RGB` and the signal contains video.

Audio output sampling mode

Select **Sample based** or **Frame based** output. This parameter appears when you specify a file containing audio for the **File name** parameter.

The **Data Types** pane of the To Multimedia File block dialog box appears as follows.
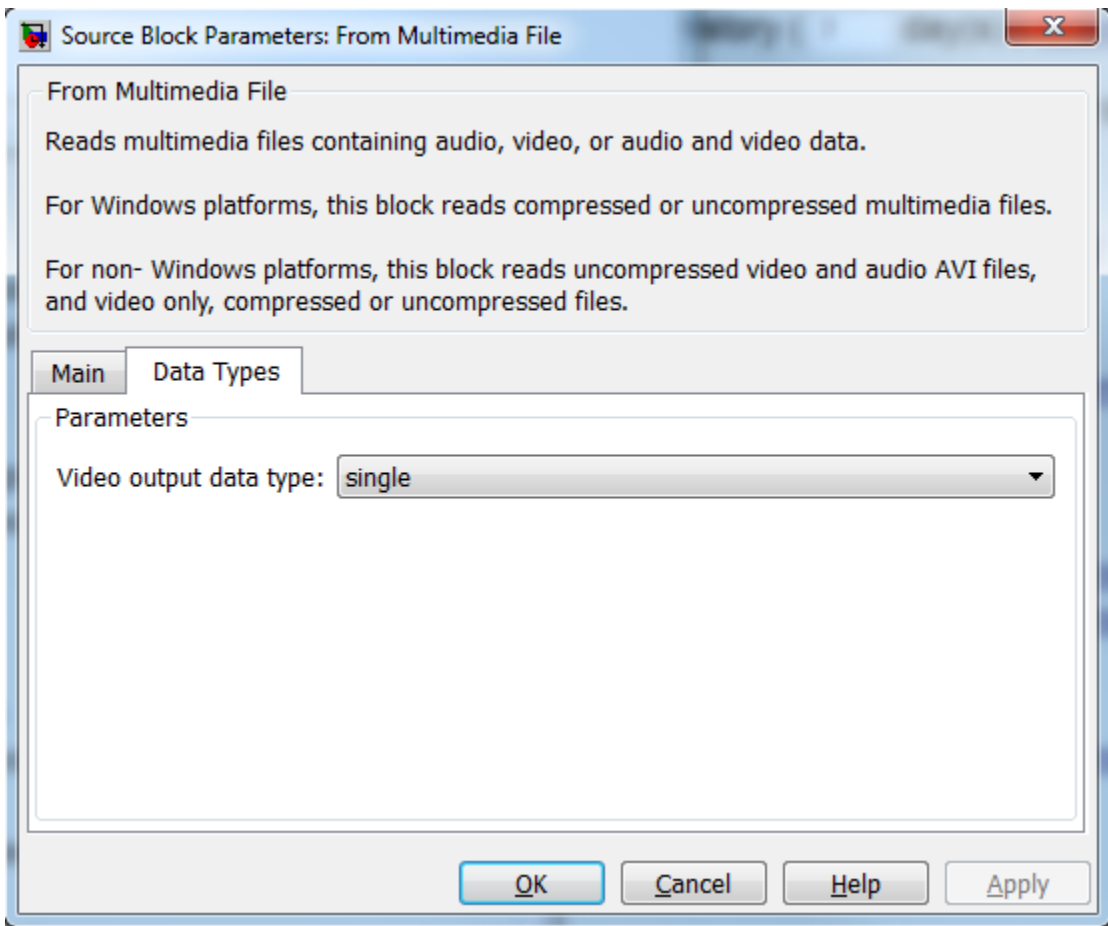
Audio output data type

Set the data type of the audio samples output at the Audio port. This parameter becomes available only if the multimedia file contains audio. You can choose `double`, `single`, `int16`, or `uint8` types.

Video output data type

Set the data type of the video frames output at the **R**, **G**, **B**, or **Image** ports. This parameter becomes available only if the multimedia file contains video. You can choose double, single, int8, uint8, int16, uint16, int32, uint32, or Inherit from file types.

Supported Data Types

For source blocks to display video data properly, double- and single-precision floating-point pixel values must be between 0 and 1. For other data types, the pixel values must be between the minimum and maximum values supported by their data type.

Port	Supported Data Types	Supports Complex Values?
Image	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers	No
R, G, B	Same as the Image port	No
Audio	• Double-precision floating point • Single-precision floating point • 16-bit signed integers • 8-bit unsigned integers	No
Y, Cb,Cr	Same as the Image port	No

See Also   To Multimedia File   Computer Vision System Toolbox

"Specify Sample   Simulink
Time"

Gamma Correction

Purpose   Apply or remove gamma correction from images or video streams

Library   Conversions

visionconversions
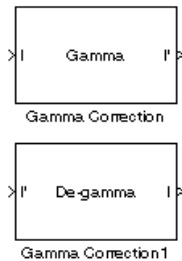
Description   Use the Gamma Correction block to apply or remove gamma correction from an image or video stream. For input signals normalized between 0 and 1, the block performs gamma correction as defined by the following equations. For integers and fixed-point data types, these equations are generalized by applying scaling and offset values specific to the data type:

$$S_{LS} = \frac{1}{\dfrac{\gamma}{B_P^{(1/\gamma - 1)}} - \gamma B_P + B_P}$$

$$F_S = \frac{\gamma S_{LS}}{B_P^{(1/\gamma - 1)}}$$

$$C_O = F_S B_P^{1/\gamma} - S_{LS} B_P$$

$$I' = \left\{ \begin{array}{ll} S_{LS}I, & I \leq B_P \\ F_S I^{1/\gamma} - C_O, & I > B_P \end{array} \right\}$$

S_{LS} is the slope of the straight line segment. B_P is the break point of the straight line segment, which corresponds to the **Break point** parameter. F_S is the slope matching factor, which matches the slope of the linear segment to the slope of the power function segment. C_O is the segment offset, which ensures that the linear segment and the power function segments connect. Some of these parameters are illustrated by the following diagram.
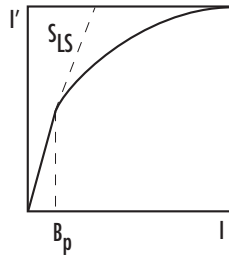
For normalized input signals, the block removes gamma correction, which linearizes the input video stream, as defined by the following equation:

$$
I = \begin{cases} \dfrac{I'}{S_{LS}}, & I' \le B_P \\[2mm] \left(\dfrac{I' + C_O}{F_S}\right)^{\gamma}, & I' > B_P \end{cases}
$$

Typical gamma values range from 1 to 3. Most monitor gamma values range from 1.8 to 2.2. Check with the manufacturer of your hardware to obtain the exact gamma value. Gamma function parameters for some common standards are shown in the following table:

Standard	Slope	Break Point	Gamma
CIE L*	9.033	0.008856	3
Recommendation ITU-R BT.709-3, Parameter Values for the HDTV Standards for Production and International Programme Exchange	4.5	0.018	$20/9$
sRGB	12.92	0.00304	2.4

Gamma Correction

The properties of the input and output ports are summarized in the following table:

Port	Input/Output	Supported Data Types	Complex Values Supported
I	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	- Double-precision floating point - Single-precision floating point - Fixed point (up to 16-bit word length) 8- and 16-bit signed integer 8- and 16-bit unsigned integer	No
I'	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	Same as I port	No

Use the **Operation** parameter to specify the block's operation. If you want to perform gamma correction, select Gamma. If you want to linearize the input signal, select De-gamma.

If, for the **Operation** parameter, you select Gamma, use the **Gamma** parameter to enter the desired gamma value of the output video stream. This value must be greater than or equal to 1. If, for the **Operation** parameter, you select De-gamma, use the **Gamma** parameter to enter the gamma value of the input video stream.

Select the **Linear segment** check box if you want the gamma curve to have a linear portion near black. If you select this check box, the **Break point** parameter appears on the dialog box. Enter a scalar value that

indicates the *I*-axis value of the end of the linear segment. The break point is shown in the first diagram of this block reference page.

Dialog Box

The Gamma Correction dialog box appears as shown in the following figure.
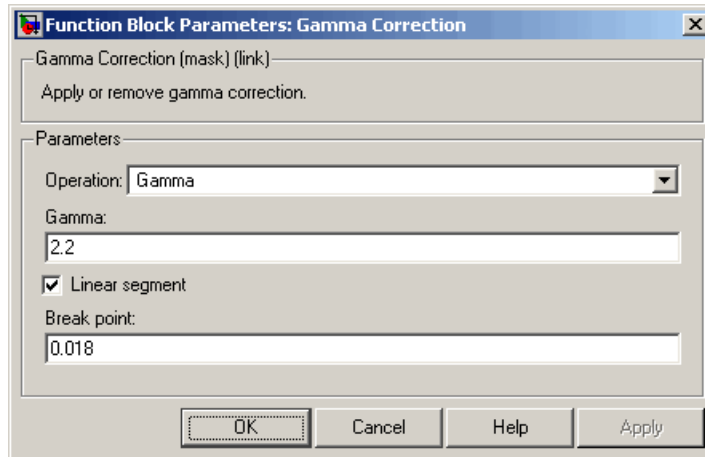
Operation

Specify the block's operation. Your choices are Gamma or De-gamma.

Gamma

If, for the **Operation** parameter, you select Gamma, enter the desired gamma value of the output video stream. This value must be greater than or equal to 1. If, for the **Operation** parameter, you select De-gamma, enter the gamma value of the input video stream.

Linear segment

Select this check box if you want the gamma curve to have a linear portion near the origin.

Gamma Correction

Break point
Enter a scalar value that indicates the *I*-axis value of the end of the linear segment. This parameter is visible if you select the **Linear segment** check box.

References   [1] Poynton, Charles. *Digital Video and HDTV Algorithms and Interfaces*. San Francisco, CA: Morgan Kaufman Publishers, 2003.

See Also

Color Space Conversion	Computer Vision System Toolbox software
imadjust	Image Processing Toolbox software

Purpose　　Perform Gaussian pyramid decomposition

Library　　Transforms

`visiontransforms`

Description　　The Gaussian Pyramid block computes Gaussian pyramid reduction or expansion to resize an image. The image reduction process involves lowpass filtering and downsampling the image pixels. The image expansion process involves upsampling the image pixels and lowpass filtering. You can also use this block to build a Laplacian pyramid. For more information, see "Examples" on page 1-415.

Note This block supports intensity and color images on its ports.

| Port	Output	Supported Data Types	Complex Values Supported
Input	In `Reduce` mode, the input can be an M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes. In `Expand` mode, the input can be a scalar, vector, or M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes.	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Output	In `Reduce` mode, the output can be a scalar, vector, or matrix that	Same as Input port	No

Gaussian Pyramid

Port	Output	Supported Data Types	Complex Values Supported
	represents one level of a Gaussian pyramid. In Expand mode, the output can be a matrix that represents one level of a Gaussian pyramid.		

Use the **Operation** parameter to specify whether to reduce or expand the input image. If you select Reduce, the block applies a lowpass filter and then downsamples the input image. If you select Expand, the block upsamples and then applies a lowpass filter to the input image.

Use the **Pyramid level** parameter to specify the number of times the block upsamples or downsamples each dimension of the image by a factor of 2. For example, suppose you have a 4-by-4 input image. You set the **Operation** parameter to Reduce and the **Pyramid level** to 1. The block filters and downsamples the image and outputs a 2-by-2 pixel output image. If you have an M-by-N input image and you set the **Operation** parameter to Reduce, you can calculate the dimensions of the output image using the following equation:

$$\mathrm{ceil}\left(\frac{M}{2}\right) - \mathrm{by} - \mathrm{ceil}\left(\frac{N}{2}\right)$$

You must repeat this calculation for each successive pyramid level. If you have an M-by-N input image and you set the **Operation** parameter to Expand, you can calculate the dimensions of the output image using the following equation:

$$\left[(M-1)2^l + 1\right] - \mathrm{by} - \left[(N-1)2^l + 1\right]$$

In the previous equation, l is the scalar value from 1 to inf that you enter for the **Pyramid level** parameter.

Use the **Coefficient source** parameter to specify the coefficients of the lowpass filter. If you select `Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]`, use the **a** parameter to define the coefficients in the vector of separable filter coefficients. If you select `Specify via dialog`, use the **Coefficient for separable filter** parameter to enter a vector of separable filter coefficients.

Examples

The following example model shows how to construct a Laplacian pyramid:

1 Open this model by typing

```
ex_laplacian
```

at the MATLAB command prompt.
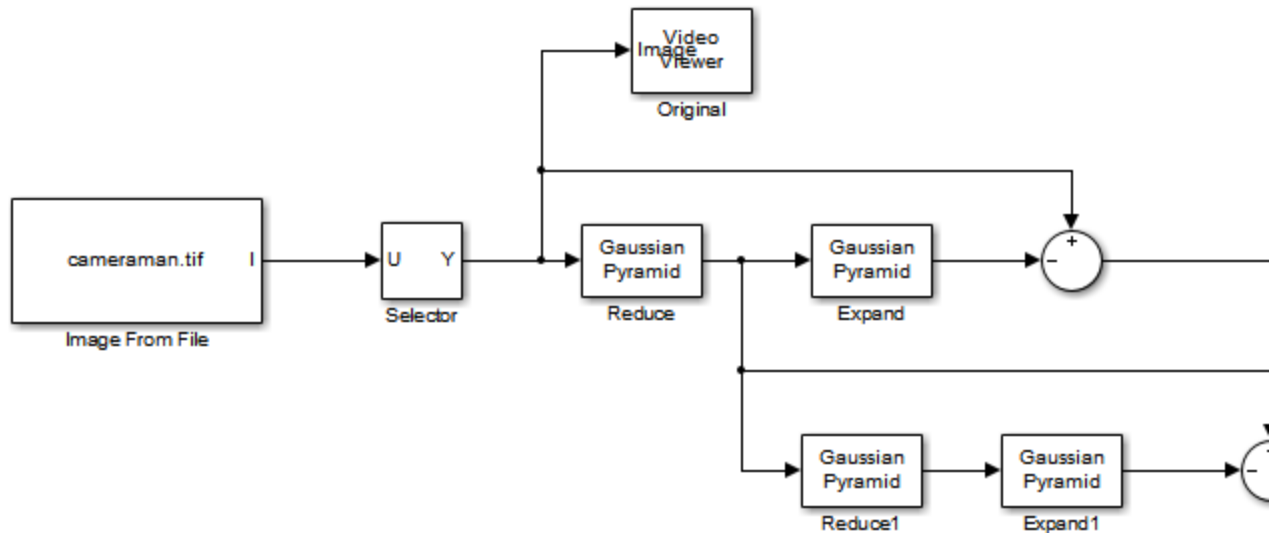
2 Run the model to see the following results.

Gaussian Pyramid
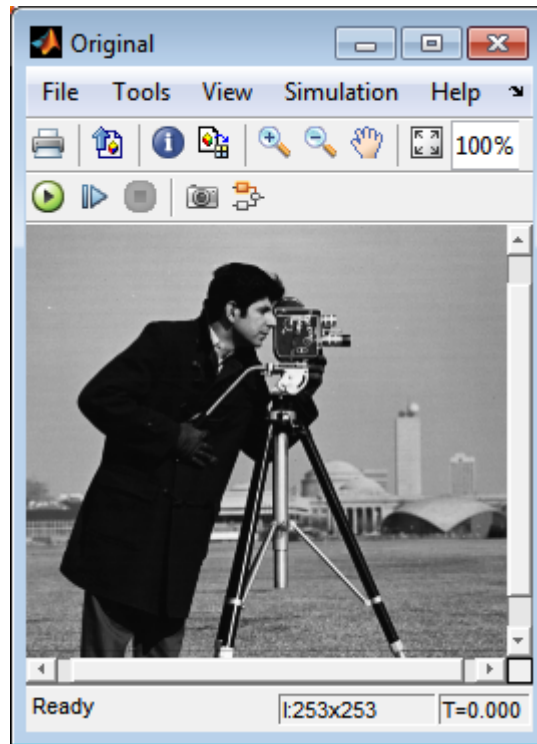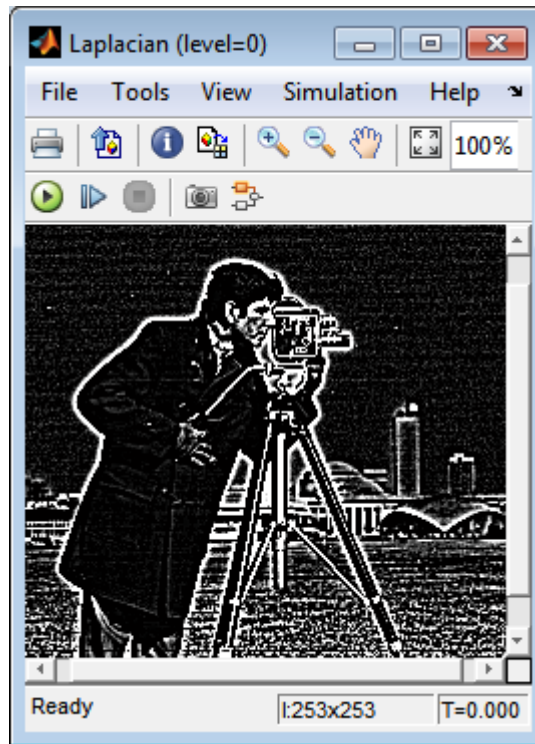
Gaussian Pyramid
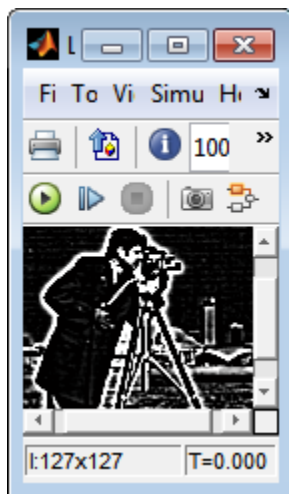
You can construct a Laplacian pyramid if the dimensions of the input image, R-by-C, satisfy $R = M_R 2^N + 1$ and $C = M_c 2^N + 1$, where M_R, M_C, and N are integers. In this example, you have an input matrix that is 256-by-256. If you set M_R and M_C equal to 63 and N equal to 2, you find that the input image needs to be 253-by-253. So you use a Submatrix block to crop the dimensions of the input image to 253-by-253.

Fixed-Point Data Types

The following diagram shows the data types used in the Gaussian Pyramid block for fixed-point signals:

The result of each addition remains
in the accumulator data type.

```
Input          →  MULTIPLIER  →  CAST  →  ADDER  →  CAST  →  Output data
data type(s)      Product output  Accumulator  Accumulator  type
                  data type       data type    data type
```
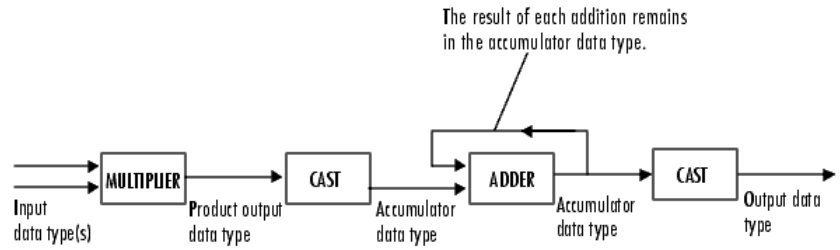
You can set the coefficients table, product output, accumulator, and
output data types in the block mask.

Gaussian Pyramid

Dialog Box

The **Main** pane of the Gaussian Pyramid dialog box appears as shown in the following figure.

Function Block Parameters: Gaussian Pyramid

Gaussian Pyramid

This block computes Gaussian pyramid reduction or expansion. The input is either upsam and a lowpass filter is applied pre- or post-sampling.

| Main | Data Types |

Parameters

Operation: Reduce

Pyramid level: 1

Coefficient source: Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]

a: 0.375

Coefficients for separable filter: [0.0625 0.25 0.375 0.25 0.0625]

OK   Cancel
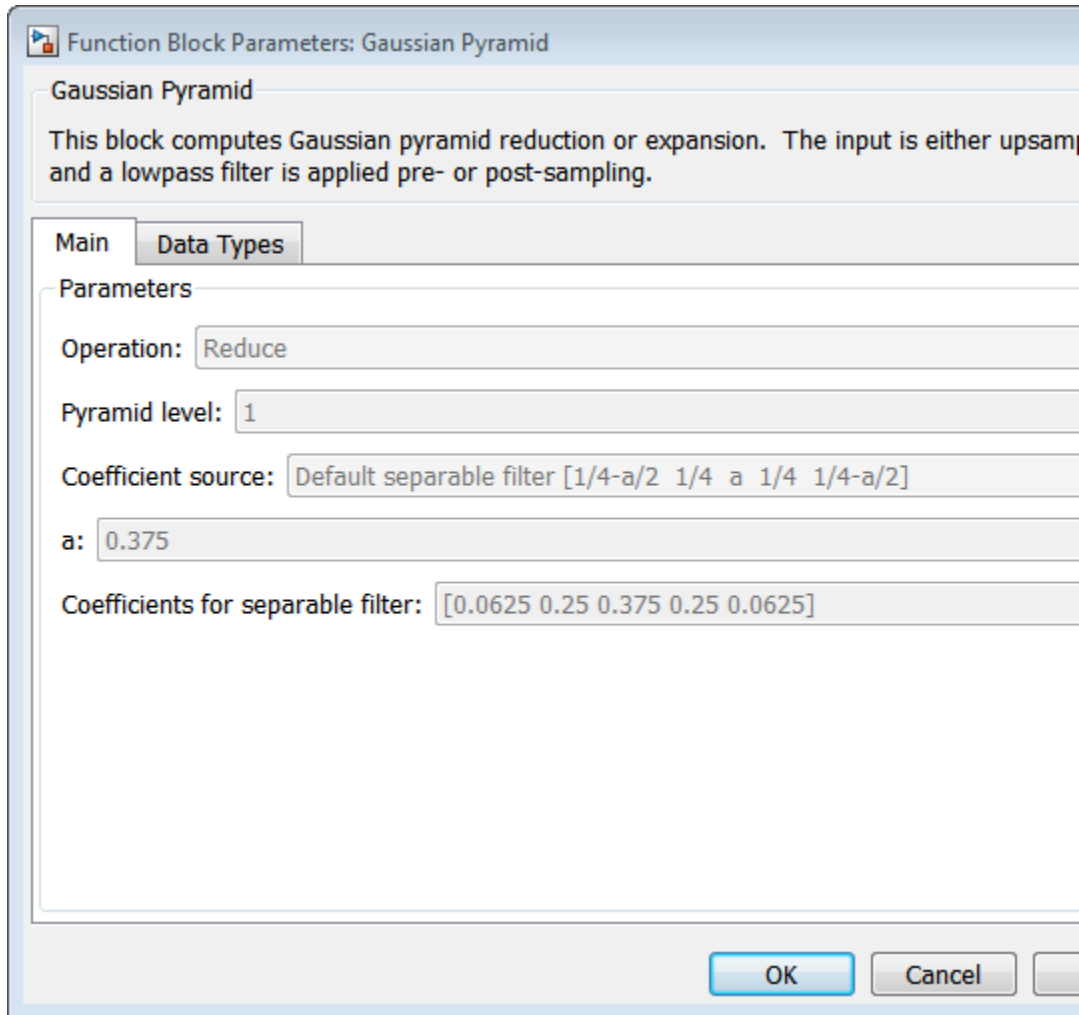
Operation

    Specify whether you want to reduce or expand the input image.

Pyramid level

    Specify the number of times the block upsamples or downsamples each dimension of the image by a factor of 2.

Coefficient source

    Determine how to specify the coefficients of the lowpass filter. Your choices are `Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]` or `Specify via dialog`.

a

    Enter a scalar value that defines the coefficients in the default separable filter `[1/4-a/2 1/4 a 1/4 1/4-a/2]`. This parameter is visible if, for the **Coefficient source** parameter, you select `Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]`.

Coefficients for separable filter

    Enter a vector of separable filter coefficients. This parameter is visible if, for the **Coefficient source** parameter, you select `Specify via dialog`.

The **Data Types** pane of the Gaussian Pyramid dialog box appears as shown in the following figure.

Gaussian Pyramid

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Coefficients

Choose how to specify the word length and the fraction length of the coefficients:

- When you select `Same word length as input`, the word length of the coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. The block automatically sets the fraction length to give you the best precision.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Product output

As shown in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate the product output word and fraction lengths.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

As shown in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate the accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

See Also   Resize   Computer Vision System Toolbox software

Histogram Equalization

Purpose
Enhance contrast of images using histogram equalization

Library
Analysis & Enhancement

visionanalysis

Description

The Histogram Equalization block enhances the contrast of images by transforming the values in an intensity image so that the histogram of the output image approximately matches a specified histogram.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Hist	Vector of integer values that represents the desired intensity values in each bin	• Double-precision floating point • Single-precision floating point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Output	Matrix of intensity values	Same as I port	No

If the data type of input to the I port is floating point, the input to Hist port must be the same data type. The output signal has the same data type as the input signal.

Use the **Target histogram** parameter to designate the histogram you want the output image to have.

If you select Uniform, the block transforms the input image so that the histogram of the output image is approximately flat. Use the **Number of bins** parameter to enter the number of equally spaced bins you want the uniform histogram to have.

If you select User-defined, the **Histogram source** and **Histogram** parameters appear on the dialog box. Use the **Histogram source** parameter to select how to specify your histogram. If, for the **Histogram source** parameter, you select Specify via dialog, you can use the **Histogram** parameter to enter the desired histogram of the output image. The histogram should be a vector of integer values that represents the desired intensity values in each bin. The block transforms the input image so that the histogram of the output image is approximately the specified histogram.

If, for the **Histogram source** parameter, you select Input port, the Hist port appears on the block. Use this port to specify your desired histogram.

Note The vector input to the Hist port must be normalized such that the sum of the values in all the bins is equal to the number of pixels in the input image. The block does not error if the histogram is not normalized.

Examples   See "Adjust the Contrast of Intensity Images" and "Adjust the Contrast of Color Images" in the *Computer Vision System Toolbox User's Guide*.

Histogram Equalization

Dialog Box

The Histogram Equalization dialog box appears as shown in the following figure.
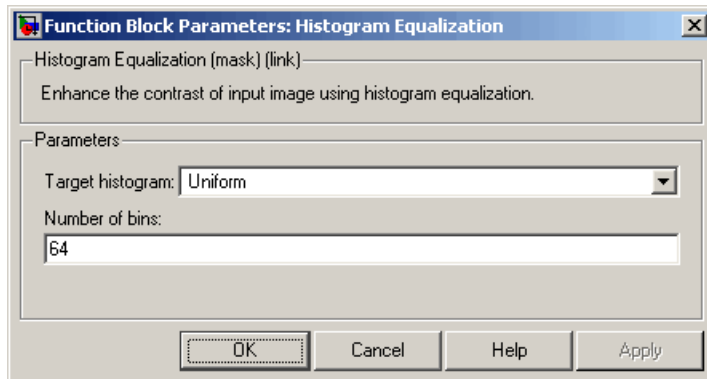
Target histogram

Designate the histogram you want the output image to have. If you select Uniform, the block transforms the input image so that the histogram of the output image is approximately flat. If you select User-defined, you can specify the histogram of your output image.

Number of bins

Enter the number of equally spaced bins you want the uniform histogram to have. This parameter is visible if, for the **Target histogram** parameter, you select Uniform.

Histogram source

Select how to specify your histogram. Your choices are Specify via dialog and Input port. This parameter is visible if, for the **Target histogram** parameter, you select User-defined.

Histogram

Enter the desired histogram of the output image. This parameter is visible if, for the **Target histogram** parameter, you select User-defined.

See Also

imadjust	Image Processing Toolbox
histeq	Image Processing Toolbox

Hough Lines

Purpose   Find Cartesian coordinates of lines described by rho and theta pairs

Library   Transforms

visiontransforms

Description
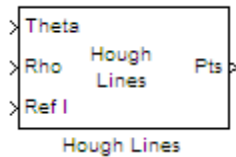
The Hough Lines block finds the points of intersection between the reference image boundary lines and the line specified by a (rho, theta) pair. The block outputs one-based [x y] coordinates for the points of intersection. The boundary lines indicate the left and right vertical boundaries and the top and bottom horizontal boundaries of the reference image.

If the line specified by the (rho, theta) pair does not intersect two border lines in the reference image, the block outputs the values, [(0,0), (0,0)]. This output intersection value allows the next block in your model to ignore the points. Generally, the Hough Lines block precedes a block that draws a point or shape at the intersection.

The following figure shows the input and output coordinates for the Hough Lines block.

$$\begin{bmatrix} theta1 \\ theta2 \end{bmatrix} or \begin{bmatrix} theta1 & theta2 \end{bmatrix}$$

$$\begin{bmatrix} rho1 \\ rho2 \end{bmatrix} or \begin{bmatrix} rho1 & rho2 \end{bmatrix}$$

Reference image

$$\begin{bmatrix} x_{11} & y_{11} & x_{12} \\ x_{21} & y_{21} & x_{22} \end{bmatrix}$$

Port Description

Port	Input/Output	Supported Data Types	Complex Values Supported
Theta	Vector of theta values that represent input lines	• Double-precision floating point • Single-precision floating point • Fixed point (signed, word length less than or equal to 32) • 8-, 16-, and 32-bit signed integer	No
Rho	Vector of rho values that represent input lines	Same as Theta port	No

Port	Input/Output	Supported Data Types	Complex Values Supported
Ref I	Matrix that represents a binary or intensity image or matrix that represents one plane of an RGB image	• Double-precision floating point • Single-precision floating point • Fixed-point (signed and unsigned) • Custom data types • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Pts	*M*-by-4 matrix of intersection values, where *M* is the number of input lines	• 32-bit signed integer	No

Dialog Box

The **Main** pane of the Hough Lines dialog box appears as shown in the following figure.
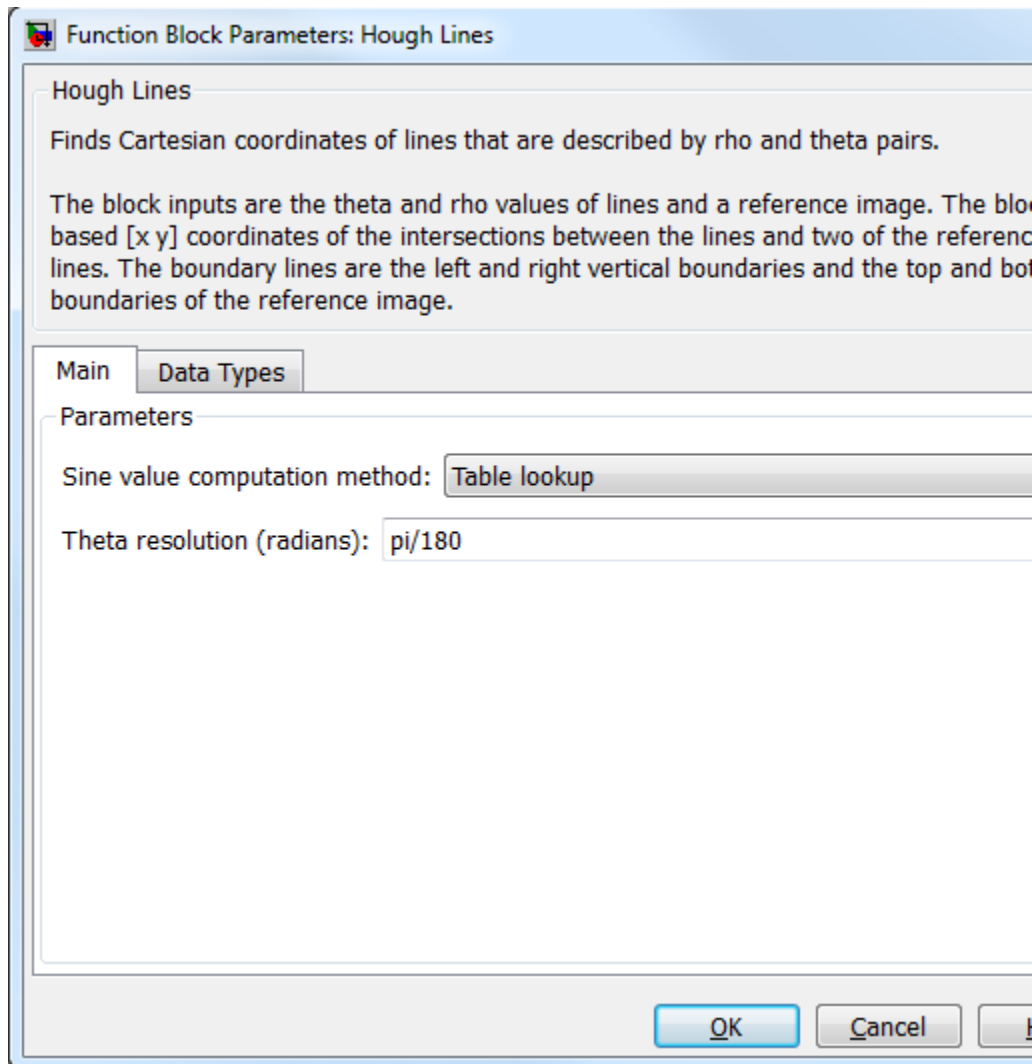
Function Block Parameters: Hough Lines

Hough Lines

Finds Cartesian coordinates of lines that are described by rho and theta pairs.

The block inputs are the theta and rho values of lines and a reference image. The bloc
based [x y] coordinates of the intersections between the lines and two of the referenc
lines. The boundary lines are the left and right vertical boundaries and the top and bot
boundaries of the reference image.

Main | Data Types

Parameters

Sine value computation method: Table lookup

Theta resolution (radians): pi/180

OK | Cancel |

Sine value computation method

If you select `Trigonometric function`, the block computes sine and cosine values to calculate the intersections of the lines during the simulation. If you select `Table lookup`, the block computes and stores the trigonometric values to calculate the intersections of the lines before the simulation starts. In this case, the block requires extra memory.

For floating-point inputs, set the **Sine value computation method** parameter to `Trigonometric function`. For fixed-point inputs, set the parameter to `Table lookup`.

Theta resolution (radians)

Use this parameter to specify the spacing of the theta-axis. This parameter appears in the dialog box only if, for the **Sine value computation method** parameter, you select `Table lookup`. parameter appears in the dialog box.

The **Data Types** pane of the Hough Lines dialog box appears as shown in the following figure.
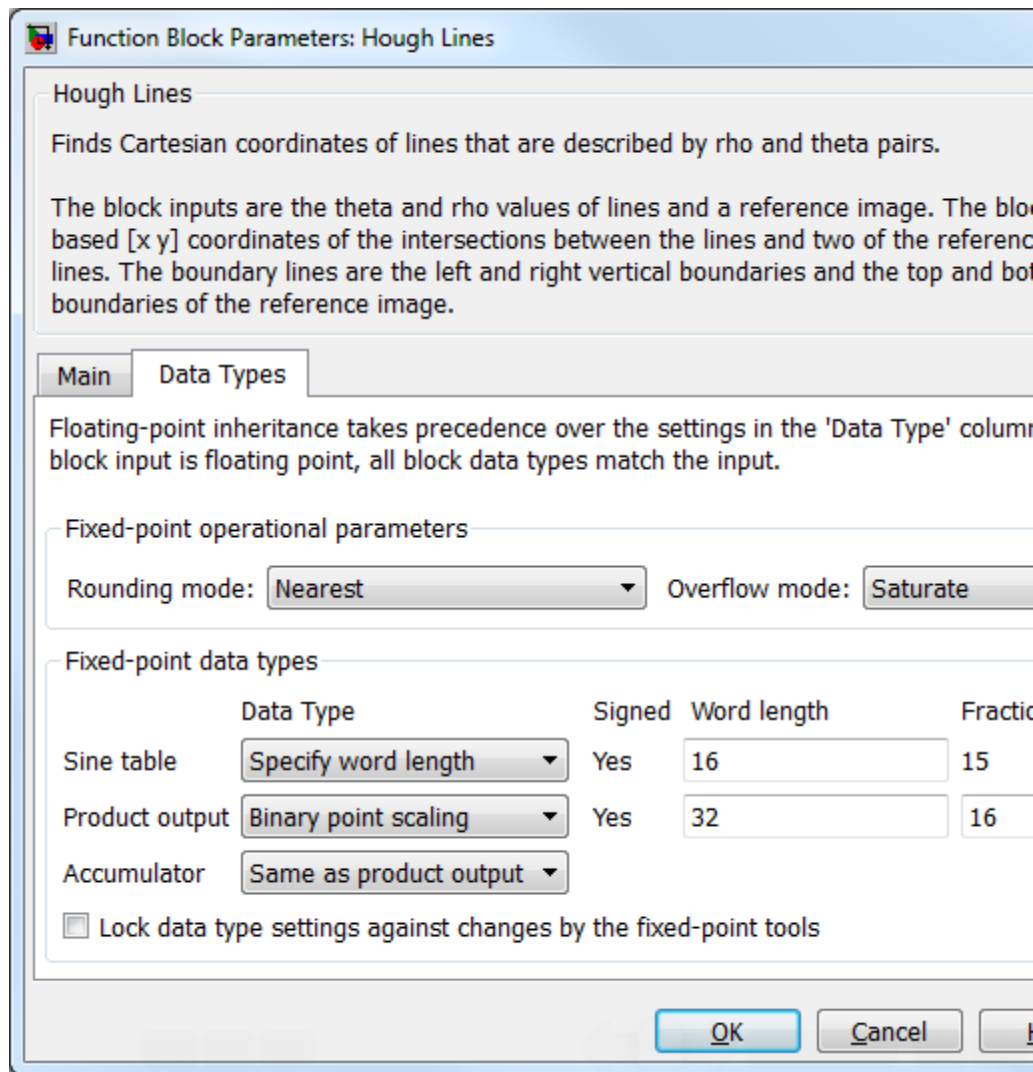
Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Sine table

Choose how to specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one:

When you select `Specify word length`, you can enter the word length of the sine table.

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; they saturate and round to `Nearest`.

Product output

Use this parameter to specify how to designate this product output word and fraction lengths:

When you select `Same as first input`, the characteristics match the characteristics of the first input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. All signals in the Computer Vision System Toolbox blocks have a bias of 0.

See "Multiplication Data Types" for illustrations depicting the use of the product output.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths.

When you select `Same as product output` the characteristics match the characteristics of the product output.

When you select `Binary point scaling`, you can enter the **Word length** and the **Fraction length** of the accumulator, in bits.

When you select `Slope and bias scaling`, you can enter the **Word length**, in bits, and the **Slope** of the **Accumulator**. All signals in the Computer Vision System Toolbox software have a bias of 0.

See "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Examples   The following figure shows Line 1 intersecting the boundaries of the reference image at $[(x_{11}, y_{11}) (x_{12}, y_{12})]$ and Line 2 intersecting the boundaries at $[(x_{21}, y_{21}) (x_{22}, y_{22})]$

Hough Lines
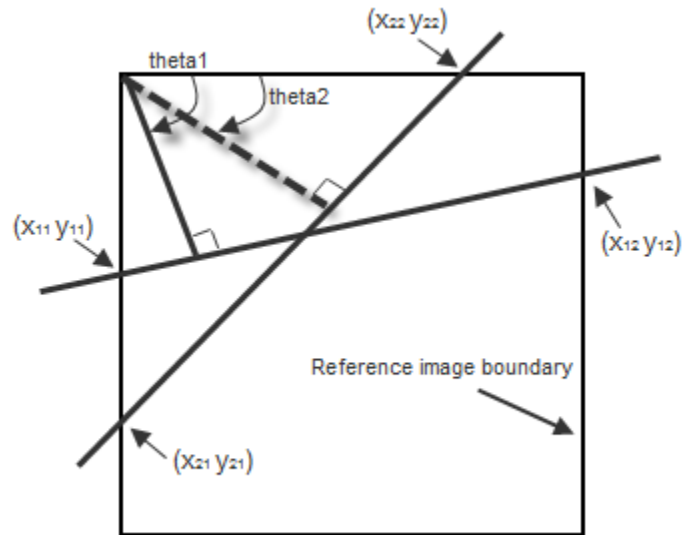
See "Detect Lines in Images" and "Measure Angle Between Lines" in the *Computer Vision System Toolbox User Guide*.

See Also

Find Local Maxima	Computer Vision System Toolbox
Hough Transform	Computer Vision System Toolbox

Purpose   Find Cartesian coordinates of lines described by rho and theta pairs

Library   Transforms

Description

```
  ┌─────────────────────┐
 ›│ Theta               │
  │        Hough        │
 ›│ Rho    Lines    Pts ›│
  │                     │
 ›│ Ref I               │
  └─────────────────────┘
       Hough Lines
```
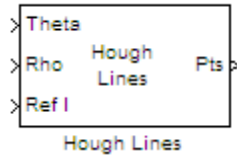
Note This Hough Lines block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Hough Lines block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Hough Lines documentation to view the reference material related to the obsolete block.

Hough Transform

Purpose   Find lines in images

Library   Transforms

                  visiontransforms

Description   Use the Hough Transform block to find lines in an image. The block
                  outputs the Hough space matrix and, optionally, the *rho*-axis and
                  theta-axis vectors. Peak values in the matrix represent potential lines
                  in the input image. Generally, the Hough Transform block precedes the
                  Hough Lines block which uses the output of this block to find lines in an
                  image. You can instead use a custom algorithm to locate peaks in the
                  Hough space matrix in order to identify potential lines.

Port	Input/Output	Supported Data Types	Supported Complex Values
BW	Matrix that represents a binary image	Boolean	No
Hough	Parameter space matrix	• Double-precision floating point • Single-precision floating point • Fixed point (unsigned, fraction length equal to 0) • 8-, 16-, 32-bit unsigned integer	No
Theta	Vector of theta values	• Double-precision floating point • Single-precision floating point • Fixed point (signed) • 8-, 16-, 32-bit signed integer	No
Rho	Vector of rho values	Same as Theta port	No

Dialog Boxes

The **Main** pane of the Hough Transform dialog box appears as shown in the following figure.
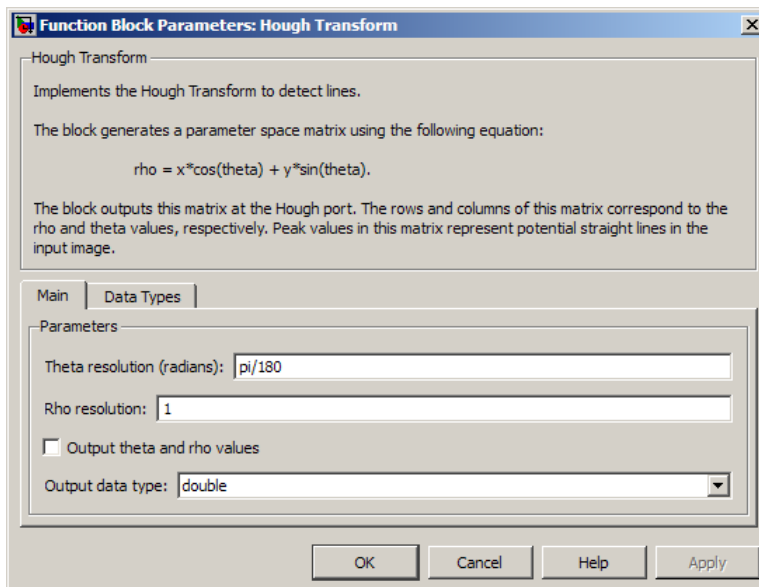
Theta resolution (radians)
 Specify the spacing of the Hough transform bins along the *theta*-axis.

Rho resolution (pixels)
 Specify the spacing of the Hough transform bins along the *rho*-axis.

Output theta and rho values
 If you select this check box, the **Theta** and **Rho** ports appear on the block. The block outputs theta and rho-axis vector values at these ports.

Output data type
 Specify the data type of your output signal.

Hough Transform

The **Data Types** pane of the Hough Transform block dialog appears as shown in the following figure. The Data Types pane will not show fixed-point parameters when **Output data type** parameter is set to `double` or `single`.
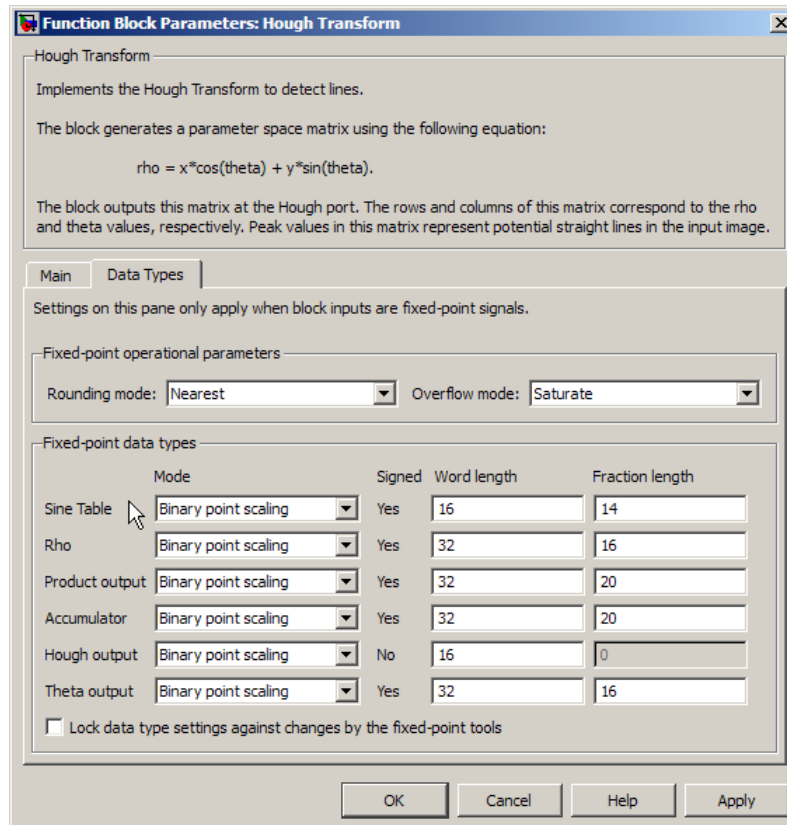
Rounding mode

    Select the rounding mode for fixed-point operations.

Overflow mode

    Select the overflow mode for fixed-point operations.

Sine table

Choose how to specify the word length of the values of the sine table:

- When you select `Binary point scaling`, you can enter the word length of the sine table values, in bits.

- When you select `Slope and bias scaling`, you can enter the word length of the sine table values, in bits.

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; they always saturate and round to `Nearest`.

Rho

Choose how to specify the word length and the fraction length of the rho values:

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the rho values, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the rho values. All signals in Computer Vision System Toolbox blocks have a bias of 0.

Product output

. Use this parameter to specify how to designate the product output word and fraction lengths:

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. All signals in Computer Vision System Toolbox blocks have a bias of 0.

See "Multiplication Data Types" for illustrations depicting the use of the product output.

Hough Transform

Accumulator

Use this parameter to specify how to designate this accumulator word and fraction lengths:

- When you select `Same as product output`, these characteristics match the characteristics of the product output.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. All signals in Computer Vision System Toolbox blocks have a bias of 0.

  See "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Hough output

Choose how to specify the word length and fraction length of the Hough output of the block:

- When you select `Binary point scaling`, you can enter the word length of the Hough output, in bits. The fraction length always has a value of 0.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, of the Hough output. The slope always has a value of 0. All signals in Computer Vision System Toolbox blocks have a bias of 0.

Theta output

Choose how to specify the word length and fraction length of the theta output of the block:

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the theta output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the theta output. All signals in Computer Vision System Toolbox blocks have a bias of 0.

Algorithm
The Hough Transform block implements the Standard Hough Transform (SHT). The SHT uses the parametric representation of a line:

$$rho = x * \cos(theta) + y * \sin(theta)$$
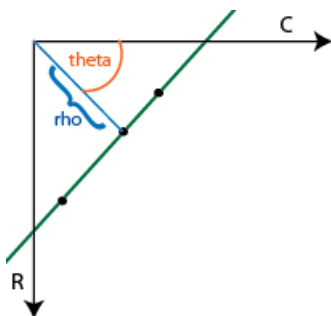
The variable *rho* indicates the perpendicular distance from the origin to the line.

The variable *theta* indicates the angle of inclination of the normal line from the x-axis. The range of *theta* is $-\frac{\pi}{2} \leq \theta < +\frac{\pi}{2}$ with a step-size determined by the **Theta resolution (radians)** parameter. The SHT measures the angle of the line clockwise with respect to the positive x-axis.

The Hough Transform block creates an accumulator matrix. The (*rho, theta*) pair represent the location of a cell in the accumulator matrix. Every valid (logical true) pixel of the input binary image represented by (*R,C*) produces a rho value for all theta values. The block quantizes

Hough Transform

the rho values to the nearest number in the rho vector. The rho vector depends on the size of the input image and the user-specified rho resolution. The block increments a counter (initially set to zero) in those accumulator array cells represented by (*rho, theta*) pairs found for each pixel. This process validates the point (*R,C*) to be on the line defined by (*rho, theta*). The block repeats this process for each logical true pixel in the image. The **Hough** block outputs the resulting accumulator matrix.

Examples See "Detect Lines in Images" and "Measure Angle Between Lines" in the *Computer Vision System Toolbox User Guide*.

See Also

Find Local Maxima	Computer Vision System Toolbox
Hough Lines	Computer Vision System Toolbox
hough	Image Processing Toolbox
houghlines	Image Processing Toolbox
houghpeaks	Image Processing Toolbox

Purpose   Compute complement of pixel values in binary or intensity images

Library   Conversions

visionconversions

Description   The Image Complement block computes the complement of a binary or intensity image. For binary images, the block replaces pixel values equal to 0 with 1 and pixel values equal to 1 with 0. For an intensity image, the block subtracts each pixel value from the maximum value that can be represented by the input data type and outputs the difference.

For example, suppose the input pixel values are given by $x(i)$ and the output pixel values are given by $y(i)$. If the data type of the input is double or single precision floating-point, the block outputs $y(i) = 1.0 - x(i)$. If the input is an 8-bit unsigned integer, the block outputs $y(i) = 255 - x(i)$.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Output	Complement of a binary or intensity	Same as Input port	No

The dimensions, data type, complexity, and frame status of the input and output signals are the same.

Image Complement

Dialog Box

The Image Complement dialog box appears as shown in the following figure.
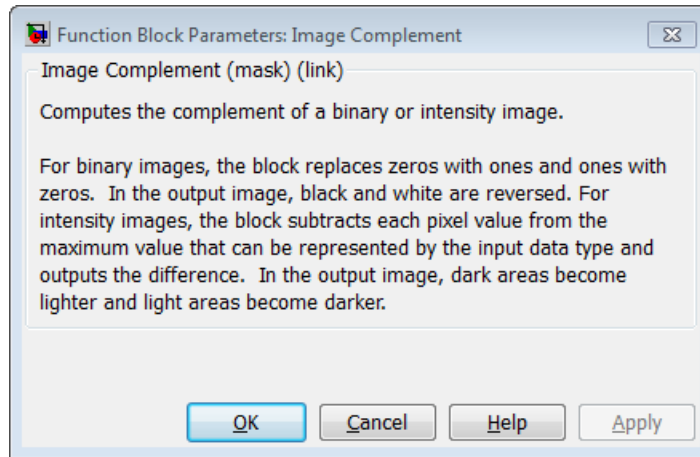
See Also

Autothreshold	Computer Vision System Toolbox software
Chroma Resampling	Computer Vision System Toolbox software
Color Space Conversion	Computer Vision System Toolbox software
imcomplement	Image Processing Toolbox software

Purpose   Convert and scale input image to specified output data type

Library   Conversions

visionconversions

Description   The Image Data Type Conversion block changes the data type of the input to the user-specified data type and scales the values to the new data type's dynamic range. To convert between data types without scaling, use the Simulink Data Type Conversion block.

When converting between floating-point data types, the block casts the input into the output data type and clips values outside the range to 0 or 1. When converting to the Boolean data type, the block maps 0 values to 0 and all other values to one. When converting to or between all other data types, the block casts the input into the output data type and scales the data type values into the dynamic range of the output data type. For double- and single-precision floating-point data types, the dynamic range is between 0 and 1. For fixed-point data types, the dynamic range is between the minimum and maximum values that can be represented by the data type.

Note This block supports intensity and color images on its ports.

Port	Input/Output	Supported Data Types	Complex Values Supported
Input	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point (word length less than or equal to 16) • Boolean • 8-, 16-bit signed integer	No

Image Data Type Conversion

Port	Input/Output	Supported Data Types	Complex Values Supported
		• 8-, 16-bit unsigned integer	
Output	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	Same as Input port	No

The dimensions, complexity, and frame status of the input and output signals are the same.

Use the **Output data type** parameter to specify the data type of your output signal values.

Dialog Box

The Image Data Type Conversion dialog box appears as shown in the following figure.
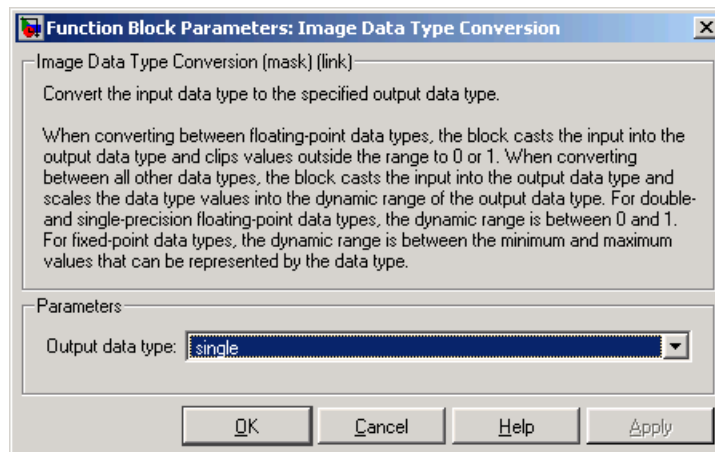
Output data type

Use this parameter to specify the data type of your output signal.

Signed

Select this check box if you want the output fixed-point data to be signed. This parameter is visible if, for the **Output data type** parameter, you choose Fixed-point.

Word length

Use this parameter to specify the word length of your fixed-point output. This parameter is visible if, for the **Output data type** parameter, you choose Fixed-point.

Fraction length

Use this parameter to specify the fraction length of your fixed-point output. This parameter is visible if, for the **Output data type** parameter, you choose Fixed-point.

See Also   Autothreshold   Computer Vision System Toolbox software

Image From File

Purpose   Import image from image file

Library   Sources

visionsources
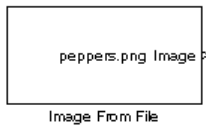
Description   Use the Image From File block to import an image from a supported image file. For a list of supported file formats, see the imread function reference page in the MATLAB documentation. If the image is a M-by-N array, the block outputs a binary or intensity image, where M and N are the number of rows and columns in the image. If the image is a M-by-N-by-P array, the block outputs a color image, where M and N are the number of rows and columns in each color plane, P.

Port	Output	Supported Data Types	Complex Values Supported
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	Yes
R, G, B	Scalar, vector, or matrix that represents one plane of the input RGB video stream. Outputs from the R, G, or B ports have the same dimensions.	Same as I port	Yes

For the Computer Vision System Toolbox blocks to display video data properly, double- and single-precision floating-point pixel values must be between 0 and 1. If the input pixel values have a different data type than the one you select using the **Output data type** parameter, the

block scales the pixel values, adds an offset to the pixel values so that they are within the dynamic range of their new data type, or both.

Use the **File name** parameter to specify the name of the graphics file that contains the image to import into the Simulink modeling and simulation software. If the file is not on the MATLAB path, use the **Browse** button to locate the file. This parameter supports URL paths.

Use the **Sample time** parameter to set the sample period of the output signal.

Use the **Image signal** parameter to specify how the block outputs a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Use the **Output port labels** parameter to label your output ports. Use the spacer character, |, as the delimiter. This parameter is visible if you set the **Image signal** parameter to Separate color signals.

On the **Data Types** pane, use the **Output data type** parameter to specify the data type of your output signal.

Image From File

Dialog Box

The **Main** pane of the Image From File dialog box appears as shown in the following figure.
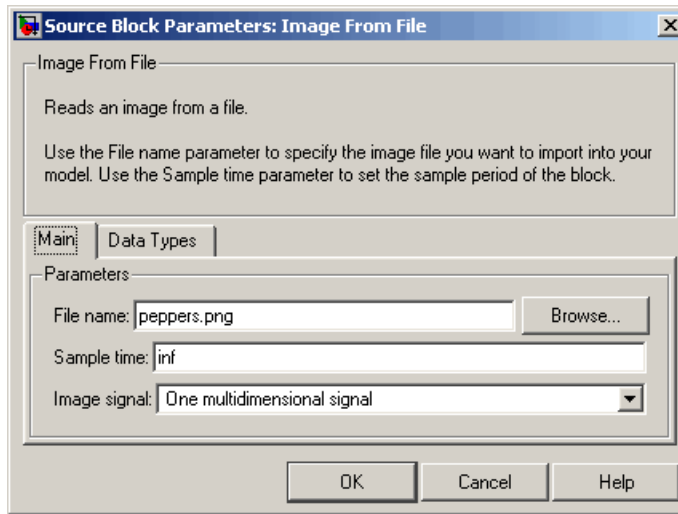
File name

Specify the name of the graphics file that contains the image to import into the Simulink environment.

Sample time

Enter the sample period of the output signal.

Image signal

Specify how the block outputs a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Output port labels

Enter the labels for your output ports using the spacer character, |, as the delimiter. This parameter is visible if you set the **Image signal** parameter to Separate color signals.

The **Data Types** pane of the Image From File dialog box appears as shown in the following figure.

Output data type

Specify the data type of your output signal.

Signed

Select to output a signed fixed-point signal. Otherwise, the signal will be unsigned. This parameter is only visible if, from the **Output data type** list, you select Fixed-point.

Image From File

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible if, from the **Output data type** list, you select `Fixed-point`.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible if, from the **Output data type** list, you select `Fixed-point` or when you select `User-defined`.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac` functions from the Fixed-Point Designer™ library. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

See Also

From Multimedia File	Computer Vision System Toolbox software
Image From Workspace	Computer Vision System Toolbox software

To Video Display	Video and Image Processing Blockset software
Video From Workspace	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software
im2double	Image Processing Toolbox software
im2uint8	Image Processing Toolbox software
imread	MATLAB

Image From Workspace

Purpose   Import image from MATLAB workspace

Library   Sources

visionsources

Description   Use the Image From Workspace block to import an image from the
MATLAB workspace. If the image is a M-by-N workspace array,
the block outputs a binary or intensity image, where M and N are
the number of rows and columns in the image. If the image is a
M-by-N-by-P workspace array, the block outputs a color image, where
M and N are the number of rows and columns in each color plane, P.

checker_board(10) Image ▶

Image From Workspace

Port	Output	Supported Data Types	Complex Values Supported
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
R, G, B	Scalar, vector, or matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports have the same dimensions.	Same as I port	No

For the Computer Vision System Toolbox blocks to display video data
properly, double- and single-precision floating-point pixel values must
be between 0 and 1. If the input pixel values have a different data type
than the one you select using the **Output data type** parameter, the

block scales the pixel values, adds an offset to the pixel values so that they are within the dynamic range of their new data type, or both.

Use the **Value** parameter to specify the MATLAB workspace variable that contains the image you want to import into Simulink environment.

Use the **Sample time** parameter to set the sample period of the output signal.

Use the **Image signal** parameter to specify how the block outputs a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Use the **Output port labels** parameter to label your output ports. Use the spacer character, |, as the delimiter. This parameter is visible if you set the **Image signal** parameter to Separate color signals.

On the **Data Types** pane, use the **Output data type** parameter to specify the data type of your output signal.

Image From Workspace

Dialog Box

The **Main** pane of the Image From Workspace dialog box appears as shown in the following figure.
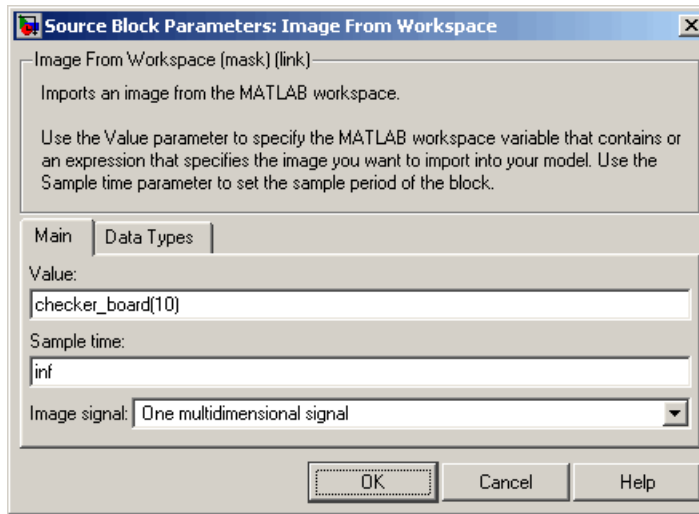
Value

Specify the MATLAB workspace variable that you want to import into Simulink environment.

Sample time

Enter the sample period of the output signal.

Image signal

Specify how the block outputs a color video signal. If you select One multidimensional signal, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Output port labels

Enter the labels for your output ports using the spacer character, |, as the delimiter. This parameter is visible if you set the **Image signal** parameter to Separate color signals.

The **Data Types** pane of the Image From Workspace dialog box appears as shown in the following figure.

Output data type

Specify the data type of your output signal.

Signed

Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible if, from the **Output data type** list, you select Fixed-point.

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible if, from the **Output data type** list, you select Fixed-point.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose Best precision to have the output scaling automatically set such that the output signal has the best possible precision.

- Choose User-defined to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible if, from the **Output data type** list, you select Fixed-point or when you select User-defined.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select Fixed-point or User-defined for the **Output data type** parameter and User-defined for the **Set fraction length in output to** parameter.

User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the sfix, ufix, sint, uint, sfrac, and ufrac functions from the Fixed-Point Designer library. This parameter is only visible when you select User-defined for the **Output data type** parameter.

See Also

| From Multimedia File | Computer Vision System Toolbox software |
| To Video Display | Computer Vision System Toolbox software |

Video From Workspace	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software
im2double	Image Processing Toolbox software
im2uint8	Image Processing Toolbox software

Image Pad

Purpose   Pad signal along its rows, columns, or both

Library   Utilities

visionutilities

Description The Image Pad block expands the dimensions of a signal by padding its
rows, columns, or both. To crop an image, you can use the Simulink
Selector block, DSP System Toolbox Submatrix block, or the Image
Processing Toolbox `imcrop` function.

Port	Input/Output	Supported Data Types	Complex Values Supported
Image / I	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color video signal, where *P* is the number of color planes.	• Double-precision floating point. • Single-precision floating point. • Fixed point. • Boolean. • 8-, 16-, 32-bit signed integer. • 8-, 16-, 32-bit unsigned integer.	Yes
PVal	Scalar value that represents the constant pad value.	Same as I port.	Yes
Output	Padded scalar, vector, or matrix.	Same as I port.	Yes

Examples

Pad with a Constant Value

Suppose you want to pad the rows of your input signal with three initial values equal to 0 and your input signal is defined as follows:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Set the Image Pad block parameters as follows:

- **Method** = Constant
- **Pad value source** = Specify via dialog
- **Pad value** = 0
- **Specify** = Output size
- **Add columns to** = Left
- **Output row mode** = User-specified
- **Number of output columns** = 6
- **Add rows to** = No padding

The Image Pad block outputs the following signal:

$$\begin{bmatrix} 0 & 0 & 0 & a_{00} & a_{01} & a_{02} \\ 0 & 0 & 0 & a_{10} & a_{11} & a_{12} \\ 0 & 0 & 0 & a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Pad by Repeating Border Values

Suppose you want to pad your input signal with its border values, and your input signal is defined as follows:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Set the Image Pad block parameters as follows:

- **Method** = Replicate
- **Specify** = Pad size
- **Add columns to** = Both left and right
- **Number of added columns** = 2
- **Add rows to** = Both top and bottom
- **Number of added rows** = [1 3]

The Image Pad block outputs the following signal:

$$
\begin{bmatrix}
a_{00} & a_{00} & a_{00} & a_{01} & a_{02} & a_{02} & a_{02} \\
a_{00} & a_{00} & a_{00} & a_{01} & a_{02} & a_{02} & a_{02} \\
a_{10} & a_{10} & a_{10} & a_{11} & a_{12} & a_{12} & a_{12} \\
a_{20} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{22} \\
a_{20} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{22} \\
a_{20} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{22} \\
a_{20} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{22}
\end{bmatrix}
\quad \text{Input matrix}
$$

The border values of the input signal are replicated on the top, bottom, left, and right of the input signal so that the output is a 7-by-7 matrix. The values in the corners of this output matrix are determined by replicating the border values of the matrices on the top, bottom, left and right side of the original input signal.

Pad with Mirror Image

Suppose you want to pad your input signal using its mirror image, and your input signal is defined as follows:

$$
\begin{bmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22}
\end{bmatrix}
$$

Set the Image Pad block parameters as follows:

- **Method** = Symmetric
- **Specify** = Pad size
- **Add columns to** = Both left and right

Image Pad

- **Number of added columns** = [5 6]
- **Add rows to** = Both top and bottom
- **Number of added rows** = 2

The Image Pad block outputs the following signal:

$$
\begin{bmatrix}
a_{11} & a_{12} & a_{12} & a_{11} & a_{10} & a_{10} & a_{11} & a_{12} & a_{12} & a_{11} & a_{10} & a_{10} & a_{11} & a_{12} \\
a_{01} & a_{02} & a_{02} & a_{01} & a_{00} & a_{00} & a_{01} & a_{02} & a_{02} & a_{01} & a_{00} & a_{00} & a_{01} & a_{02} \\
a_{01} & a_{02} & a_{02} & a_{01} & a_{00} & a_{00} & a_{01} & a_{02} & a_{02} & a_{01} & a_{00} & a_{00} & a_{01} & a_{02} \\
a_{11} & a_{12} & a_{12} & a_{11} & a_{01} & a_{10} & a_{11} & a_{12} & a_{12} & a_{11} & a_{10} & a_{10} & a_{11} & a_{12} \\
a_{21} & a_{22} & a_{22} & a_{21} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{21} & a_{20} & a_{20} & a_{21} & a_{22} \\
a_{21} & a_{22} & a_{22} & a_{21} & a_{20} & a_{20} & a_{21} & a_{22} & a_{22} & a_{21} & a_{20} & a_{20} & a_{21} & a_{22} \\
a_{11} & a_{12} & a_{12} & a_{11} & a_{01} & a_{01} & a_{11} & a_{12} & a_{12} & a_{11} & a_{10} & a_{10} & a_{11} & a_{12}
\end{bmatrix}
\longleftarrow \text{Input matrix}
$$

The block flips the original input matrix and each matrix it creates about their top, bottom, left, and right sides to populate the 7-by-13 output signal. For example, in the preceding figure, you can see how the block flips the input matrix about its right side to create the matrix directly to its right.

Pad Using a Circular Repetition of Elements

Suppose you want to pad your input signal using a circular repetition of its values. Your input signal is defined as follows:

$$
\begin{bmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22}
\end{bmatrix}
$$

Set the Image Pad block parameters as follows:

- **Method** = Circular
- **Specify** = Output size
- **Add columns to** = Both left and right
- **Number of output columns** = 9
- **Add rows to** = Both top and bottom
- **Number of output rows** = 9

The Image Pad block outputs the following signal:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} \\ a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} \\ a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} & a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} & a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} & a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Input matrix

The block repeats the values of the input signal in a circular pattern to populate the 9-by-9 output matrix.

Image Pad

Dialog Box

The Image Pad dialog box appears as shown in the following figure.

Function Block Parameters: Image Pad

Image Pad (mask) (link)

Pad or crop a two-dimensional input image

Pad your image by adding rows, columns, or rows and columns. You can pad your image by using a constant value, repeating its border values, using its mirror image, or using circular repetition of its elements.

You can crop your image by specifying output dimensions that are smaller than the corresponding input dimensions.

Parameters

Method: Constant

Pad value source: Specify via dialog

Pad value:

0

Specify: Pad size

Add columns to: Both left and right

Number of added columns:

2

Add rows to: Both top and bottom

Number of added rows:

[2 3]

OK   Cancel   Help   Apply

Method

Specify how you want the block to pad your signal. The data type of the input signal is the data type of the output signal.

Use the **Method** parameter to specify how you pad the input signal.

- Constant — Pad with a constant value

- Replicate — Pad by repeating its border values

- Symmetric — Pad with its mirror image

- Circular — Pad using a circular repetition of its elements

If you set the **Method** parameter to Constant, the **Pad value source** parameter appears on the dialog box.

- Input port — The PVal port appears on the block. Use this port to specify the constant value with which to pad your signal

- Specify via dialog — The **Pad value** parameter appears in the dialog box. Enter the constant value with which to pad your signal.

Pad value source

If you select Input port, the PVal port appears on the block. Use this port to specify the constant value with which to pad your signal. If you select Specify via dialog, the **Pad value** parameter becomes available. This parameter is visible if, for the **Method** parameter, you select Constant.

Pad value

Enter the constant value with which to pad your signal. This parameter is visible if, for the **Pad value source** parameter, you select Specify via dialog. This parameter is tunable.

Specify

If you select Pad size, you can enter the size of the padding in the horizontal and vertical directions.

Image Pad

If you select `Output size`, you can enter the total number of output columns and rows. This setting enables you to pad the input signal. See the previous section for descriptions of the **Add columns to** and **Add rows to** parameters.

Add columns to

The **Add columns to** parameter controls the padding at the left, right or both sides of the input signal.

- `Left` — The block adds additional columns on the left side.

- `Right` — The block adds additional columns on the right side.

- `Both left and right` — The block adds additional columns to the left and right side.

- `No padding` — The block does not change the number of columns.

Use the **Add columns to** and **Number of added columns** parameters to specify the size of the padding in the horizontal direction. Enter a scalar value, and the block adds this number of columns to the left, right, or both sides of your input signal. If you set the **Add columns to** parameter to `Both left and right`, you can enter a two element vector. The left element controls the number of columns the block adds to the left side of the signal; the right element controls the number of columns the block adds to the right side of the signal.

Output row mode

Use the **Output row mode** parameter to describe how to pad the input signal.

- `User-specified` — Use the **Number of output rows** parameter to specify the total number of rows.

- `Next power of two` — The block pads the input signal along the rows until the length of the rows is equal to a power of two. When the length of the input signal's rows is equal to a power of two, the block does not pad the input signal's rows.

Number of added columns

This parameter controls how many columns are added to the right and/or left side of your input signal. Enter a scalar value, and the block adds this number of columns to the left, right, or both sides of your signal. If, for the **Add columns to** parameter you select `Both left and right`, enter a two-element vector. The left element controls the number of columns the block adds to the left side of the signal and the right element controls how many columns the block adds to the right side of the signal. This parameter is visible if, for the **Specify** parameter, you select `Pad size`.

Add rows to

The **Add rows to** parameter controls the padding at the top and bottom of the input signal.

- `Top` — The block adds additional rows to the top.

- `Bottom` — The block adds additional rows to the bottom.

- `Both top and bottom` — The block adds additional rows to the top and bottom.

- `No padding` — The block does not change the number of rows.

Use the **Add rows to** and **Number of added rows** parameters to specify the size of the padding in the vertical direction. Enter a scalar value, and the block adds this number of rows to the top, bottom, or both of your input signal. If you set the **Add rows to** parameter to `Both top and bottom`, you can enter a two element vector. The left element controls the number of rows the block adds to the top of the signal; the right element controls the number of rows the block adds to the bottom of the signal.

Output column mode

Describe how to pad the input signal. If you select `User-specified`, the Row size parameter appears on the block dialog box. If you select Next power of two, the block pads the input signal along the rows until the length of the rows is equal

to a power of two. This parameter is visible if, for the Specify parameter, you select Output size.

Use the **Output column mode** parameter to describe how to pad the input signal.

- `User-specified` — Use the **Number of column rows** parameter to specify the total number of columns.

- `Next power of two` — The block pads the input signal along the columns until the length of the columns is equal to a power of two. When the length of the input signal's columns is equal to a power of two, the block does not pad the input signal's columns.

Number of added rows

This parameter controls how many rows are added to the top, bottom, or both of your input signal. Enter a scalar value and the block adds this number of columns to the top, bottom, or both of your signal. If, for the **Add rows to** parameter you select `Both top and bottom`, enter a two-element vector. The left element controls the number of rows the block adds to the top of the signal and the right element controls how many rows the block adds to the bottom of the signal. This parameter is visible if you set the **Specify** parameter to `Pad size`.

Action when truncation occurs

The following options are available for the **Action when truncation occurs** parameter:

- `None` — Select this option when you do not want to be notified that the input signal is truncated.

- `Warning` — Select this option when you want to receive a warning in the MATLAB Command Window when the input signal is truncated.

- `Error` — Select this option when you want an error dialog box displayed and the simulation terminated when the input signal is truncated.

See Also   Selector | Submatrix | imcrop

Insert Text

Purpose   Draw text on image or video stream.

Library   Text & Graphics

visiontextngfix

Description   The Insert Text block draws formatted text or numbers on an image or
video stream. The block uses the FreeType 2.3.5 library, an open-source
font engine, to produce stylized text bitmaps. To learn more about the
FreeType Project, visit http://www.freetype.org/. The Insert Text
block does not support character sets other than ASCII.

The Insert Text block lets you draw one or more instances of one or
more strings, including:

- A single instance of one text string

- Multiple instances of one text string

- Multiple instances of text, with a different text string at each location

Port Description

Port	Description	Supported Data Types
Image	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color video signal where *P* represents the number of color planes.	• Double-precision floating point • Single-precision floating point • Fixed point(signed, word length less than or equal to 32.) • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer
R, G, B	Matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports	Same as Input port

Port	Description	Supported Data Types
	have the same dimensions and data type.	
Select	One-based index value that indicates which text string to display.	• Double-precision floating point. (This data type is only supported if the input to the I or R, G, and B ports is a floating-point data type.) • Single-precision floating point. (This data type is only supported if the input to the I or R, G, and B ports is a floating-point data type.) • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer
Variables	Vector or matrix whose values are used to replace ANSI C printf-style format specifications.	The data types supported by this port depend on the conversion specification you are using in the **Text** parameter. %d, %i, and %u: • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer %c and %s: • 8-bit unsigned integer %f: • Double-precision floating point

Port	Description	Supported Data Types
		• Single-precision floating point %o, %x, %X, %e, %E, %g, and %G: • Double-precision floating point • Single-precision floating point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer
Color	Intensity input — Scalar value used for all strings or a vector of intensity values whose length is equal to the number of strings. Color input — Three-element vector that specifies one color for all strings or a M-by-3 matrix of color values, where M represents the number of strings.	Same as Input port (The input to this port must be the same data type as the input to the Input port.)

Port	Description	Supported Data Types
Location	M-by-2 matrix of one-based [x y] coordinates, where M represents the number of text strings to insert. **Location** specifies the top-left corner of the text string bounding box.	• Double-precision floating point. (This data type is only supported if the input to the I or R, G, and B ports is a floating-point data type.) • Single-precision floating point. (This data type is only supported if the input to the I or R, G, and B ports is a floating-point data type.) • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer
Opacity	Scalar value that is used for all strings or vector of opacity values whose length is equal to the number of strings.	• Double-precision floating point. (This data type is only supported if the input to the Input or R, G, and B ports is a double-precision floating-point data type.) • Single-precision floating point. (This data type is only supported if the input to the I or R, G, and B ports is a single-precision floating-point data type.) • ufix8_En7 (This data type is only supported if the input to the I or R, G, and B ports is a fixed-point data type.)

Row-Major Data Format

MATLAB and the Computer Vision System Toolbox blocks use column-major data organization. However, the Insert Text block gives

you the option to process data that is stored in row-major format. When you select the **Input image is transposed (data order is row major)** check box, the block assumes that the input buffer contains contiguous data elements from the first row first, then data elements from the second row second, and so on through the last row. Use this functionality only when you meet all the following criteria:

- You are developing algorithms to run on an embedded target that uses the row-major format.

- You want to limit the additional processing required to take the transpose of signals at the interfaces of the row-major and column-major systems.

When you use the row-major functionality, you must consider the following issues:

- When you select this check box, the first two signal dimensions of the Insert Text block's input are swapped.

- All Computer Vision System Toolbox software blocks can be used to process data that is in the row-major format, but you need to know the image dimensions when you develop your algorithms.

  For example, if you use the 2-D FIR Filter block, you need to verify that your filter coefficients are transposed. If you are using the Rotate block, you need to use negative rotation angles, etc.

- Only three blocks have the **Input image is transposed (data order is row major)** check box. They are the Chroma Resampling, Deinterlacing, and Insert Text blocks. You need to select this check box to enable row-major functionality in these blocks. All other blocks must be properly configured to process data in row-major format.

Use the following two-step workflow to develop algorithms in row-major format to run on an embedded target.

Step 1:
Create block diagram

Algorithm blocks

Video source block

Transpose block

Transpose block

Video sink block

Embedded target source block

Step 2:
Replace source, transpose, and sink blocks with target source and sink blocks that produce data in row-major format

Embedded target sink block
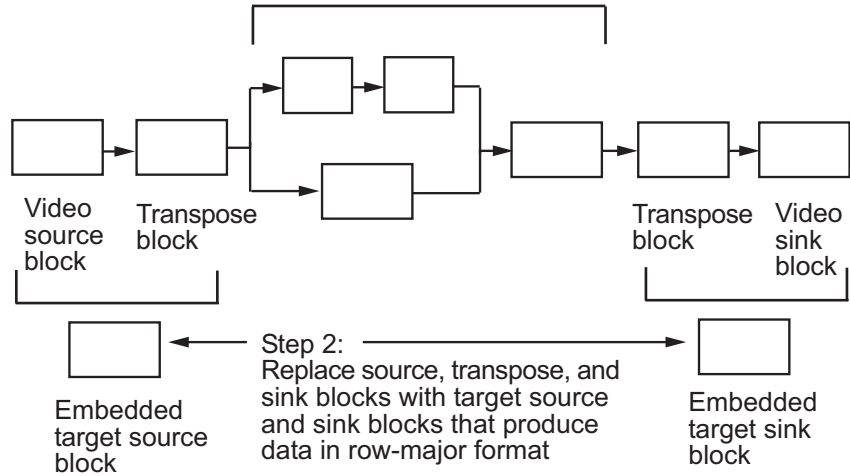
See the DM642 EVM Video ADC and DM642 EVM Video DAC reference pages.

Insert Text

Dialog Box

The **Main** pane of the Insert Text dialog box appears as shown in the following figure.
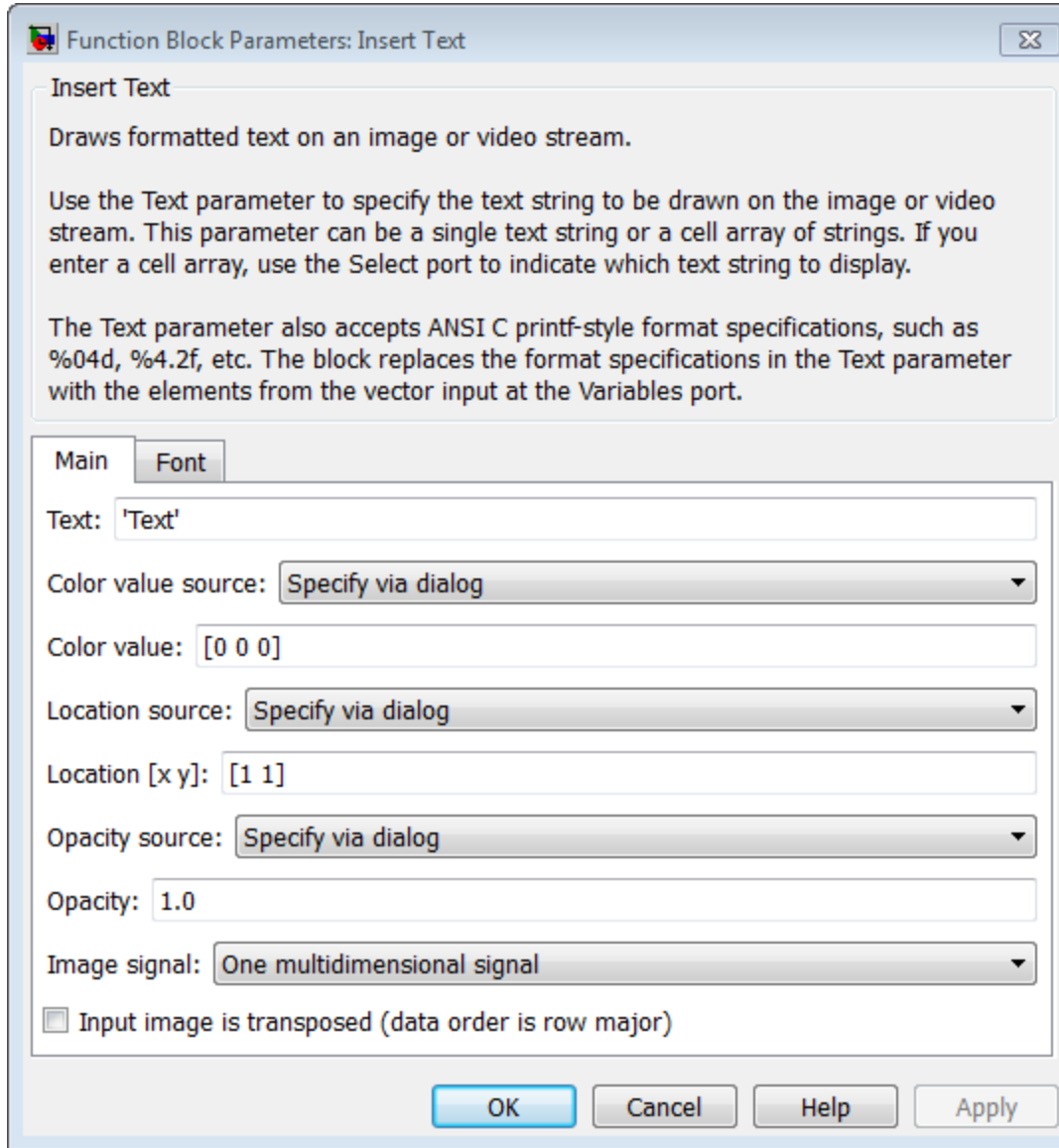
Text

Specify the text string to be drawn on the image or video stream. This parameter can be a single text string, such as `'Figure1'`, a cell array of strings, such as `{'Figure1','Figure2'}`, or an ANSI C `printf`-style format specifications, such as %s.. To create a **Select** port enter a cell array of strings. To create a **Variables** port, enter ANSI C printf-style format specifications, such as `%d`, `%f`, or `%s`.

When you enter a cell array of strings, the Insert Text block does not display all of the strings simultaneously. Instead, the **Select** port appears on the block to let you indicate which text string to display. The input to this port must be a scalar value, where `1` indicates the first string. If the input is less than `1` or greater than one less than the number of strings in the cell array, no text will be drawn on the image or video frame.

When you enter ANSI C `printf`-style format specifications, such as `%d`, `%f`, or `%s`, the **Variables** port appears on the block. The block replaces the format specifications in the **Text** parameter with each element of the input vector . Use the `%s` option to specify a set of text strings for the block to display simultaneously at different locations. For example, using a Constant block, enter `[uint8('Text1') 0 uint8('Text2')]` for the **Constant value** parameter. The following table summarizes the supported conversion specifications.

Text Parameter Supported Conversion Specifications

Supported specifications	Support for multiple instances of the same specification	Support for mixed specifications
%d, %i, %u, %c, %f, %o, %x, %X, %e, %E, %g, and %G	Yes	No
%s	No	No

Color value source

Select where to specify the text color. Your choices are:

- Specify via dialog — the **Color value** parameter appears on the dialog box.

- Input port — the Color port appears on the block.

Color value

Specify the intensity or color of the text. This parameter is visible if, for the **Color source** parameter, you select Specify via dialog. Tunable.

The following table describes how to format the color of the text strings, which depend on the block input and the number of strings you want to insert. Color values for a floating-point data type input image must be between 0 and 1. Color values for an 8-bit unsigned integer data type input image must between 0 and 255.

Text String Color Values

Block Input	One Text String	Multiple Text Strings
Intensity image	**Color value** parameter or the input to the **Color** port specified as a scalar intensity value	**Color value** parameter or the input to the **Color** port specified as a vector of intensity values whose length is equal to the number of strings
Color image	**Color value** parameter or the input to the **Color** port specified as an RGB triplet that defines the color of the text	**Color value** parameter or the input to the **Color** port specified as an M-by-3 matrix of color values, where M represents the number of strings

Location source

Indicate where you want to specify the text location. Your choices are:

- Specify via dialog — the **Location [x y]** parameter appears on the dialog box.
- Input port — the Location port appears on the block.

Location [x y]

Specify the text location. This parameter is visible if, for the **Location source** parameter, you select Specify via dialog. Tunable.

The following table describes how to format the location of the text strings depending on the number of strings you specify to

insert. You can specify more than one location regardless of how many text strings you specify, but the only way to get a different text string at each location is to use the %s option for the **Text** parameter to specify a set of text strings. You can enter negative values or values that exceed the dimensions of the input image or video frame, but the text might not be visible.

Location Parameter Text String Insertion

Parameter	One Instance of One Text String	Multiple Instances of the Same Text String	Multiple Instances of Unique Text Strings
Location [x y] parameter setting or the input to the Location port	Two-element vector of the form [x y] that indicates the top-left corner of the text bounding box.	*M*-by-2 matrix, where *M* represents the number of locations at which to display the text string. Each row contains the coordinates of the top-left corner of the text bounding box for the string, e.g., [x_1 y_1; x_2 y_2]	*M*-by-2 matrix, where *M* represents the number of text strings. Each row contains the coordinates of the top-left corner of the text bounding box for the string, e.g., [x_1 y_1; x_2 y_2].

Opacity source

Indicate where you want to specify the text's opaqueness. Your choices are:

- `Specify via dialog` — the **Opacity** parameter appears on the dialog box.

- `Input port` — the Opacity port appears on the block.

Opacity

Specify the opacity of the text. This parameter is visible if, for the **Opacity source** parameter, you select `Specify via dialog`. Tunable.

The following table describes how to format the opacity of the text strings depending on the number of strings you want to insert.

Text String Opacity Values

Parameter	One Text String	Multiple Text Strings
Opacity parameter setting or the input to the Opacity port	Scalar value between 0 and 1, where 0 is translucent and 1 is opaque	Vector whose length is equal to the number of strings

Use the **Image signal** parameter to specify how to input and output a color video signal:

- One multidimensional signal — the block accepts an *M*-by-*N*-by-*P* color video signal, where P is the number of color planes, at one port.

- Separate color signals — additional ports appear on the block. Each port accepts one *M*-by-*N* plane of an RGB video stream.

Image signal

Specify how to input and output a color video signal. If you select One multidimensional signal, the block accepts an *M*-by-*N*-by-*P* color video signal, where *P* is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port accepts one *M*-by-*N* plane of an RGB video stream.

Input image is transposed (data order is row major)

When you select this check box, the block assumes that the input buffer contains data elements from the first row first, then data elements from the second row second, and so on through the last row.

Insert Text

The **Font** pane of the Insert Text dialog box appears as shown in the following figure.

Function Block Parameters: Insert Text

Insert Text

Draws formatted text on an image or video stream.

Use the Text parameter to specify the text string to be drawn on the image or video stream. This parameter can be a single text string or a cell array of strings. If you enter a cell array, use the Select port to indicate which text string to display.

The Text parameter also accepts ANSI C printf-style format specifications, such as %04d, %4.2f, etc. The block replaces the format specifications in the Text parameter with the elements from the vector input at the Variables port.

| Main | Font |

Font face: LucidaSansRegular

Font size (points): 12

☑ Anti-aliased

OK   Cancel   Help   Apply

Insert Text

Font face
> Specify the font of your text. The block populates this list with the fonts installed on your system. On Windows, the block searches the system registry for font files. On UNIX, the block searches the X Server's font path for font files.

Font size (points)
> Specify the font size.

Anti-aliased
> Select this check box if you want the block to smooth the edges of the text. This can be computationally expensive. If you want your model to run faster, clear this check box.

Examples
• "Annotate Video Files with Frame Numbers"

See Also

Draw Shapes	Computer Vision System Toolbox
Draw Markers	Computer Vision System Toolbox

Purpose   Draw text on image or video stream.

Library   Text & Graphics

Description

Insert Text

Note This Insert Text block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Insert Text block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.

See the Insert Text documentation to view the reference material related to the obsolete block.

Label

Purpose	Label connected components in binary images
Library	Morphological Operations
	`visionmorphops`
Description	The Label block labels the objects in a binary image, BW, where the background is represented by pixels equal to 0 (black) and objects are represented by pixels equal to 1 (white). At the Label port, the block outputs a label matrix that is the same size as the input matrix. In the label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. At the Count port, the block outputs a scalar value that represents the number of labeled objects.

Port	Input/Output	Supported Data Types	Complex Values Supported
BW	Vector or matrix that represents a binary image	Boolean	No
Label	Label matrix	• 8-, 16-, and 32-bit unsigned integer	No
Count	Scalar that represents the number of labeled objects	Same as Label port	No

Use the **Connectivity** parameter to define which pixels are connected to each other. If you want a pixel to be connected to the other pixels located on the top, bottom, left, and right, select 4. If you want a pixel to be connected to the other pixels on the top, bottom, left, right, and diagonally, select 8.

Consider the following 3-by-3 image. If, for the **Connectivity** parameter, you select 4, the block considers the white pixels marked by black circles to be connected.
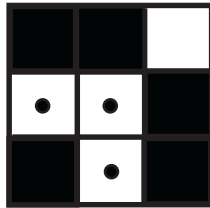
If, for the **Connectivity** parameter, you select 8, the block considers the white pixels marked by black circles to be connected.
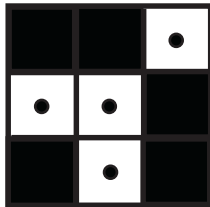
Use the **Output** parameter to determine the block's output. If you select Label matrix and number of labels, ports Label and Count appear on the block. The block outputs the label matrix at the Label port and the number of labeled objects at the Count port. If you select Label matrix, the Label port appears on the block. If you select Number of labels, the Count port appears on the block.

Use the **Output data type** parameter to set the data type of the outputs at the Label and Count ports. If you select Automatic, the block calculates the maximum number of objects that can fit inside the image based on the image size and the connectivity you specified. Based on this calculation, it determines the minimum output data type size that guarantees unique region labels and sets the output data type appropriately. If you select uint32, uint16, or uint8, the data type of the output is 32-, 16-, or 8-bit unsigned integers, respectively. If you select uint16, or uint8, the **If label exceeds data type size, mark remaining regions using** parameter appears in the dialog box. If the number of found objects exceeds the maximum number that can be represented by the output data type, use this parameter to specify the

block's behavior. If you select Maximum value of the output data type, the remaining regions are labeled with the maximum value of the output data type. If you select Zero, the remaining regions are labeled with zeroes.

Dialog Box

The Label dialog box appears as shown in the following figure.
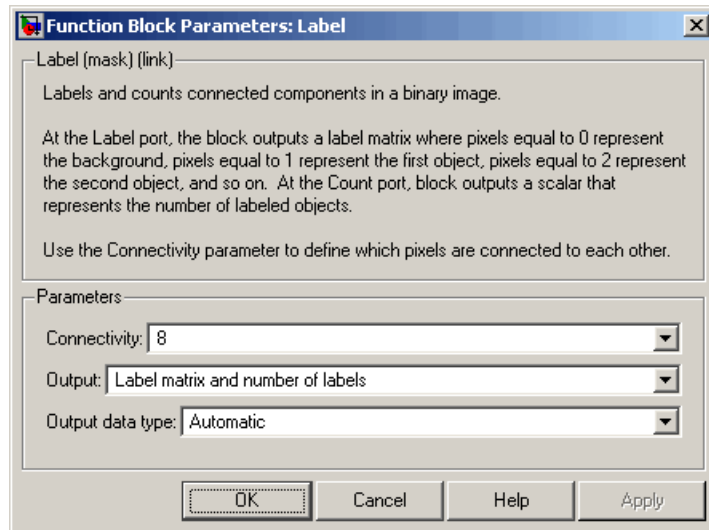
Connectivity

Specify which pixels are connected to each other. If you want a pixel to be connected to the pixels on the top, bottom, left, and right, select 4. If you want a pixel to be connected to the pixels on the top, bottom, left, right, and diagonally, select 8.

Output

Determine the block's output. If you select Label matrix and number of labels, the Label and Count ports appear on the block. The block outputs the label matrix at the Label port and the number of labeled objects at the Count port. If you select

Label matrix, the Label port appears on the block. If you select Number of labels, the Count port appears on the block.

Output data type
> Set the data type of the outputs at the Label and Count ports. If you select Automatic, the block determines the appropriate data type for the output. If you select uint32, uint16, or uint8, the data type of the output is 32-, 16-, or 8-bit unsigned integers, respectively.

If label exceeds data type size, mark remaining regions using
> Use this parameter to specify the block's behavior if the number of found objects exceeds the maximum number that can be represented by the output data type. If you select Maximum value of the output data type, the remaining regions are labeled with the maximum value of the output data type. If you select Zero, the remaining regions are labeled with zeroes. This parameter is visible if, for the **Output data type** parameter, you choose uint16 or uint8.

See Also

Bottom-hat	Computer Vision System Toolbox software
Closing	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Erosion	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
bwlabel	Image Processing Toolbox software
bwlabeln	Image Processing Toolbox software

Median Filter

Purpose   Perform 2-D median filtering

Library   Filtering and Analysis & Enhancement

visionanalysis

visionfilter
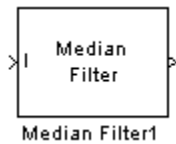
Description   The Median Filter block replaces the central value of an *M-by-N* neighborhood with its median value. If the neighborhood has a center element, the block places the median value there, as illustrated in the following figure.
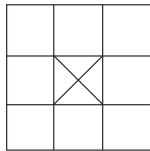
The block has a bias toward the upper-left corner when the neighborhood does not have an exact center. See the median value placement in the following figure.
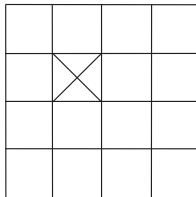
The block pads the edge of the input image, which sometimes causes the pixels within [M/2 N/2] of the edges to appear distorted. The median value is less sensitive than the mean to extreme values. As a result, the Median Filter block can remove salt-and-pepper noise from an image without significantly reducing the sharpness of the image.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Val	Scalar value that represents the constant pad value	Same as I port	No
Output	Matrix of intensity values	Same as I port	No

If the data type of the input signal is floating point, the output has the same data type. The data types of the signals input to the I and Val ports must be the same.

Fixed-Point Data Types

The information in this section is applicable only when the dimensions of the neighborhood are even.

For fixed-point inputs, you can specify accumulator and output data types as discussed in "Dialog Box" on page 1-499. Not all these fixed-point parameters apply to all types of fixed-point inputs. The following table shows the output and accumulator data type used for each fixed-point input.

Fixed-Point Input	Output Data Type	Accumulator Data Type
Even M	X	X
Odd M	X	

Median Filter

Fixed-Point Input	Output Data Type	Accumulator Data Type
Odd `M` and complex	X	X
Even `M` and complex	X	X

When `M` is even, fixed-point signals use the accumulator and output data types. The accumulator data type store the result of the sum performed while calculating the average of the two central rows of the input matrix. The output data type stores the total result of the average.

Complex fixed-point inputs use the accumulator parameters. The calculation for the sum of the squares of the real and imaginary parts of the input occur, before sorting input elements. The accumulator data type stores the result of the sum of the squares.

**Dialog
Box**

The **Main** pane of the Median Filter dialog box appears as shown in
the following figure.
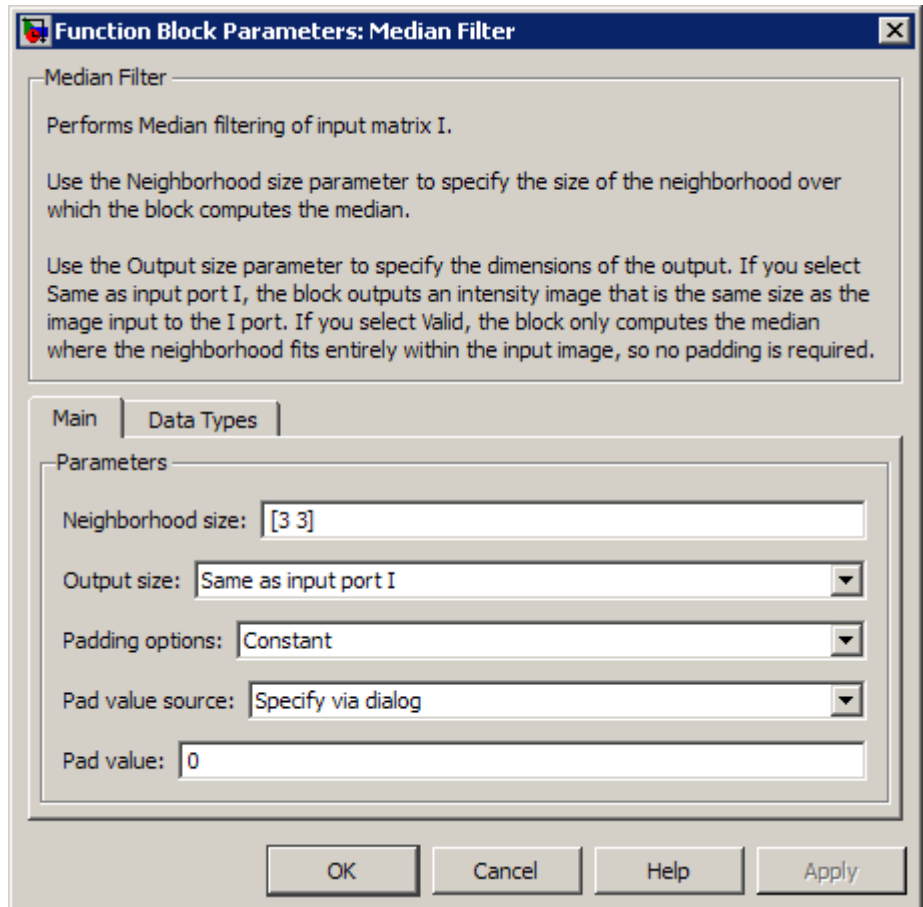
Neighborhood size
> Specify the size of the neighborhood over which the block
> computes the median.

Median Filter

- Enter a scalar value that represents the number of rows and columns in a square matrix.

- Enter a vector that represents the number of rows and columns in a rectangular matrix.

Output size

This parameter controls the size of the output matrix.

- If you choose `Same as input port I`, the output has the same dimensions as the input to port I. The **Padding options** parameter appears in the dialog box. Use the **Padding options** parameter to specify how to pad the boundary of your input matrix.

- If you select `Valid`, the block only computes the median where the neighborhood fits entirely within the input image, with no need for padding. The dimensions of the output image are,
`output rows = input rows - neighborhood rows + 1`,
and
`output columns = input columns - neighborhood columns + 1`.

Padding options

Specify how to pad the boundary of your input matrix.

- Select `Constant` to pad your matrix with a constant value. The **Pad value source** parameter appears in the dialog box

- Select `Replicate` to pad your input matrix by repeating its border values.

- Select `Symmetric` to pad your input matrix with its mirror image.

- Select `Circular` to pad your input matrix using a circular repetition of its elements. This parameter appears if, for the **Output size** parameter, you select `Same as input port I`.

For more information on padding, see the Image Pad block reference page.

Pad value source

Use this parameter to specify how to define your constant boundary value.

- Select `Specify via dialog` to enter your value in the block parameters dialog box. The **Pad value** parameter appears in the dialog box.

- Select `Input port` to specify your constant value using the `Val` port. This parameter appears if, for the **Padding options** parameter, you select `Constant`.

Pad value

Enter the constant value with which to pad your matrix. This parameter appears if, for the **Pad value source** parameter, you select `Specify via dialog`. Tunable.

The **Data Types** pane of the Median Filter dialog box appears as follows. The parameters on this dialog box becomes visible only when the dimensions of the neighborhood are even.

Median Filter
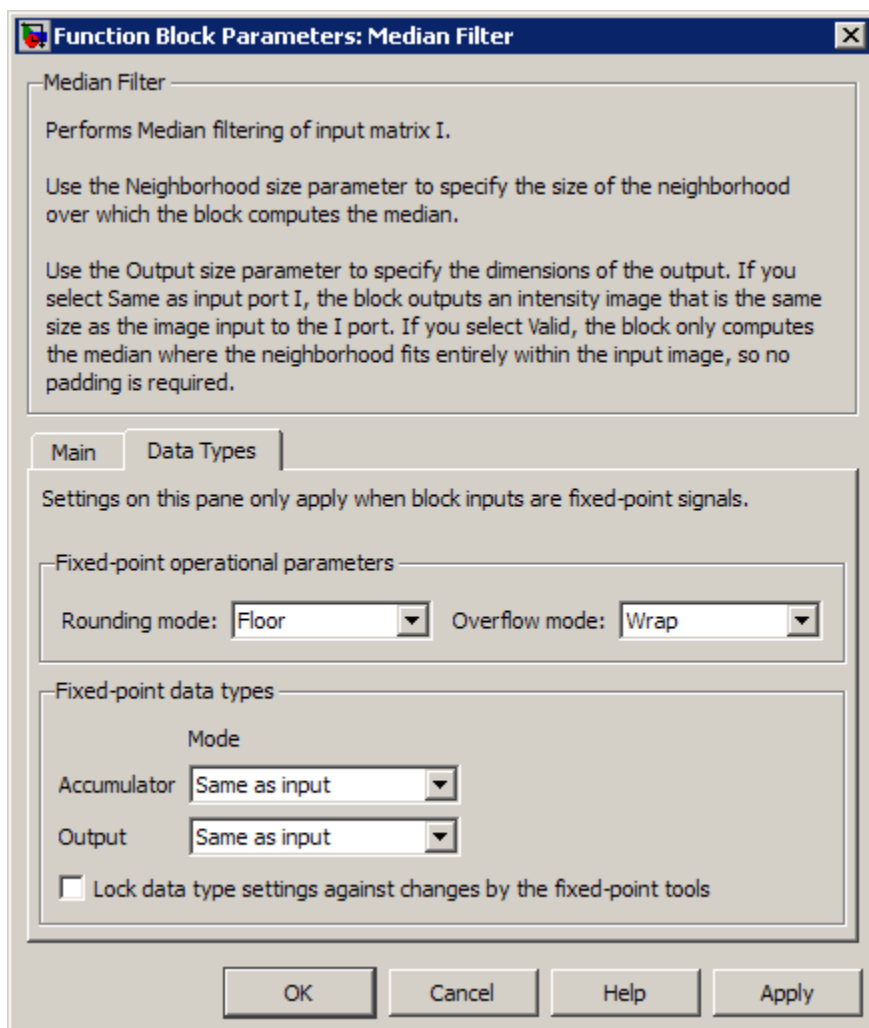
Rounding mode
　　　　Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Note Only certain cases require the use of the accumulator and output parameters. Refer to "Fixed-Point Data Types" on page 1-497 for more information.

Accumulator

Use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block:

- When you select `Same as input`, these characteristics match the related input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of 0.

Output

Choose how to specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match the related input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more

Median Filter

information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References   [1] Gonzales, Rafael C. and Richard E. Woods. *Digital Image Processing. 2nd ed*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

See Also

2-D Convolution	Computer Vision System Toolbox
2-D FIR Filter	Computer Vision System Toolbox
`medfilt2`	Image Processing Toolbox

Purpose   Perform morphological opening on binary or intensity images

Library   Morphological Operations

visionmorphops

Description   The Opening block performs an erosion operation followed by a dilation operation using a predefined neighborhood or structuring element. This block uses flat structuring elements only.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of ones and zeros that represents the neighborhood values	Boolean	No
Output	Scalar, vector, or matrix of intensity values that represents the opened image	Same as I port	No

The output signal has the same data type as the input to the I port.

Opening

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. You can only specify a structuring element using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the region the block moves throughout the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the use of a more efficient algorithm.

Dialog Box

The Opening dialog box appears as shown in the following figure.

Neighborhood or structuring element source

Specify how to enter your neighborhood or structuring element values. Select `Specify via dialog` to enter the values in the dialog box. Select `Input port` to use the Nhood port to specify the neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the `strel` function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or structuring element source** parameter, you select `Specify via dialog`.

References   [1] Soille, Pierre. *Morphological Image Analysis. 2nd ed.* New York: Springer, 2003.

See Also

Bottom-hat	Computer Vision System Toolbox software
Closing	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Erosion	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
Top-hat	Computer Vision System Toolbox software
imopen	Image Processing Toolbox software
strel	Image Processing Toolbox software

Optical Flow

Purpose   Estimate object velocities

Library   Analysis & Enhancement

              visionanalysis

Description
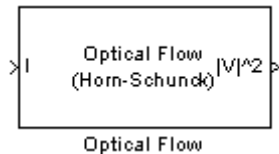
Optical Flow

The Optical Flow block estimates the direction and speed of object motion from one image to another or from one video frame to another using either the Horn-Schunck or the Lucas-Kanade method.

Port	Output	Supported Data Types	Complex Values Supported
I/I1	Scalar, vector, or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point (supported when the **Method** parameter is set to Lucas-Kanade)	No
I2	Scalar, vector, or matrix of intensity values	Same as I port	No
|V|^2	Matrix of velocity magnitudes	Same as I port	No
V	Matrix of velocity components in complex form	Same as I port	Yes

To compute the optical flow between two images, you must solve the following optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

In this equation, the following values are represented:

- I_x, I_y and I_t are the spatiotemporal image brightness derivatives
- u is the horizontal optical flow
- v is the vertical optical flow

Because this equation is underconstrained, there are several methods to solve for u and v:

- Horn-Schunck Method
- Lucas-Kanade Method

See the following two sections for descriptions of these methods

Horn-Schunck Method

By assuming that the optical flow is smooth over the entire image, the Horn-Schunck method computes an estimate of the velocity field,

$[u \quad v]^T$, that minimizes this equation:

$$E = \iint (I_x u + I_y v + I_t)^2 \, dxdy + \alpha \iint \left\{ \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right\} dxdy$$

In this equation, $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ are the spatial derivatives of the optical velocity component u, and α scales the global smoothness term. The Horn-Schunck method minimizes the previous equation to obtain the velocity field, $[u \ v]$, for each pixel in the image, which is given by the following equations:

Optical Flow

$$u_{x,y}^{k+1} = \bar{u}_{x,y}^{-k} - \frac{I_x[I_x\bar{u}_{x,y}^k + I_y\bar{v}_{x,y}^k + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v_{x,y}^{k+1} = \bar{v}_{x,y}^{-k} - \frac{I_y[I_x\bar{u}_{x,y}^k + I_y\bar{v}_{x,y}^k + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

In this equation, $\begin{bmatrix} u_{x,y}^k & v_{x,y}^k \end{bmatrix}$ is the velocity estimate for the pixel at (x,y), and $\begin{bmatrix} \bar{u}_{x,y}^{-k} & \bar{v}_{x,y}^{-k} \end{bmatrix}$ is the neighborhood average of $\begin{bmatrix} u_{x,y}^k & v_{x,y}^k \end{bmatrix}$. For $k=0$, the initial velocity is 0.

When you choose the `Horn-Schunck` method, u and v are solved as follows:

1 Compute I_x and I_y using the Sobel convolution kernel:

$\begin{bmatrix} -1 & -2 & -1; & 0 & 0 & 0; & 1 & 2 & 1 \end{bmatrix}$, and its transposed form for each pixel in the first image.

2 Compute I_t between images 1 and 2 using the $\begin{bmatrix} -1 & 1 \end{bmatrix}$ kernel.

3 Assume the previous velocity to be 0, and compute the average velocity for each pixel using $\begin{bmatrix} 0 & 1 & 0; & 1 & 0 & 1; & 0 & 1 & 0 \end{bmatrix}$ as a convolution kernel.

4 Iteratively solve for u and v.

Lucas-Kanade Method

To solve the optical flow constraint equation for u and v, the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity in each section. Then, it performs a weighted least-square fit of the optical flow constraint equation to a constant model for $\begin{bmatrix} u & v \end{bmatrix}^T$ in each section, Ω, by minimizing the following equation:

$$\sum_{x \in \Omega} W^2 [I_x u + I_y v + I_t]^2$$

Here, W is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is given by the following equation:

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

When you choose the `Lucas-Kanade` method, I_t is computed using a difference filter or a derivative of a Gaussian filter.

The two following sections explain how I_x, I_y, I_t, and then u and v are computed.

Difference Filter

When you set the **Temporal gradient filter** to `Difference filter [-1 1]`, u and v are solved as follows:

1 Compute I_x and I_y using the kernel $\begin{bmatrix} -1 & 8 & 0 & -8 & 1 \end{bmatrix}/12$ and its transposed form.

   If you are working with fixed-point data types, the kernel values are signed fixed-point values with word length equal to 16 and fraction length equal to 15.

2 Compute I_t between images 1 and 2 using the $\begin{bmatrix} -1 & 1 \end{bmatrix}$ kernel.

3 Smooth the gradient components, I_x, I_y, and I_t, using a separable and isotropic 5-by-5 element kernel whose effective 1-D coefficients

   are $\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}/16$. If you are working with fixed-point data types, the kernel values are unsigned fixed-point values with word length equal to 8 and fraction length equal to 7.

4 Solve the 2-by-2 linear equations for each pixel using the following method:

- If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

  Then the eigenvalues of A are $\lambda_i = \dfrac{a+c}{2} \pm \dfrac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

  In the fixed-point diagrams, $P = \dfrac{a+c}{2}, Q = \dfrac{\sqrt{4b^2 + (a-c)^2}}{2}$

- The eigenvalues are compared to the threshold, τ, that corresponds to the value you enter for the threshold for noise reduction. The results fall into one of the following cases:

  Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

  A is nonsingular, the system of equations are solved using Cramer's rule.

  Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

  A is singular (noninvertible), the gradient flow is normalized to calculate u and v.

  Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

  The optical flow, u and v, is 0.

Derivative of Gaussian

If you set the temporal gradient filter to `Derivative of Gaussian`, u and v are solved using the following steps. You can see the flow chart for this process at the end of this section:

1 Compute I_x and I_y using the following steps:

a Use a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the **Number of frames to buffer for temporal smoothing** parameter.

b Use a Gaussian filter and the derivative of a Gaussian filter to smooth the image using spatial filtering. Specify the standard deviation and length of the image smoothing filter using the **Standard deviation for image smoothing filter** parameter.

2 Compute I_t between images 1 and 2 using the following steps:

a Use the derivative of a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the **Number of frames to buffer for temporal smoothing** parameter.

b Use the filter described in step 1b to perform spatial filtering on the output of the temporal filter.

3 Smooth the gradient components, I_x, I_y, and I_t, using a gradient smoothing filter. Use the **Standard deviation for gradient smoothing filter** parameter to specify the standard deviation and the number of filter coefficients for the gradient smoothing filter.

4 Solve the 2-by-2 linear equations for each pixel using the following method:

- If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

  Then the eigenvalues of A are $\lambda_i = \dfrac{a+c}{2} \pm \dfrac{\sqrt{4b^2 + (a-c)^2}}{2} ; i = 1, 2$

- When the block finds the eigenvalues, it compares them to the threshold, τ, that corresponds to the value you enter for the

Threshold for noise reduction parameter. The results fall into one of the following cases:

Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

A is nonsingular, so the block solves the system of equations using Cramer's rule.

Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

A is singular (noninvertible), so the block normalizes the gradient flow to calculate u and v.

Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

The optical flow, u and v, is 0.
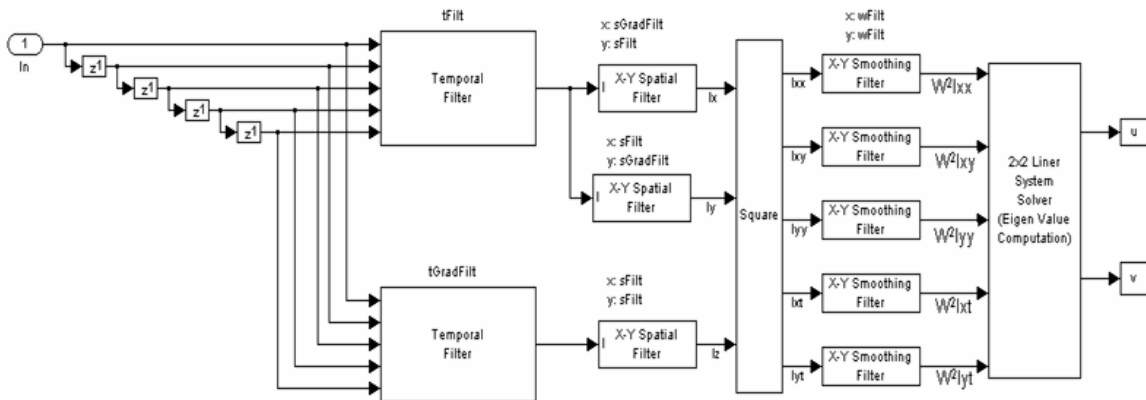
tFilt = Coefficients of Gaussian Filter
tGradFilt = Coefficients of the Derivative of a Gaussian Filter

sFilt = Coefficients of Gaussian Filter
sGradFilt = Coefficients of the Derivative of a Gaussian Filter

Fixed-Point Data Type Diagram

The following diagrams shows the data types used in the Optical Flow block for fixed-point signals. The block supports fixed-point data types only when the **Method** parameter is set to Lucas-Kanade.

Data type diagram for Optical Flow block's overall algorithm
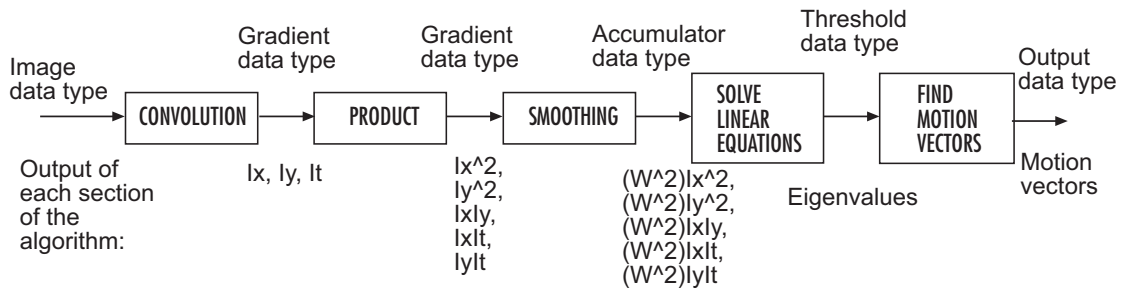
Data type diagram for convolution algorithm
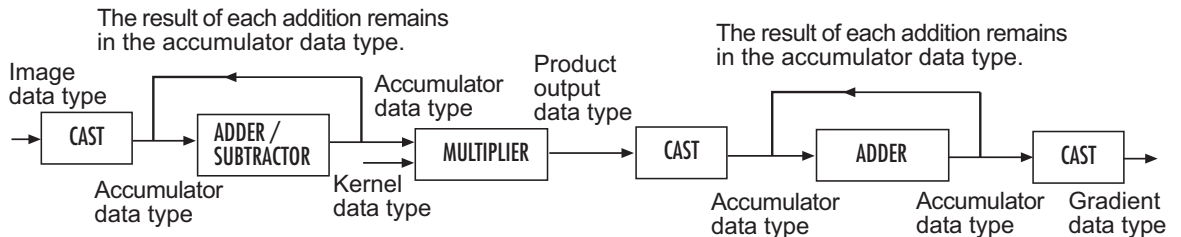
Data type diagram for smoothing algorithm
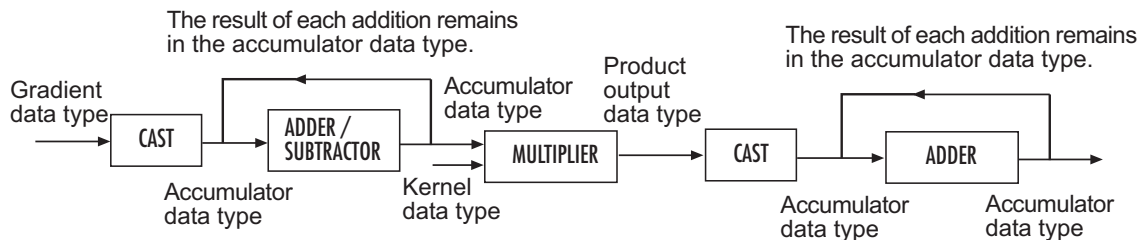
Optical Flow

Data type diagram for product algorithm
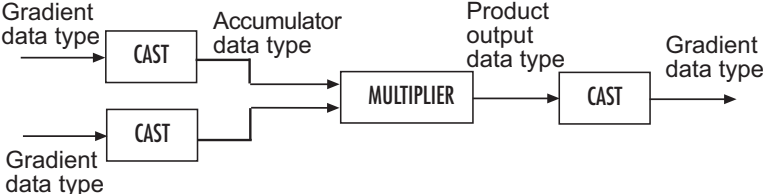
Solving linear equations to compute eigenvalues
(see Step 4 in the Lucas-Kanade Method section for the eigenvalue equations)

Data type diagram for P

The result of each addition remains
in the accumulator data type.
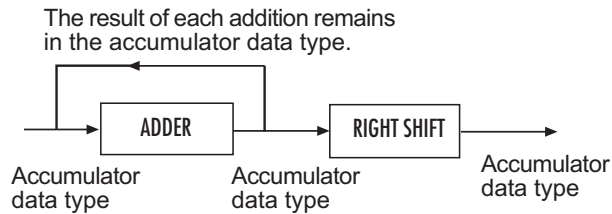
Data type diagram for 4b^2

Data type diagram for (a-c)^2

The result of each addition remains
in the accumulator data type.

Data type diagram for Q

The result of each addition remains
in the accumulator data type.
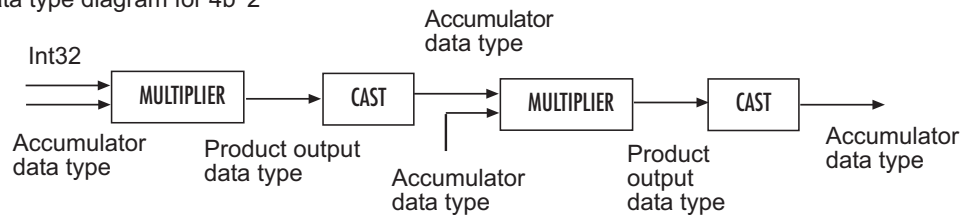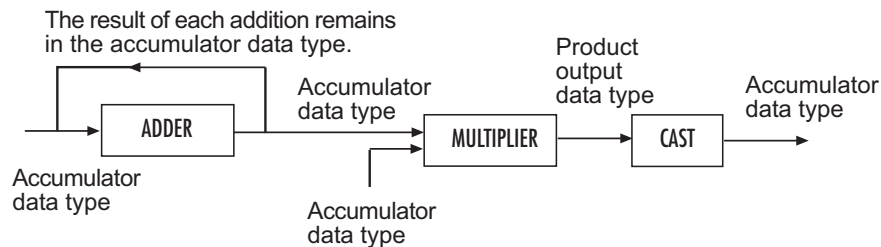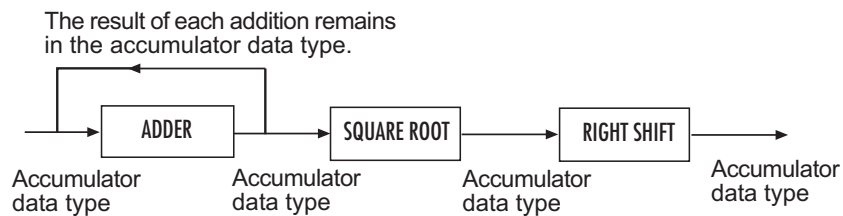
Data type diagram for eigenvalues

The result of each addition remains
in the accumulator data type.
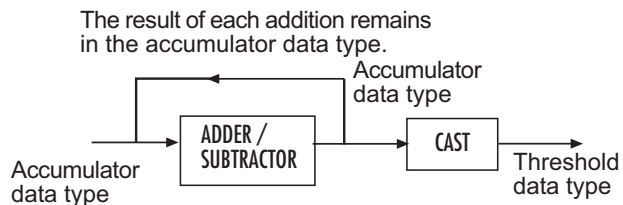
Optical Flow

Data type diagram for finding the motion vectors algorithm
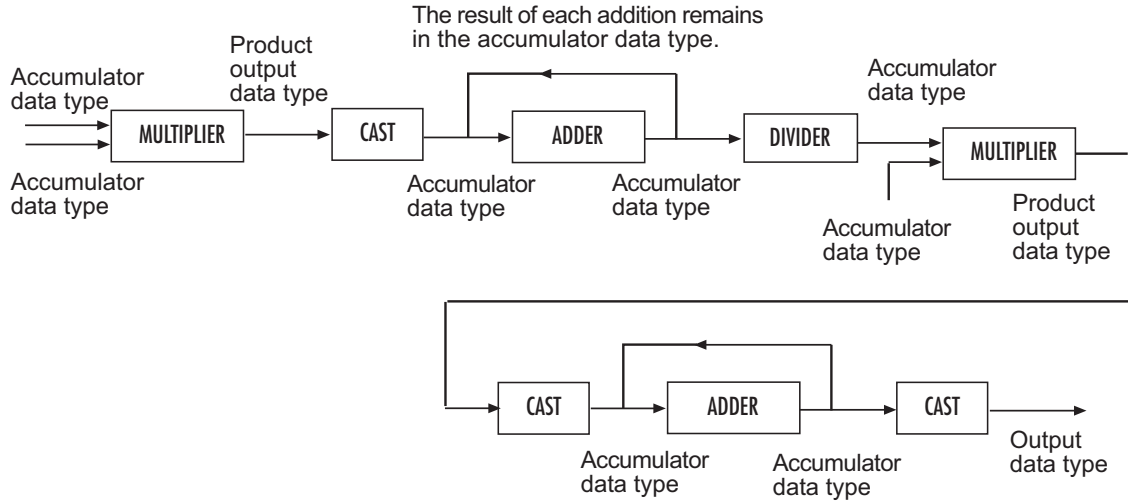
You can set the product output, accumulator, gradients, threshold, and output data types in the block mask.

Dialog Box

The **Main** pane of the Optical Flow dialog box appears as shown in the following figure.
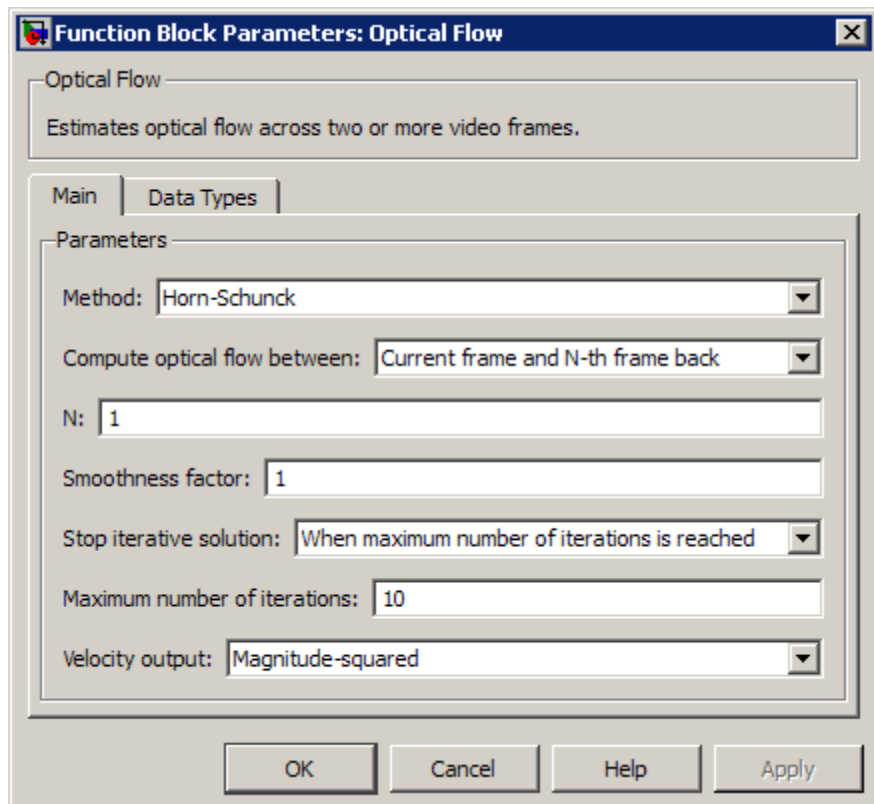
Method

Select the method the block uses to calculate the optical flow. Your choices are Horn-Schunck or Lucas-Kanade.

Compute optical flow between

Select Two images to compute the optical flow between two images. Select Current frame and N-th frame back to compute the optical flow between two video frames that are N frames apart.

This parameter is visible if you set the **Method** parameter to `Horn-Schunck` or you set the **Method** parameter to `Lucas-Kanade` and the **Temporal gradient filter** to `Difference filter [-1 1]`.

N

Enter a scalar value that represents the number of frames between the reference frame and the current frame. This parameter becomes available if you set the **Compute optical flow between** parameter, you select `Current frame and N-th frame back`.

Smoothness factor

If the relative motion between the two images or video frames is large, enter a large positive scalar value. If the relative motion is small, enter a small positive scalar value. This parameter becomes available if you set the **Method** parameter to `Horn-Schunck`.

Stop iterative solution

Use this parameter to control when the block's iterative solution process stops. If you want it to stop when the velocity difference is below a certain threshold value, select `When velocity difference falls below threshold`. If you want it to stop after a certain number of iterations, choose `When maximum number of iterations is reached`. You can also select `Whichever comes first`. This parameter becomes available if you set the **Method** parameter to `Horn-Schunck`.

Maximum number of iterations

Enter a scalar value that represents the maximum number of iterations you want the block to perform. This parameter is only visible if, for the **Stop iterative solution** parameter, you select `When maximum number of iterations is reached` or `Whichever comes first`. This parameter becomes available if you set the **Method** parameter to `Horn-Schunck`.

Velocity difference threshold

Enter a scalar threshold value. This parameter is only visible if, for the **Stop iterative solution** parameter, you select `When`

velocity difference falls below threshold or Whichever comes first. This parameter becomes available if you set the **Method** parameter to Horn-Schunck.

Velocity output

If you select Magnitude-squared, the block outputs the optical

flow matrix where each element is of the form $u^2 + v^2$. If you select Horizontal and vertical components in complex form, the block outputs the optical flow matrix where each

element is of the form $u + jv$.

Temporal gradient filter

Specify whether the block solves for u and v using a difference filter or a derivative of a Gaussian filter. This parameter becomes available if you set the **Method** parameter to Lucas-Kanade.

Number of frames to buffer for temporal smoothing

Use this parameter to specify the temporal filter characteristics such as the standard deviation and number of filter coefficients. This parameter becomes available if you set the **Temporal gradient filter** parameter to Derivative of Gaussian.

Standard deviation for image smoothing filter

Specify the standard deviation for the image smoothing filter. This parameter becomes available if you set the **Temporal gradient filter** parameter to Derivative of Gaussian.

Standard deviation for gradient smoothing filter

Specify the standard deviation for the gradient smoothing filter. This parameter becomes available if you set the **Temporal gradient filter** parameter to Derivative of Gaussian.

Discard normal flow estimates when constraint equation is ill-conditioned

Select this check box if you want the block to set the motion vector to zero when the optical flow constraint equation is ill-conditioned. This parameter becomes available if you set the **Temporal gradient filter** parameter to Derivative of Gaussian.

Optical Flow

Output image corresponding to motion vectors (accounts for block delay)

Select this check box if you want the block to output the image that corresponds to the motion vector being output by the block. This parameter becomes available if you set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Threshold for noise reduction

Enter a scalar value that determines the motion threshold between each image or video frame. The higher the number, the less small movements impact the optical flow calculation. This parameter becomes available if you set the **Method** parameter to `Lucas-Kanade`.

The **Data Types** pane of the Optical Flow dialog box appears as shown in the following figure. The parameters on this dialog box becomes visible only when the Lucas-Kanade method is selected.

Function Block Parameters: Optical Flow

Optical Flow

Estimates optical flow across two or more video frames.

| Main | Data Types |

Settings on this pane only apply when block inputs are fixed-point signals.

Fixed-point operational parameters

Rounding mode: Nearest   Overflow mode: Saturate

Fixed-point data types

	Mode	Signed	Word length	Fraction length
Product output	Binary point scaling	Yes	32	20
Accumulator	Same as product output			
Gradients	Same as accumulator			
Threshold	Same word length as first input			
Output	Binary point scaling	No	32	20

☐ Lock data type settings against changes by the fixed-point tools

| OK | Cancel | Help | Apply |

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Product output

Use this parameter to specify how to designate the product output word and fraction lengths.

Accumulator
data type
→

Product
data type

MULTIPLIER

→

Accumulator
or kernel
data type
→

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Gradients

Choose how to specify the word length and fraction length of the gradients data type:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the quotient, in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the quotient. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Threshold
Choose how to specify the word length and fraction length of the threshold data type:

- When you select `Same word length as first input`, the threshold word length matches that of the first input.

- When you select `Specify word length`, enter the word length of the threshold data type.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the threshold, in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the threshold. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output
Choose how to specify the word length and fraction length of the output data type:

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length in bits and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more

information, see `fxptdlg`, a reference page on the Fixed-Point
Tool in the Simulink documentation.

References   [1] Barron, J.L., D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt.
Performance of optical flow techniques. CVPR, 1992.

See Also

Block Matching	Computer Vision System Toolbox software
Gaussian Pyramid	Computer Vision System Toolbox software

Projective Transformation (To Be Removed)

Purpose   Transform quadrilateral into another quadrilateral

Library   Geometric Transformations

vipgeotforms

Description
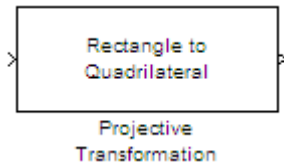
> **Note** This block will be removed in a future release. It is recommended that you replace this block with the Apply Geometric Transformation and Estimate Geometric Transformation block. The replacement will require that you modify your model.
>
> See the Projective Transformation documentation to view the reference material related to the obsolete block.

PSNR

Purpose   Compute peak signal-to-noise ratio (PSNR) between images

Library   Statistics

visionstatistics

Description   The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is often used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

The *Mean Square Error (MSE)* and the *Peak Signal to Noise Ratio (PSNR)* are the two error metrics used to compare image compression quality. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error.

To compute the PSNR, the block first calculates the mean-squared error using the following equation:

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M*N}$$

In the previous equation, *M* and *N* are the number of rows and columns in the input images, respectively. Then the block computes the PSNR using the following equation:

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right)$$

In the previous equation, *R* is the maximum fluctuation in the input image data type. For example, if the input image has a double-precision floating-point data type, then *R* is 1. If it has an 8-bit unsigned integer data type, *R* is 255, etc.

Recommendation for Computing PSNR for Color Images

Different approaches exist for computing the PSNR of a color image. Because the human eye is most sensitive to luma information, compute the PSNR for color images by converting the image to a color space that separates the intensity (luma) channel, such as YCbCr. The Y (luma), in YCbCr represents a weighted average of R, G, and B. G is given the most weight, again because the human eye perceives it most easily. With this consideration, compute the PSNR only on the luma channel.

Ports

Port	Output	Supported Data Types	Complex Values Supported
I1	Scalar, vector, or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
I2	Scalar, vector, or matrix of intensity values	Same as I1 port	No
Output	Scalar value that represents the PSNR	• Double-precision floating point For fixed-point or integer input, the block output is double-precision floating point. Otherwise, the block input and output are the same data type.	No

Dialog Box

The PSNR dialog box appears as shown in the following figure.
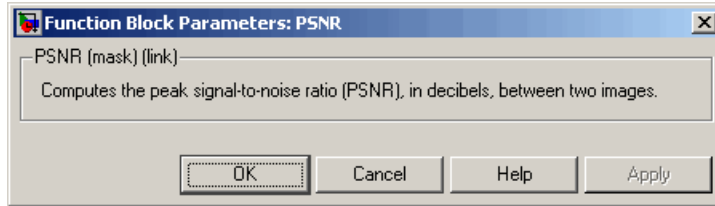
Purpose   Read uncompressed video frames from AVI file

Library   vipobslib

Description

barcodes.avi
480x640, 30 fps
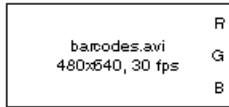
Read AVI File

Note The Read AVI File block is obsolete. It may be removed in a future version of the Computer Vision System Toolbox blocks. Use the replacement block From Multimedia File.

See the Read AVI File documentation to view the reference material related to the obsolete block.

Read Binary File

Purpose   Read binary video data from files

Library   Sources

              visionsources

Description   The Read Binary File block reads video data from a binary file and
              imports it into a Simulink model.

              This block takes user specified parameters that describe the format of
              the video data. These parameters together with the raw binary file,
              which stores only raw pixel values, creates the video data signal for a
              Simulink model. The video data read by this block must be stored in
              row major format.

              Note This block supports code generation only for platforms that have
              file I/O available. You cannot use this block to do code generation with
              Real-Time Windows Target (RTWin).

Port	Output	Supported Data Types	Complex Values Supported
Output	Scalar, vector, or matrix of integer values	• 8-, 16- 32-bit signed integer • 8-, 16- 32-bit unsigned integer	No
EOF	Scalar value	Boolean	No

Four Character Code Video Formats

Four Character Codes (FOURCC) identify video formats. For more
information about these codes, see http://www.fourcc.org.

Use the **Four character code** parameter to identify the binary file
format. Then, use the **Rows** and **Cols** parameters to define the size
of the output matrix. These dimensions should match the matrix
dimensions of the data inside the file.

Custom Video Formats

If your binary file contains data that is not in FOURCC format, you can configure the Read Binary File block to understand a custom format:

- Use the **Bit stream format** parameter to specify whether your data is planar or packed. If your data is packed, use the **Rows** and **Cols** parameters to define the size of the output matrix.

- Use the **Number of output components** parameter to specify the number of components in the binary file. This number corresponds to the number of block output ports.

- Use the **Component**, **Bits**, **Rows**, and **Cols** parameters to specify the component name, bit size, and size of the output matrices, respectively. The block uses the **Component** parameter to label the output ports.

- Use the **Component order in binary file** parameter to specify how the components are arranged within the file.

- Select the **Interlaced video** check box if the binary file contains interlaced video data.

- Select the **Input file has signed data** check box if the binary file contains signed integers.

- Use the **Byte order in binary file** to indicate whether your binary file has little endian or big endian byte ordering.

Read Binary File

Dialog Box

The Read Binary File dialog box appears as shown in the following figure.
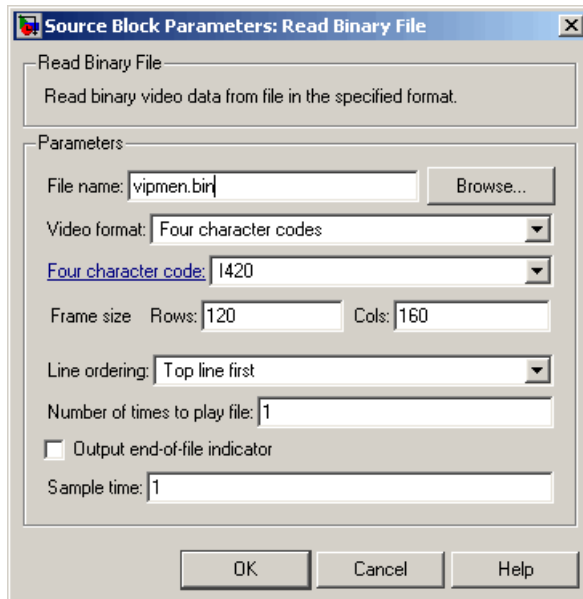
File name

Specify the name of the binary file to read. If the location of this file is on your MATLAB path, enter the filename. If the location of this file is not on your MATLAB path, use the **Browse** button to specify the full path to the file as well as the filename.

Video format

Specify the format of the binary video data. Your choices are Four character codes or Custom. See "Four Character Code Video Formats" on page 1-532 or "Custom Video Formats" on page 1-533 for more details.

Four character code

From the drop-down list, select the binary file format.

Frame size: Rows, Cols

Define the size of the output matrix. These dimensions should match the matrix dimensions of the data inside the file.

Line ordering

Specify how the block fills the output matrix.If you select `Top line first`, the block first fills the first row of the output matrix with the contents of the binary file. It then fills the other rows in increasing order. If you select `Bottom line first`, the block first fills the last row of the output matrix. It then fills the other rows in decreasing order.

Number of times to play file

Specify the number of times to play the file. The number you enter must be a positive integer or `inf`, to play the file until you stop the simulation.

Output end-of-file indicator

Specifies the output is the last video frame in the binary file. When you select this check box, a Boolean output port labeled EOF appears on the block. The output from the EOF port is 1 when the last video frame in the binary file is output from the block. Otherwise, the output from the EOF port is 0.

Sample time

Specify the sample period of the output signal.

Read Binary File
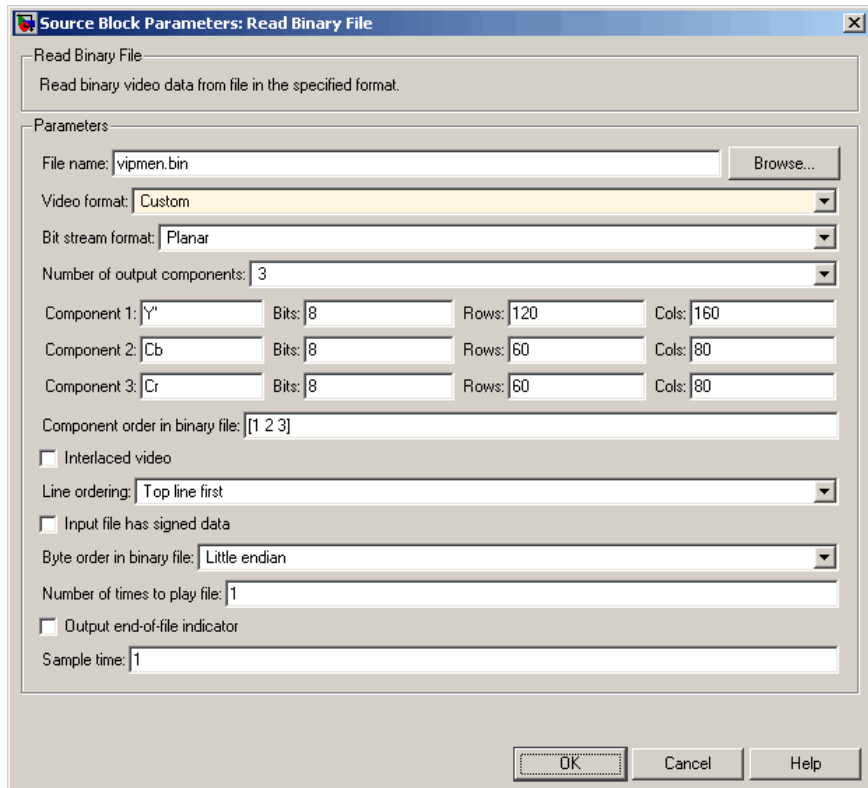
Bit stream format

Specify whether your data is planar or packed.

Frame size: Rows, Cols

Define the size of the output matrix. This parameter appears when you select a **Bit stream format** parameter of `Packed`.

Number of output components

Specify the number of components in the binary file.

Component, Bits, Rows, Cols

Specify the component name, bit size, and size of the output matrices, respectively.

Component order in binary file
   Specify the order in which the components appear in the binary file.

Interlaced video
   Select this check box if the binary file contains interlaced video data.

Input file has signed data
   Select this check box if the binary file contains signed integers.

Byte order in binary file
   Use this parameter to indicate whether your binary file has little endian or big endian byte ordering.

See Also

From Multimedia File	Computer Vision System Toolbox
Write Binary File	Computer Vision System Toolbox

Resize

Purpose   Enlarge or shrink image sizes

Library   Geometric Transformations

visiongeotforms

Description   The Resize block enlarges or shrinks an image by resizing the image along one dimension (row or column). Then, it resizes the image along the other dimension (column or row).

This block supports intensity and color images on its ports. When you input a floating point data type signal, the block outputs the same data type.

Shrinking an image can introduce high frequency components into the image and aliasing might occur. If you select the **Perform antialiasing when resize factor is between 0 and 100** check box, the block performs low pass filtering on the input image before shrinking it.

Port Description

Port	Input/Output	Supported Data Types	Complex Values Supported
Image / Input	*M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* color video signal where *P* is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
ROI	Four-element vector [x y width height] that defines the ROI	• Double-precision floating point (only supported if the input to the Input port is floating point) • Single-precision floating point (only supported if the input to the Input port is floating point)	No

Port	Input/Output	Supported Data Types	Complex Values Supported
		• 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	
Output	Resized image	Same as Input port	No
Flag	Boolean value that indicates whether the ROI is within the image bounds	Boolean	No

ROI Processing

To resize a particular region of each image, select the **Enable ROI processing** check box. To enable this option, select the following parameter values.

- **Specify** = Number of output rows and columns

- **Interpolation method** = Nearest neighbor, Bilinear, or Bicubic

- Clear the **Perform antialiasing when resize factor is between 0 and 100** check box.

If you select the **Enable ROI processing** check box, the ROI port appears on the block. Use this port to define a region of interest (ROI) in the input matrix, that you want to resize. The input to this port must be a four-element vector, [x y width height]. The first two elements define the upper-left corner of the ROI, and the second two elements define the width and height of the ROI.

If you select the **Enable ROI processing** check box, the **Output flag indicating if any part of ROI is outside image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output.

Flag Port Output	Description
0	ROI is completely inside the input image.
1	ROI is completely or partially outside the input image.

Fixed-Point Data Types

The following diagram shows the data types used in the Resize block for fixed-point signals.
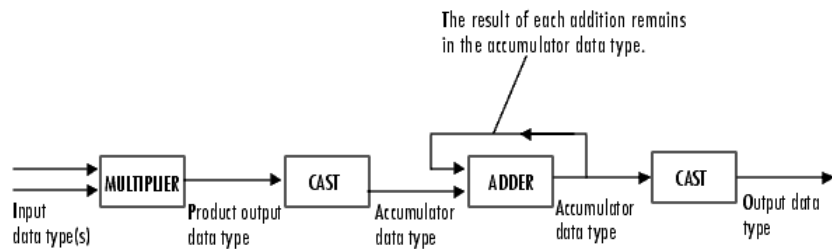
You can set the interpolation weights table, product output, accumulator, and output data types in the block mask.

Dialog Box

The **Main** pane of the Resize dialog box appears as shown in the following figure:

Function Block Parameters: Resize

Resize

Change the size of an image or a region of interest within an image.

Use the Specify parameter to designate the parameters you want to use to resize you
specify a scalar percentage that is applied to both rows and columns, or you can spec
vector to scale the rows and columns differently. You can specify the number of rows a
want your output image to have, and you can preserve or change the image's aspect

Use the Interpolation method parameter to specify the type of interpolation performed
it is resizing a image. If you select the Perform antialiasing when resize factor is betw
box, the block performs lowpass filtering on the input image before shrinking it.

| Main | Data Types |

Parameters

Specify: Output size as a percentage of input size

Resize factor in %: [200 150]

Interpolation method: Bilinear

☑ Perform antialiasing when resize factor is between 0 and 100

OK   Cancel

Resize

Specify

Specify which aspects of the image to resize. Your choices are
`Output size as a percentage of input size`, `Number of
output columns and preserve aspect ratio`, `Number of
output rows and preserve aspect ratio`, or `Number of
output rows and columns`.

When you select `Output size as a percentage of input
size`, the **Resize factor in %** parameter appears in the dialog
box. Enter a scalar percentage value that is applied to both rows
and columns.

When you select `Number of output columns and preserve
aspect ratio`, the **Number of output columns** parameter
appears in the dialog box. Enter a scalar value that represents
the number of columns you want the output image to have. The
block calculates the number of output rows so that the output
image has the same aspect ratio as the input image.

When you select `Number of output rows and preserve aspect
ratio`, the **Number of output rows** parameter appears in the
dialog box. Enter a scalar value that represents the number of
rows you want the output image to have. The block calculates the
number of output columns so that the output image has the same
aspect ratio as the input image.

When you select `Number of output rows and columns`, the
Number of output rows and columns parameter appears
in the dialog box. Enter a two-element vector, where the first
element is the number of rows in the output image and the second
element is the number of columns. In this case, the aspect ratio of
the image can change.

Resize factor in %

Enter a scalar percentage value that is applied to both rows and
columns or a two-element vector, where the first element is the
percentage by which to resize the rows and the second element is

the percentage by which to resize the columns. This parameter is visible if, for the **Specify** parameter, you select Output size as a percentage of input size.

You must enter a scalar value that is greater than zero. The table below describes the affect of the resize factor value:

Resize factor in %	**Resizing of image**
0 < resize factor < 100	The block shrinks the image.
resize factor = 100	Image unchanged.
resize factor > 100	The block enlarges the image.

The dimensions of the output matrix depend on the **Resize factor in %** parameter and are given by the following equations:

```
number_output_rows =
round(number_input_rows*resize_factor/100);
```

```
number_output_cols =
round(number_input_cols*resize_factor/100);
```

Number of output columns
Enter a scalar value that represents the number of columns you want the output image to have. This parameter is visible if, for the **Specify** parameter, you select Number of output columns and preserve aspect ratio.

Number of output rows
Enter a scalar value that represents the number of rows you want the output image to have. This parameter is visible if, for the **Specify** parameter, you select Number of output rows and preserve aspect ratio.

Resize

Number of output rows and columns

Enter a two-element vector, where the first element is the number of rows in the output image and the second element is the number of columns. This parameter is visible if, for the **Specify** parameter, you select `Number of output rows and columns`.

Interpolation method

Specify which interpolation method to resize the image.

When you select `Nearest neighbor`, the block uses one nearby pixel to interpolate the pixel value. This option though the most efficient, is the least accurate. When you select `Bilinear`, the block uses four nearby pixels to interpolate the pixel value. When you select `Bicubic` or `Lanczos2`, the block uses 16 nearby pixels to interpolate the pixel value. When you select `Lanczos3`, the block uses 36 surrounding pixels to interpolate the pixel value.

The Resize block performs optimally when you set this parameter to `Nearest neighbor` with one of the following conditions:

- You set the **Resize factor in %** parameter to a multiple of 100.

- Dividing 100 by the **Resize factor in %** parameter value results in an integer value.

Perform antialiasing when resize factor is between 0 and 100

If you select this check box, the block performs low-pass filtering on the input image before shrinking it to prevent aliasing.

Enable ROI processing

Select this check box to resize a particular region of each image. This parameter is available when the **Specify** parameter is set to `Number of output rows and columns`, the **Interpolation method** parameter is set to `Nearest neighbor`, `Bilinear`, or `Bicubic`, and the **Perform antialiasing when resize factor is between 0 and 100** check box is not selected.

Output flag indicating if any part of ROI is outside image bounds

> If you select this check box, the Flag port appears on the block. The block outputs 1 at this port if the ROI is completely or partially outside the input image. Otherwise, it outputs 0.

The **Data Types** pane of the Resize dialog box appears as shown in the following figure.

Resize
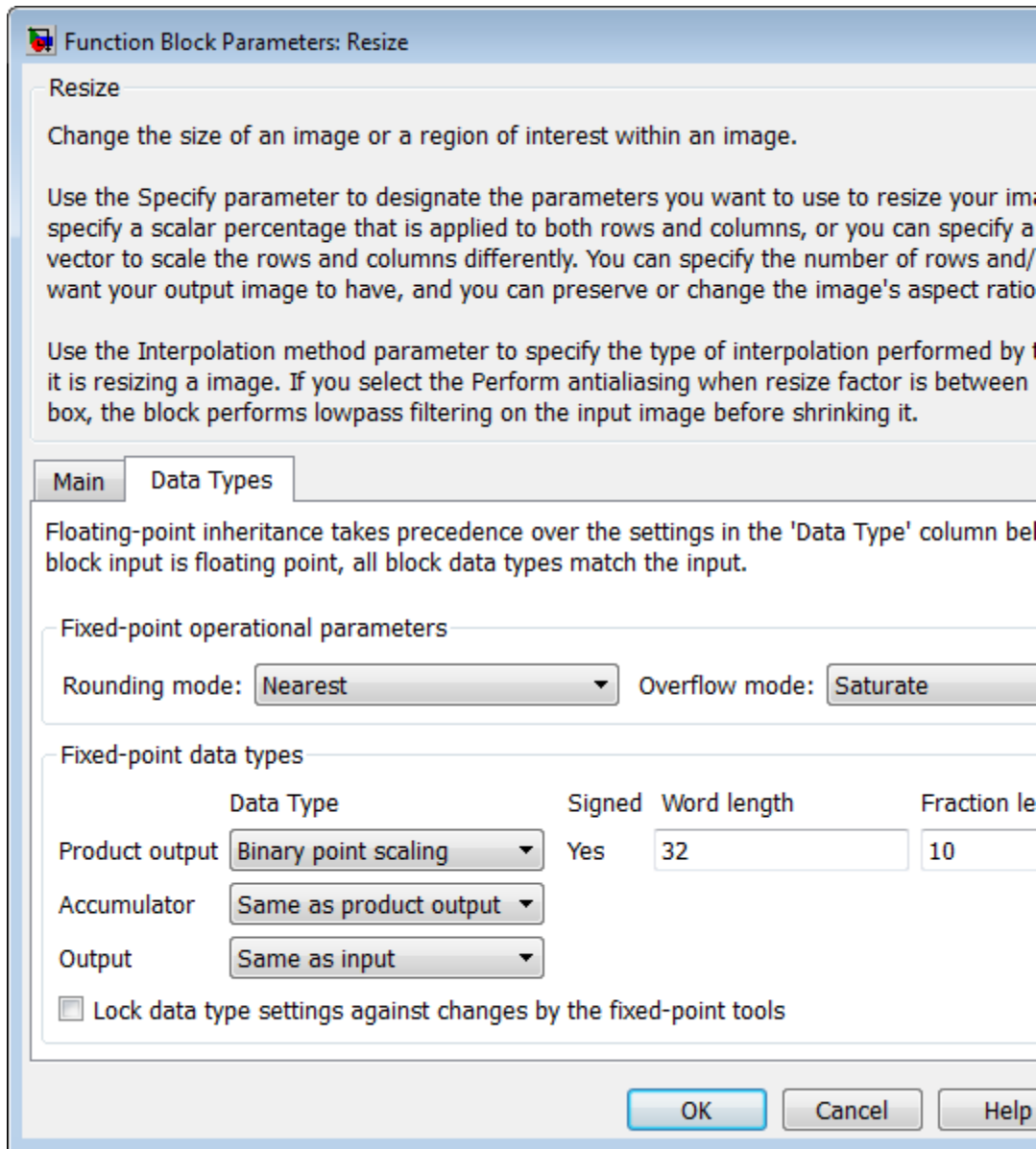
Rounding mode

> Select the rounding mode for fixed-point operations.

Overflow mode

> Select the overflow mode for fixed-point operations.

Interpolation weights table

> Choose how to specify the word length of the values of the interpolation weights table. The fraction length of the interpolation weights table values is always equal to the word length minus one:

> - When you select `Same as input`, the word length of the interpolation weights table values match that of the input to the block.

> - When you select `Binary point scaling`, you can enter the word length of the interpolation weights table values, in bits.

> - When you select `Slope and bias scaling`, you can enter the word length of the interpolation weights table values, in bits.

Product output

> As depicted in the preceding diagram, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

> - When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

The result of each addition remains in the accumulator data type.
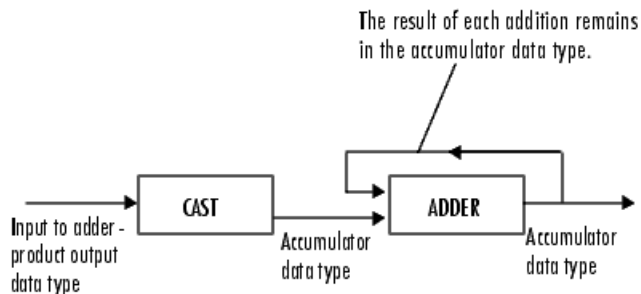
As depicted in the preceding diagram, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References   [1] Ward, Joseph and David R. Cok. "Resampling Algorithms for Image Resizing and Rotation", *Proc. SPIE Digital Image Processing Applications*, vol. 1075, pp. 260-269, 1989.

[2] Wolberg, George. *Digital Image Warping*. Washington: IEEE Computer Society Press, 1990.

See Also

Rotate	Computer Vision System Toolbox software
Shear	Computer Vision System Toolbox software
Translate	Computer Vision System Toolbox software
imresize	Image Processing Toolbox software

Resize (To Be Removed)

Purpose   Enlarge or shrink image sizes

Library   Geometric Transformations

vipgeotforms

Description

> **Note** This Resize block will be removed in a future release. It uses
> the zero-based, [row column] coordinate system. It is recommended
> that you replace this block with the updated Resize block that uses the
> one-based, [x y] coordinate system.
>
> Refer to "Conventions Changed for Indexing, Spatial Coordinates, and
> Representation of Geometric Transforms" in the R2011b Release Notes
> for details related to these changes.
>
> See the Resize documentation to view the reference material related to
> the obsolete block.

Purpose   Rotate image by specified angle

Library   Geometric Transformations

visiongeotforms

Description   Use the Rotate block to rotate an image by an angle specified in radians.

Note This block supports intensity and color images on its ports.

Port	Description
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes
Angle	Rotation angle
Output	Rotated matrix

The Rotate block uses the 3-pass shear rotation algorithm to compute its values, which is different than the algorithm used by the imrotate function in the Image Processing Toolbox.

Fixed-Point Data Types

The following diagram shows the data types used in the Rotate block for bilinear interpolation of fixed-point signals.

Rotate
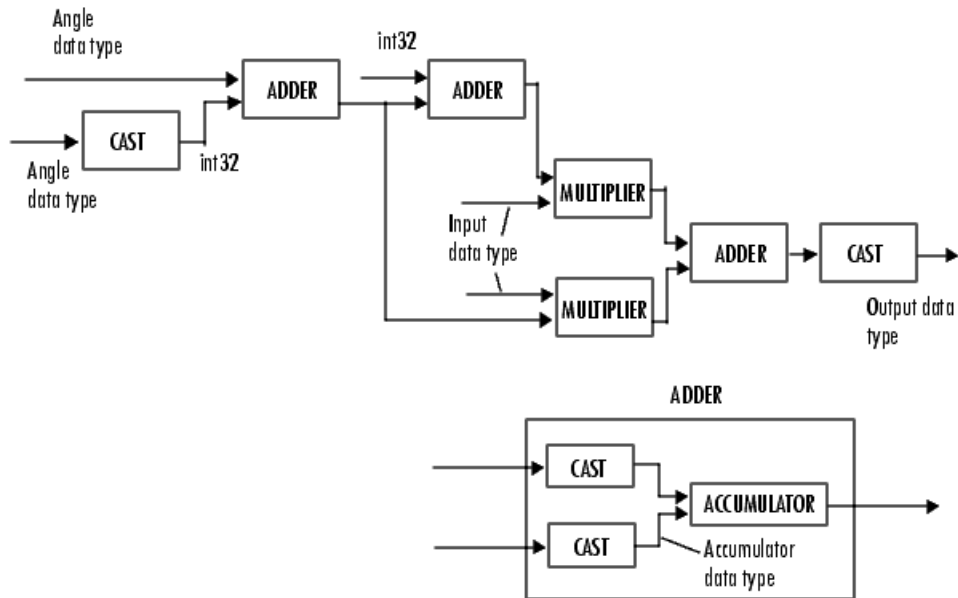
You can set the angle values, product output, accumulator, and output data types in the block mask.

The Rotate block requires additional data types. The Sine table value has the same word length as the angle data type and a fraction length that is equal to its word length minus one. The following diagram shows how these data types are used inside the block.
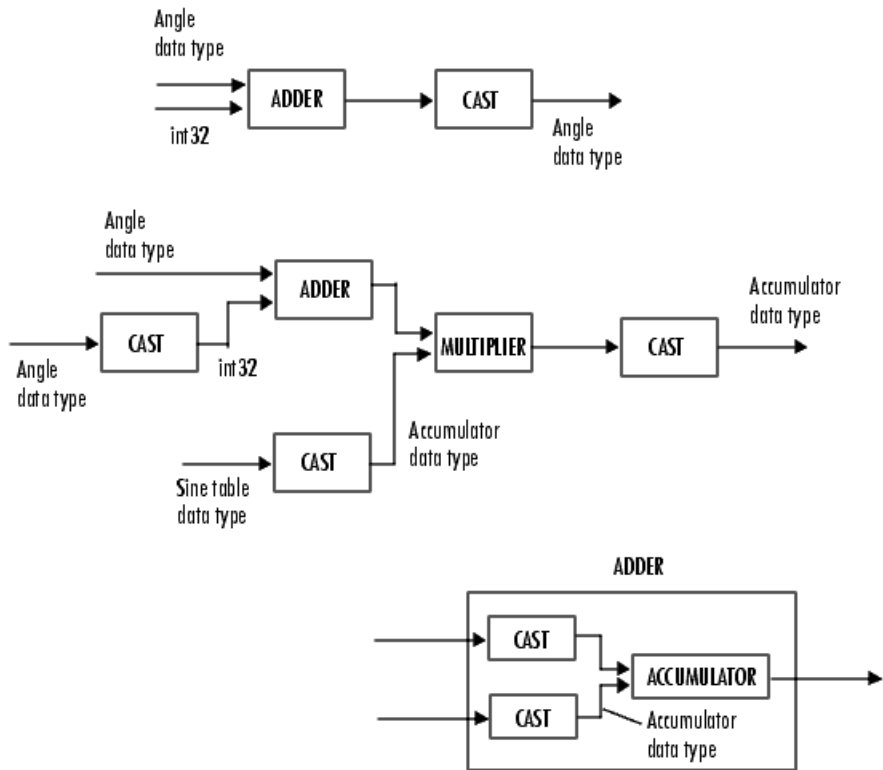
Note If overflow occurs, the rotated image might appear distorted.

Rotate

Dialog Box

The **Main** pane of the Rotate dialog box appears as shown in the following figure.
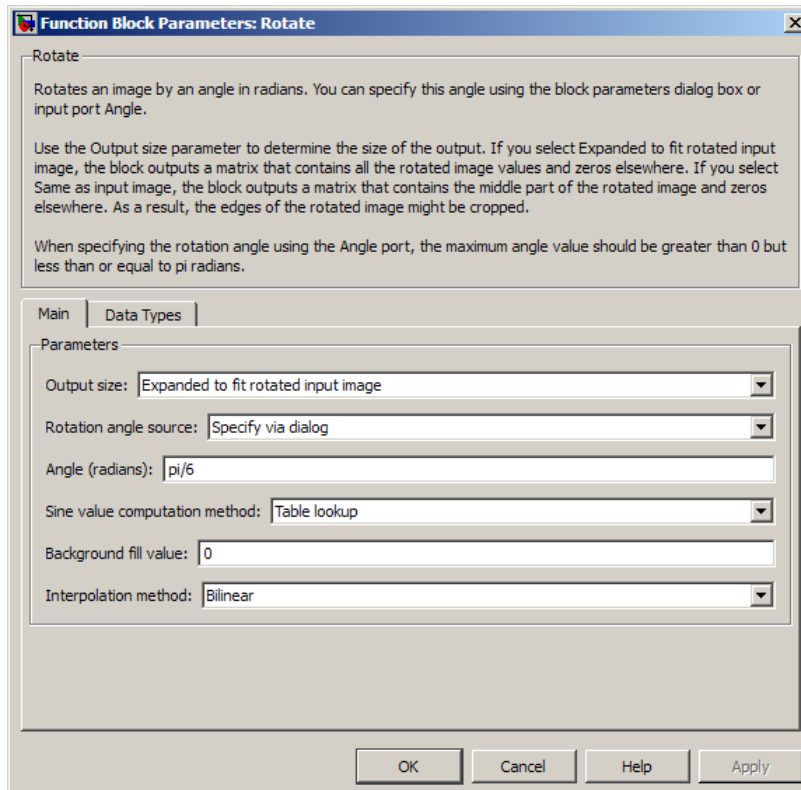
Output size

Specify the size of the rotated matrix. If you select `Expanded to fit rotated input image`, the block outputs a matrix that contains all the rotated image values. If you select `Same as input image`, the block outputs a matrix that contains the middle part of the rotated image. As a result, the edges of the rotated image might be cropped. Use the **Background fill value** parameter to specify the pixel values outside the image.

Rotation angle source

Specify how to enter your rotation angle. If you select `Specify via dialog`, the **Angle (radians)** parameter appears in the dialog box.

If you select `Input port`, the Angle port appears on the block. The block uses the input to this port at each time step as your rotation angle. The input to the Angle port must be the same data type as the input to the **I** port.

Angle (radians)

Enter a real, scalar value for your rotation angle. This parameter is visible if, for the **Rotation angle source** parameter, you select `Specify via dialog`.

When the rotation angle is a multiple of pi/2, the block uses a more efficient algorithm. If the angle value you enter for the **Angle (radians)** parameter is within 0.00001 radians of a multiple of pi/2, the block rounds the angle value to the multiple of pi/2 before performing the rotation.

Maximum angle (enter pi radians to accommodate all positive and negative angles)

Enter the maximum angle by which to rotate the input image. Enter a scalar value, between 0 and π radians. The block determines which angle, $0 \le angle \le \max angle$, requires the largest output matrix and sets the dimensions of the output port accordingly.

This parameter is visible if you set the **Output size** parameter, to `Expanded to fit rotated input image`, and the **Rotation angle source** parameter to `Input port`.

Display rotated image in

Specify how the image is rotated. If you select `Center`, the image is rotated about its center point. If you select `Top-left corner`, the block rotates the image so that two corners of the rotated

input image are always in contact with the top and left sides of
the output image.

This parameter is visible if, for the **Output size** parameter, you
select `Expanded to fit rotated input image`, and, for the
Rotation angle source parameter, you select `Input port`.

Sine value computation method

Specify the value computation method. If you select
`Trigonometric function`, the block computes sine and cosine
values it needs to calculate the rotation of your image during the
simulation. If you select `Table lookup`, the block computes and
stores the trigonometric values it needs to calculate the rotation
of your image before the simulation starts. In this case, the block
requires extra memory.

Background fill value

Specify a value for the pixels that are outside the image.

Interpolation method

Specify which interpolation method the block uses to rotate the
image. If you select `Nearest neighbor`, the block uses the value
of one nearby pixel for the new pixel value. If you select `Bilinear`,
the new pixel value is the weighted average of the four nearest
pixel values. If you select `Bicubic`, the new pixel value is the
weighted average of the sixteen nearest pixel values.

The number of pixels the block considers affects the complexity of
the computation. Therefore, the `Nearest-neighbor` interpolation
is the most computationally efficient. However, because the
accuracy of the method is proportional to the number of pixels
considered, the `Bicubic` method is the most accurate. For more
information, see "Interpolation Methods" in the *Computer Vision
System Toolbox User's Guide*.

The **Data Types** pane of the Rotate dialog box appears as shown in
the following figure.
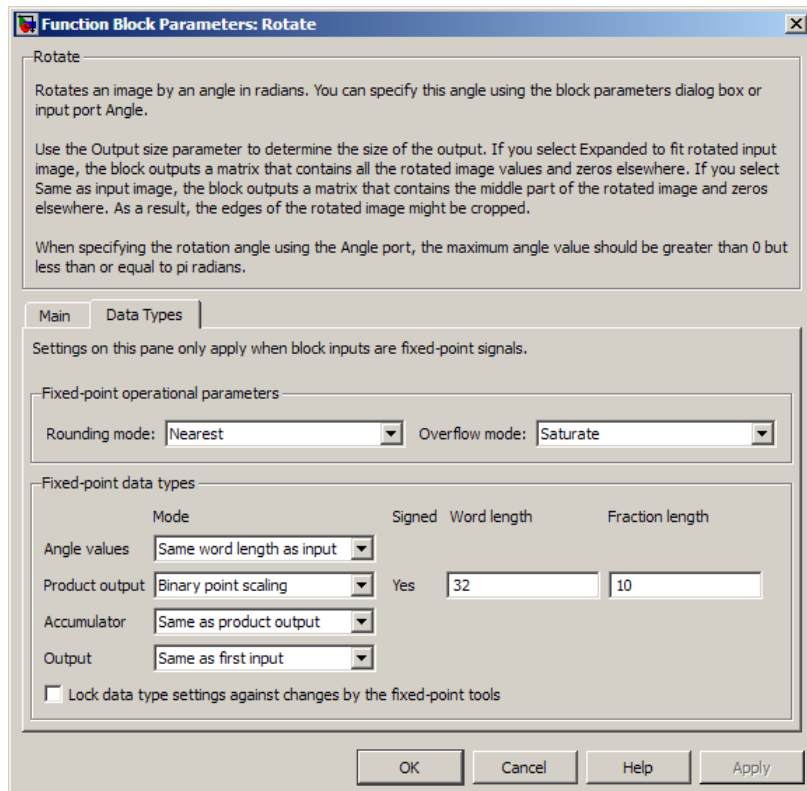
Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Angle values

Choose how to specify the word length and the fraction length of the angle values.

- When you select Same word length as input, the word length of the angle values match that of the input to the block. In this

mode, the fraction length of the angle values is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the angle values.

- When you select Specify word length, you can enter the word length of the angle values, in bits. The block automatically sets the fraction length to give you the best precision.

- When you select Binary point scaling, you can enter the word length and the fraction length of the angle values, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the angle values. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

This parameter is only visible if, for the **Rotation angle source** parameter, you select Specify via dialog.

Product output

As depicted in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

- When you select Same as first input, these characteristics match those of the input to the block.

- When you select Binary point scaling, you can enter the word length and the fraction length of the product output, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select Same as product output, these characteristics match those of the product output.

- When you select Same as first input, these characteristics match those of the first input to the block.

- When you select Binary point scaling, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select Slope and bias scaling, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output
Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Supported Data Types

Port	Supported Data Types
Image	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer
Angle	Same as Image port
Output	Same as Image port

If the data type of the input signal is floating point, the output signal is the same data type as the input signal.

References   [1] Wolberg, George. *Digital Image Warping*. Washington: IEEE Computer Society Press, 1990.

See Also

Resize	Computer Vision System Toolbox software
Translate	Computer Vision System Toolbox software
Shear	Computer Vision System Toolbox software
imrotate	Image Processing Toolbox software

SAD (To Be Removed)

Purpose　　　Perform 2-D sum of absolute differences (SAD)

Library　　　vipobslib

Description

I →
SAD　Idx →
Template →

SAD
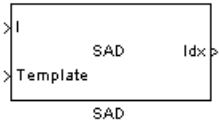
Note　The SAD block is obsolete. It may be removed in a future version of the Computer Vision System Toolboxsoftware. Use the replacement block Template Matching.

See the SAD documentation to view the reference material related to the obsolete block.

Purpose	Shift rows or columns of image by linearly varying offset
Library	Geometric Transformations
	visiongeotforms
Description	The Shear block shifts the rows or columns of an image by a gradually increasing distance left or right or up or down.

Note This block supports intensity and color images on its ports.

Port	Input/Output	Supported Data Types	Complex Values Supported
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
S	Two-element vector that represents the number of pixels by which you want to shift your first and last rows or columns	Same as I port	No
Output	Shifted image	Same as I port	No

If the data type of the input to the I port is floating point, the input to the S port of this block must be the same data type. Also, the block output is the same data type.

Use the **Shear direction** parameter to specify whether you want to shift the rows or columns. If you select Horizontal, the first row

has an offset equal to the first element of the **Row/column shear values [first last]** vector. The following rows have an offset that linearly increases up to the value you enter for the last element of the **Row/column shear values [first last]** vector. If you select `Vertical`, the first column has an offset equal to the first element of the **Row/column shear values [first last]** vector. The following columns have an offset that linearly increases up to the value you enter for the last element of the **Row/column shear values [first last]** vector.

Use the **Output size after shear** parameter to specify the size of the sheared image. If you select `Full`, the block outputs a matrix that contains the entire sheared image. If you select `Same as input image`, the block outputs a matrix that is the same size as the input image and contains the top-left portion of the sheared image. Use the **Background fill value** parameter to specify the pixel values outside the image.

Use the **Shear values source** parameter to specify how to enter your shear parameters. If you select `Specify via dialog`, the **Row/column shear values [first last]** parameter appears in the dialog box. Use this parameter to enter a two-element vector that represents the number of pixels by which you want to shift your first and last rows or columns. For example, if for the **Shear direction** parameter you select `Horizontal` and, for the **Row/column shear values [first last]** parameter, you enter `[50 150]`, the block moves the top-left corner 50 pixels to the right and the bottom left corner of the input image 150 pixels to the right. If you want to move either corner to the left, enter negative values. If for the **Shear direction** parameter you select `Vertical` and, for the **Row/column shear values [first last]** parameter, you enter `[-10 50]`, the block moves the top-left corner 10 pixels up and the top right corner 50 pixels down. If you want to move either corner down, enter positive values.

Use the **Interpolation method** parameter to specify which interpolation method the block uses to shear the image. If you select `Nearest neighbor`, the block uses the value of the nearest pixel for the new pixel value. If you select `Bilinear`, the new pixel value is the weighted average of the two nearest pixel values. If you select `Bicubic`,

the new pixel value is the weighted average of the four nearest pixel values.

The number of pixels the block considers affects the complexity of the computation. Therefore, the nearest-neighbor interpolation is the most computationally efficient. However, because the accuracy of the method is proportional to the number of pixels considered, the bicubic method is the most accurate. For more information, see "Interpolation Methods" in the *Computer Vision System Toolbox User's Guide*.

If, for the **Shear values source** parameter, you select `Input port`, the S port appears on the block. At each time step, the input to the S port must be a two-element vector that represents the number of pixels by which to shift your first and last rows or columns.

If, for the **Output size after shear** parameter, you select `Full`, and for the **Shear values source** parameter, you select `Input port`, the **Maximum shear value** parameter appears in the dialog box. Use this parameter to enter a real, scalar value that represents the maximum number of pixels by which to shear your image. The block uses this parameter to determine the size of the output matrix. If any input to the S port is greater than the absolute value of the **Maximum shear value** parameter, the block saturates to the maximum value.

Fixed-Point Data Types

The following diagram shows the data types used in the Shear block for bilinear interpolation of fixed-point signals.

Shear
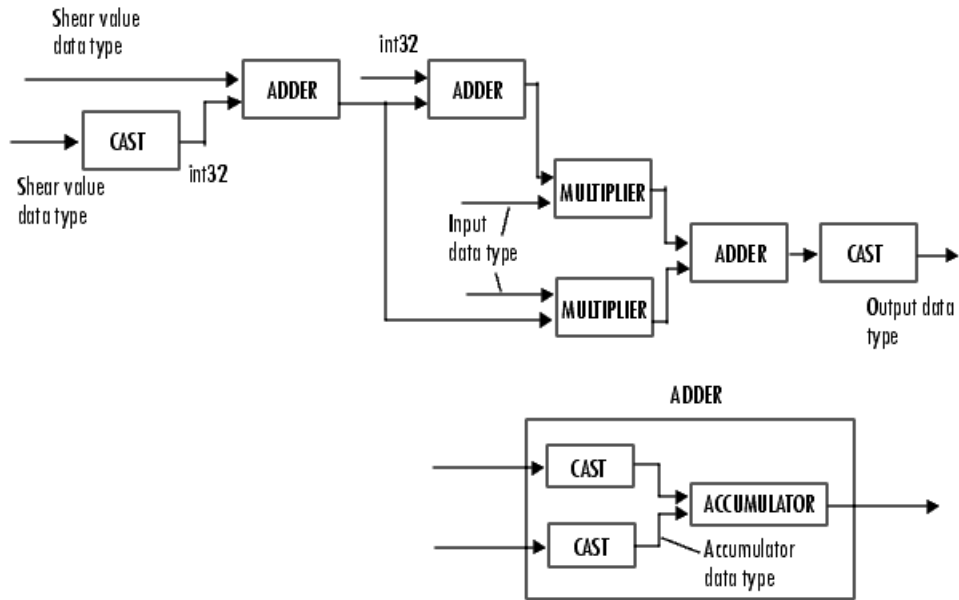
You can set the product output, accumulator, and output data types in the block mask.

Dialog Box

The **Main** pane of the Shear dialog box appears as shown in the following figure.
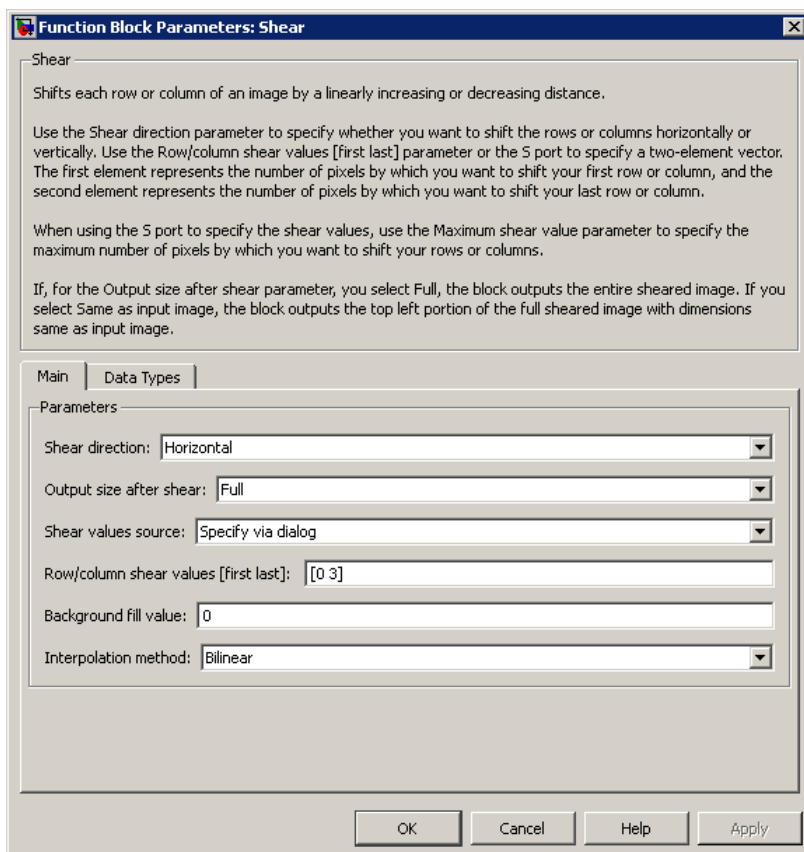
Shear direction

Specify whether you want to shift the rows or columns of the input image. Select Horizontal to linearly increase the offset of the rows. Select Vertical to steadily increase the offset of the columns.

Shear

Output size after shear
Specify the size of the sheared image. If you select `Full`, the block outputs a matrix that contains the sheared image values. If you select `Same as input image`, the block outputs a matrix that is the same size as the input image and contains a portion of the sheared image.

Shear values source
Specify how to enter your shear parameters. If you select `Specify via dialog`, the **Row/column shear values [first last]** parameter appears in the dialog box. If you select `Input port`, port S appears on the block. The block uses the input to this port at each time step as your shear value.

Row/column shear values [first last]
Enter a two-element vector that represents the number of pixels by which to shift your first and last rows or columns. This parameter is visible if, for the **Shear values source** parameter, you select `Specify via dialog`.

Maximum shear value
Enter a real, scalar value that represents the maximum number of pixels by which to shear your image. This parameter is visible if, for the **Shear values source** parameter, you select `Input port`.

Background fill value
Specify a value for the pixels that are outside the image. This parameter is tunable.

Interpolation method
Specify which interpolation method the block uses to shear the image. If you select `Nearest neighbor`, the block uses the value of one nearby pixel for the new pixel value. If you select `Bilinear`, the new pixel value is the weighted average of the two nearest pixel values. If you select `Bicubic`, the new pixel value is the weighted average of the four nearest pixel values.

The **Data Types** pane of the Shear dialog box appears as shown in the following figure.

Shifts each row or column of an image by a linearly increasing or decreasing distance.

Use the Shear direction parameter to specify whether you want to shift the rows or columns horizontally or vertically. Use the Row/column shear values [first last] parameter or the S port to specify a two-element vector. The first element represents the number of pixels by which you want to shift your first row or column, and the second element represents the number of pixels by which you want to shift your last row or column.

When using the S port to specify the shear values, use the Maximum shear value parameter to specify the maximum number of pixels by which you want to shift your rows or columns.

If, for the Output size after shear parameter, you select Full, the block outputs the entire sheared image. If you select Same as input image, the block outputs the top left portion of the full sheared image with dimensions same as input image.

Rounding mode
    Select the rounding mode for fixed-point operations.

Overflow mode
    Select the overflow mode for fixed-point operations.

Shear values
    Choose how to specify the word length and the fraction length of the shear values.

Shear

- When you select `Same word length as input`, the word length of the shear values match that of the input to the block. In this mode, the fraction length of the shear values is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the shear values.

- When you select `Specify word length`, you can enter the word length of the shear values, in bits. The block automatically sets the fraction length to give you the best precision.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the shear values, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the shear values. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

This parameter is visible if, for the **Shear values source** parameter, you select `Specify via dialog`.

Product output

As depicted in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

- When you select `Same as first input`, these characteristics match those of the first input to the block at the I port.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

The result of each addition remains in the accumulator data type.

Input to adder - product output data type

CAST

Accumulator data type

ADDER

Accumulator data type

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block at the I port.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

Shear

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the first input to the block at the I port.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References   [1] Wolberg, George. *Digital Image Warping*. Washington: IEEE Computer Society Press, 1990.

See Also

Resize	Computer Vision System Toolbox software
Rotate	Computer Vision System Toolbox software
Translate	Computer Vision System Toolbox software

Purpose   Locate a template in an image

Library   Analysis & Enhancement

                     visionanalysis

Description
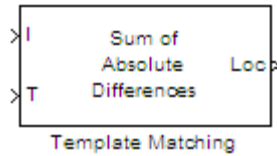
Template Matching

The Template Matching block finds the best match of a template within an input image. The block computes match metric values by shifting a template over a region of interest or the entire image, and then finds the best match location.

Port Description

Port	Supported Data Types
I (Input Image)	• Double-precision floating point
	• Single-precision floating point
	• Fixed point (signed, unsigned or both)
	• Boolean
	• 8-, 16-, and 32-bit signed integers

Port	Supported Data Types
	• 8-, 16-, and 32-bit unsigned integers
T (Template)	• Double-precision floating point • Single-precision floating point • Fixed point (signed, unsigned or both) • Boolean • 8-bit unsigned integers
ROI (Region of Interest, [x y width height])	• Double-precision floating point • Single-precision floating point • Fixed point (signed, unsigned or both) • Boolean • 8-bit unsigned integers

Port	Supported Data Types
Metric (Match Metric Values)	• Double-precision floating point • Single-precision floating point • Fixed point (signed, unsigned or both) • Boolean • 32-bit unsigned integers
Loc (Best match location [x y])	• 32-bit unsigned integers
NMetric (Metric values in Neighborhood of best match)	• Double-precision floating point • Single-precision floating point • Fixed point (signed, unsigned or both) • Boolean • 8-bit unsigned integers
NValid (Neighborhood valid)	• Boolean
ROIValid (ROI valid)	• Boolean

Template Matching

Using the Template Matching Block

Choosing an Output Option

The block outputs either a matrix of match metric values or the zero-based location of the best template match. The block outputs the matrix to the **Metric** port or the location to the **Loc** port. Optionally, the block can output an *NxN* matrix of neighborhood match metric values to the **NMetric** port.

Input and Output Signal Sizes

The Template Matching block does not pad the input data. Therefore, it can only compute values for the match metrics between the input image and the template, where the template is positioned such that it falls entirely on the input image. A set of all such positions of the template is termed as the *valid* region of the input image. The size of the valid region is the difference between the sizes of the input and template images plus one.

$$\text{size}_{\text{valid}} = \text{size}_{\text{input}} - \text{size}_{\text{template}} + 1$$

The output at the **Metric** port for the `Match metric` mode is of the valid image size. The output at the **Loc** port for the `Best match index` mode is a two-element vector of indices relative to the top-left corner of the input image.

The neighborhood metric output at the **NMetric** port is of the size *NxN*, where *N* must be an odd number specified in the block mask.

Defining the Region of Interest (ROI)

To perform template matching on a subregion of the input image, select the **Enable ROI processing** check box. This check box adds the **ROI** input port to the Template Matching block. The ROI processing option is only available for the `Best match index` mode.

The ROI port requires a four-element vector that defines a rectangular area. The first two elements represent [x y] coordinates for the upper-left corner of the region. The second two elements represent the width and height of the ROI. The block outputs the best match location index relative to the top left corner of the input image.

Choosing a Match Metric

The block computes the match metric at each step of the iteration. Choose the match metric that best suits your application. The block calculates the global optimum for the best metric value. It uses the valid subregion of the input image intersected by the ROI, if provided.
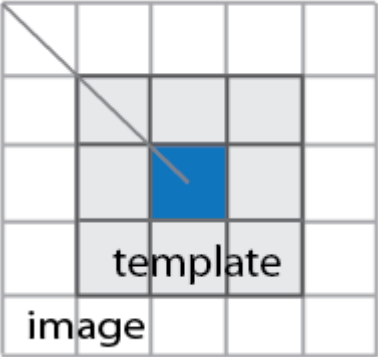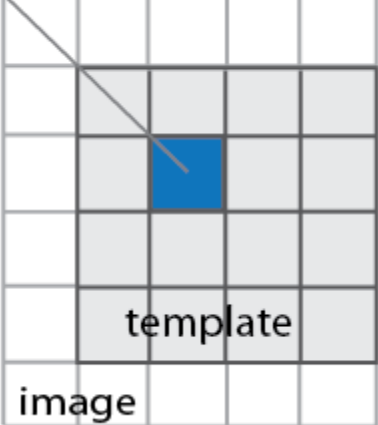
Returning the Matrix of Match Metric Values

The matrix of the match metrics always implements single-step exhaustive window iteration. Therefore, the block computes the metric values at every pixel.

Returning the Best Match Location

When in the ROI processing mode, the block treats the image around the ROI as an extension of the ROI subregion. Therefore, it computes the best match locations true to the actual boundaries of the ROI. The block outputs the best match coordinates, relative to the top-left corner of the image. The one-based [x y] coordinates of the location correspond to the center of the template. The following table shows how the block outputs the center coordinates for odd and even templates:

Template Matching

Odd number of pixels in template	Even number of pixels in template

Returning the Neighborhood Match Metric around the Best Match

When you select **Best match location** to return the matrix of metrics in a neighborhood around the best match, an exhaustive loop computes all the metric values for the *N*-by-*N* neighborhood. This output is particularly useful for performing template matching with subpixel accuracy.

Choosing a Search Method

When you select Best match location as the output option, you can choose to use either Exhaustive or Three-step search methods.

The Exhaustive search method is computationally intensive because it searches at every pixel location of the image. However, this method provides a more precise result.

The `Three-step` search method is a fast search that uses a neighborhood approach, which does not inspect every pixel. The search starts with a step size equal to or slightly greater than half of the maximum search range and then employs the following steps:

1 The block compares nine search points in each step. There is a central point and eight search points located on the search area boundary.

2 The block decrements the step size by one, after each step, ending the search with a step size of one pixel.

3 At each new step, the block moves the search center to the best matching point resulting from the previous step. The number one blue circles in the figure below represent a search with a starting step size of three. The number two green circles represent the next search, with step size of two, centered around the best match found from the previous search. Finally, the number three orange circles represent the final search, with step size of one, centered around the previous best match.

Template Matching
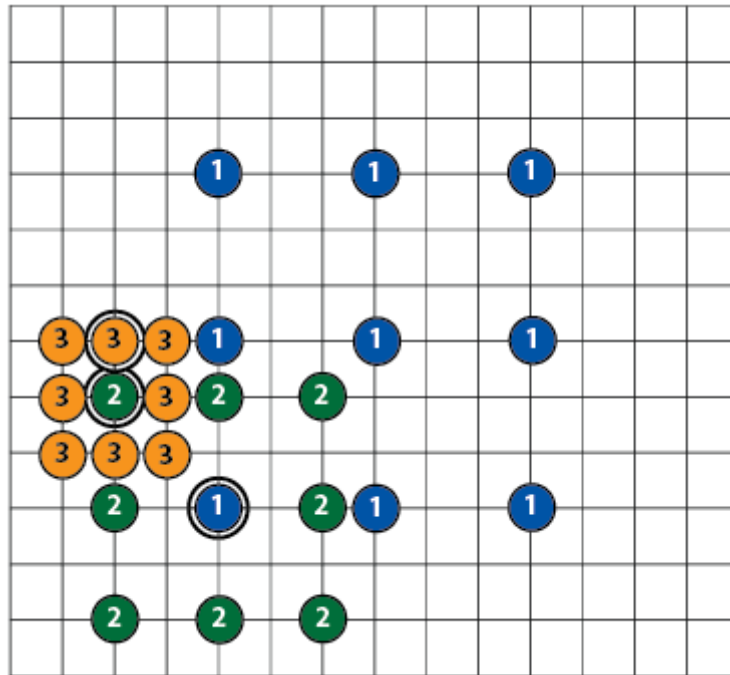
Three-Step Search

Using the ROIValid and NValid flags for Diagnostics

The **ROIValid** and **NValid** ports represent boolean flags, which track the valid Region of Interest (ROI) and neighborhood. You can use these flags to communicate with the downstream blocks and operations.

Valid Region of Interest

If you select the **Output flag indicating if ROI is valid** check box, the block adds the **ROIValid** port. If the ROI lies partially outside the valid image, the block only processes the intersection of the ROI and the valid image. The block sets the ROI flag output to this port as follows:

- True, set to 1 indicating the ROI lies completely inside the valid part of the input image.

- False, set to 0 indicating the ROI lies completely or partially outside of the valid part of the input image.

Valid Neighborhood

The neighborhood matrix of metric values is valid inside of the Region of Interest (ROI). You can use the Boolean flag at the **NValid** port to track the valid neighborhood region. The block sets the neighborhood **NValid** boolean flag output as follows:

- True, set to 1 indicating that the neighborhood containing the best match is completely inside the region of interest.

- False, set to 0 indicating that the neighborhood containing the best match is completely or partially outside of the region of interest.

Algorithm   The match metrics use a difference equation with general form:

$$d_p(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$$

l_n^p denotes the metric space (R^n, d_p) for R^n $n > 1$.

- **Sum of Absolute Differences (SAD)**
  This metric is also known as the *Taxicab* or *Manhattan Distance* metric. It sums the absolute values of the differences between pixels in the original image and the corresponding pixels in the template image. This metric is the l^1 norm of the difference image. The lowest SAD score estimates the best position of template within the search image. The general SAD distance metric becomes:

$$d_1(I_j, T) = \sum_{i=1}^{n} |I_{i,j} - T_i|$$

- **Sum of Squared Differences (SSD)**
  This metric is also known as the *Euclidean Distance* metric. It sums the square of the absolute differences between pixels in the original image and the corresponding pixels in the template image. This metric is the square of the l^2 norm of the difference image. The general SSD distance metric becomes:

$$d_2(I_j, T) = \sum_{i=1}^{n} \left| I_{i,j} - T_i \right|^2$$

- **Maximum Absolute Difference (MaxAD)**
  This metric is also known as the *Uniform Distance* metric. It sums the maximum of absolute values of the differences between pixels in the original image and the corresponding pixels in the template image. This distance metric provides the l^{∞} norm of the difference image. The general MaxAD distance metric becomes:

$$d_{\infty}(I_j, T) = \lim_{x \to \infty} \sum_{i=1}^{n} \left| I_{i,j} - T_i \right|^p$$

which simplifies to:

$$d_{\infty}(I_j, T) = \max_{i}^{n} \left| I_{i,j} - T_i \right|^p$$

Main Dialog Box

The **Main** pane of the Template Matching block appears as shown in the following figure.

Function Block Parameters: Template Matching

Template Matching

Perform template matching by shifting the template in single-pixel increments through image I.

You can use the ROI port to define a region of interest (ROI) over which you want to p matching. The format of the ROI port is [x y width heigh], where [x y] are one-based upper left corner of a rectangle.

The block outputs either the match metric values at the Metric port or the one-based [of the best template match at the Loc port. Optionally, the block can output an N-by-N match metric values centered around the best match location at the NMetric port.

| Main | Data Types |

Parameters

Match metric: Sum of absolute differences

Output options

Output: Best match location

  Search method: Exhaustive

  ☐ Output NxN matrix of metric values around best match

ROI Processing

☐ Enable ROI processing

OK   Cancel   He

Match metric

Select one of three types of match metrics:

- `Sum of absolute differences` (SAD)
- `Sum of squared differences` (SSD)
- `Maximum absolute difference` (MaxAD)

Output

Select one of two output types:

- `Metric matrix`
  Select this option to output the match metric matrix. This option adds the **Metric** output port to the block.

- `Best match location`
  Select this option to output the [x y] coordinates for the location of the best match. This option adds the **Loc** output port to the block. When you select `Best match location`, the **Search method**, **Output NxN matrix of metric values around best match**, and **Enable ROI processing** parameter options appear.

Search method

This option appears when you select `Best match location` for the **Output** parameter. Select one of two search methods.

- `Exhaustive`
- `Three-step`

Output NxN matrix of metric values around best match

This option appears when you select `Best match location` for the **Output** parameter. Select the check box to output a matrix of metric values centered around the best match. When you do so, the block adds the **NMetric** and **NValid** output ports.

N

This option appears when you select the **Output NxN matrix of metric values around best match** check box. Enter an integer number that determines the size of the N-by-N output matrix centered around the best match location index. N must be an odd number.

Enable ROI processing

This option appears when you select `Best match location` for the **Output** parameter. Select the check box for the Template Matching block to perform region of interest processing. When you do so, the block adds the **ROI** input port, and the **Output flag indicating if ROI is valid** check box appears. The **ROI** input must have the format [x y width height], where [x y] are the coordinates of the upper-left corner of the ROI.

Output flag indicating if ROI is valid

This option appears when you select the **Enable ROI processing** check box. Select the check box for the Template Matching block to indicate whether the ROI is within the valid region of the image boundary. When you do so, the block adds the **ROIValid** output port.

Template Matching

Data Types Dialog Box

The **Data Types** pane of the Template Matching block dialog box appears as shown in the following figure.
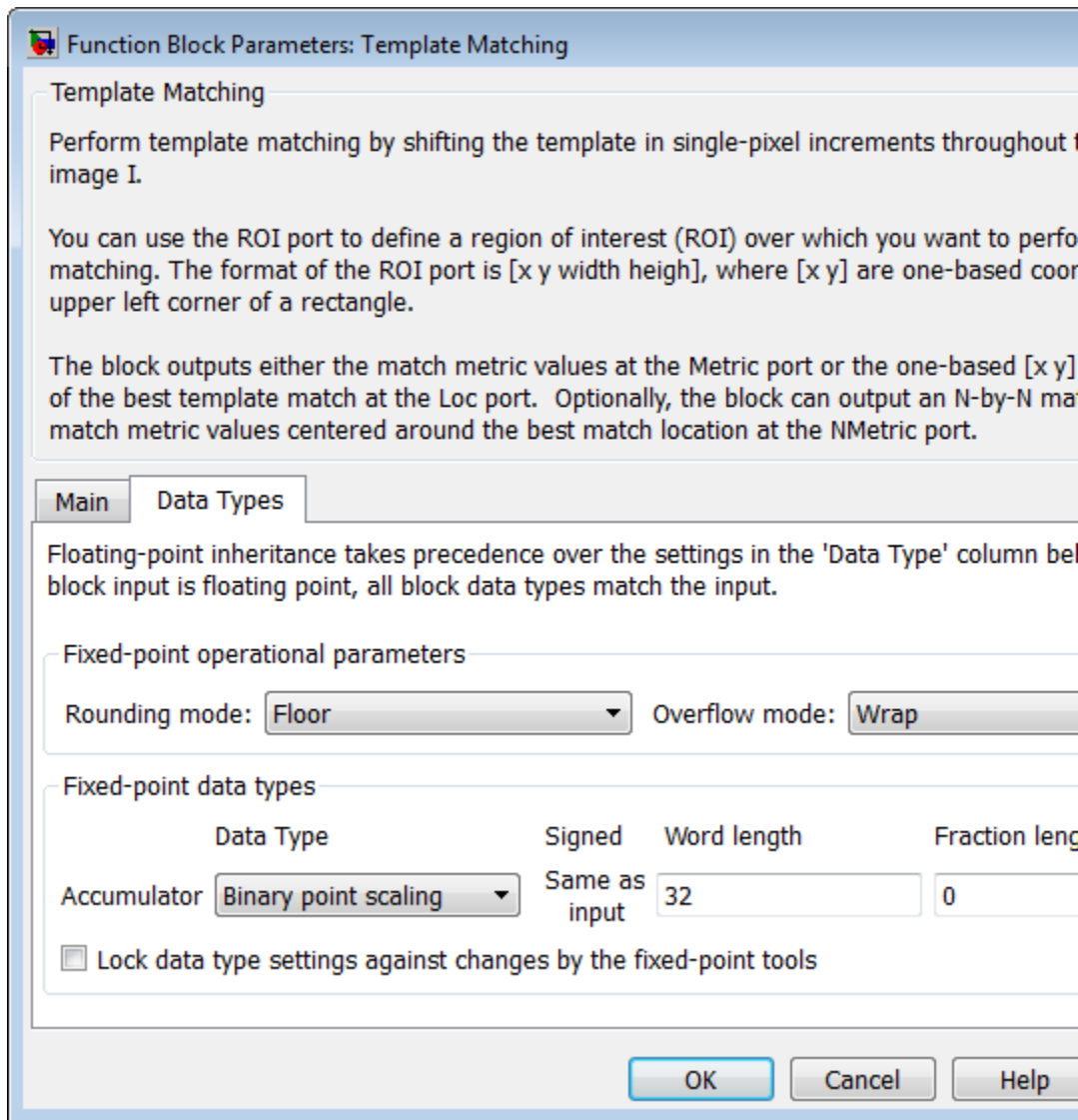
Rounding mode

Select the "Rounding Modes" for fixed-point operations.

Overflow mode

Select the Overflow mode for fixed-point operations.

- `Wrap`

- `Saturate`

Product output

- Use this parameter to specify how to designate the product output word and fraction lengths. Refer to "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

  - When you select `Same as input`, these characteristics match those of the input to the block.

  - When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

  - When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths.

- When you select `Same as product output` the characteristics match the characteristics of the product output. See "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

  When you select `Binary point scaling`, you can enter the **Word length** and the **Fraction length** of the accumulator, in bits.

When you select `Slope and bias scaling`, you can enter the **Word length**, in bits, and the **Slope** of the **Accumulator**. All signals in the Computer Vision System Toolbox software have a bias of 0.

The block casts inputs to the **Accumulator** to the accumulator data type. It adds each element of the input to the output of the adder, which remains in the accumulator data type. Use this parameter to specify how to designate this accumulator word and fraction lengths.

Output

Choose how to specify the **Word length**, **Fraction length** and **Slope** of the **Template Matching** output:

- When you select `Same as first input`, these characteristics match the characteristics of the accumulator.

- When you select `Binary point scaling`, you can enter the **Word length** and **Fraction length** of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the **Word length**, in bits, and the **Slope** of the output. All signals in the Computer Vision System Toolbox software have a bias of 0.

The **Output** parameter on the Data Types pane appears when you select `Metric matrix` or if you select `Best match location` and the **Output NxN matrix of metric values around best match** check box is selected.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

Reference [1] Koga T., et. Al. Motion-compensated interframe coding for video conferencing. In National Telecommunications Conference. Nov. 1981, G5.3.1–5, New Orleans, LA.

[2] Zakai M., "General distance criteria" *IEEE Transaction on Information Theory*, pp. 94–95, January 1964.

[3] Yu, J., J. Amores, N. Sebe, Q. Tian, "A New Study on Distance Metrics as Similarity Measurement" IEEE International Conference on Multimedia and Expo, 2006 .

See Also

Block Matching	Image Processing Toolbox
Video Stabilization	Computer Vision System Toolbox

Template Matching (To Be Removed)

Purpose   Locate a template in an image

Library   Analysis & Enhancement

Description
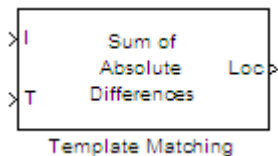
Note This Template Matching block will be removed in a future
release. It uses the zero-based, [row column] coordinate system. It is
recommended that you replace this block with the updated Template
Matching block that uses the one-based, [x y] coordinate system.

Refer to "Conventions Changed for Indexing, Spatial Coordinates, and
Representation of Geometric Transforms" in the R2011b Release Notes
for details related to these changes.

See the Template Matching documentation to view the reference
material related to the obsolete block.

Purpose   Write video frames and audio samples to multimedia file

Library   Sinks

visionsinks

Description

The To Multimedia File block writes video frames, audio samples, or both to a multimedia (.avi, .wav, .wma, or .wmv) file.

You can compress the video frames or audio samples by selecting a compression algorithm. You can connect as many of the input ports as you want. Therefore, you can control the type of video and/or audio the multimedia file receives.

Note This block supports code generation for platforms that have file I/O available. You cannot use this block with Real-Time Windows Target software, because that product does not support file I/O.

This block performs best on platforms with Version 11 or later of Windows Media® Player software. This block supports only uncompressed RGB24 AVI files on Linux and Mac platforms.

Windows 7 UAC (User Account Control), may require administrative privileges to encode WMV and WMA files.

The generated code for this block relies on prebuilt library files. You can run this code outside the MATLAB environment, or redeploy it, but be sure to account for these extra library files when doing so. The packNGo function creates a single zip file containing all of the pieces required to run or rebuild this code. See packNGo for more information.

To Multimedia File

To run an executable file that was generated from a model containing this block, you may need to add precompiled shared library files to your system path. See "Simulink Coder", "Shared Library Dependencies", and "Accelerating Simulink Models" for details.

This block allows you to write WMA/WMV streams to disk or across a network connection. Similarly, the From Multimedia File block allows you to read WMA/WMV streams to disk or across a network connection. If you want to play an MP3/MP4 file in Simulink, but you do not have the codecs, you can re-encode the file as WMA/WMV, which are supported by the Computer Vision System Toolbox.

Ports

Port	Description
Image	*M*-by-*N*-by-3 matrix RGB, Intensity, or YCbCr 4:2:2 signal.
R, G, B	Matrix that represents one plane of the RGB video stream. Inputs to the R, G, or B port must have the same dimensions and data type.
Audio	Vector of audio data
Y, Cb, Cr	Matrix that represents one frame of the YCbCr video stream. The Y, Cb, and Cr ports use the following dimensions: Y: M x N Cb: M x $\dfrac{N}{2}$ Cr: M x $\dfrac{N}{2}$

Dialog Box

The **Main** pane of the To Multimedia File block dialog appears as follows.
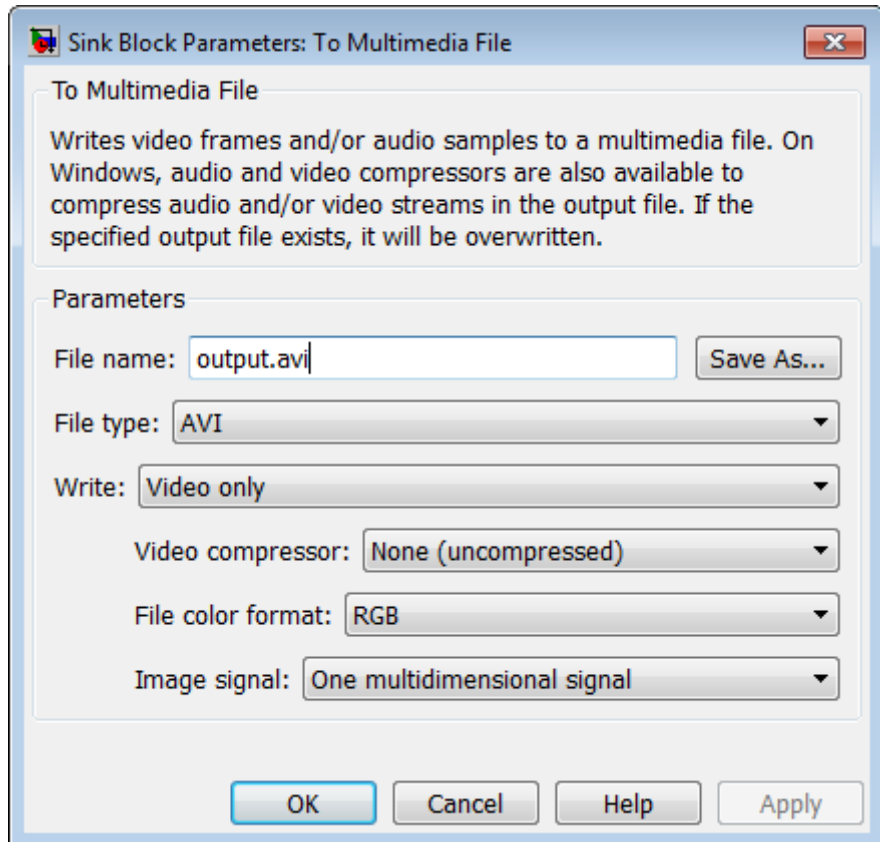
File name
Specify the name of the multimedia file. The block saves the file in your current folder. To specify a different file or location, click the **Save As...** button.

To Multimedia File

File type

> Specify the file type of the multimedia file. You can select avi, wav, wma, or wmv.

Write

> Specify whether the block writes video frames, audio samples, or both to the multimedia file. You can select Video and audio, Video only, or Audio only.

Audio compressor

> Select the type of compression algorithm to use to compress the audio data. This compression reduces the size of the multimedia file. Choose None (uncompressed) to save uncompressed audio data to the multimedia file.

> **Note** The other items available in this parameter list are the audio compression algorithms installed on your system. For information about a specific audio compressor, see the documentation for that compressor.

Audio data type

> Select the audio data type. You can use the **Audio data type** parameter only for uncompressed wave files.

Video compressor

> Select the type of compression algorithm to use to compress the video data. This compression reduces the size of the multimedia file. Choose None (uncompressed) to save uncompressed video data to the multimedia file.

> **Note** The other items available in this parameter list are the video compression algorithms installed on your system. For information about a specific video compressor, see the documentation for that compressor.

File color format

> Select the color format of the data stored in the file. You can select either RGB or YCbCr 4:2:2.

Image signal

> Specify how the block accepts a color video signal. If you select One multidimensional signal, the block accepts an *M*-by-*N*-by-*P* color video signal, where *P* is the number of color planes, at one port. If you select Separate color signals, additional ports appear on the block. Each port accepts one *M*-by-*N* plane of an RGB video stream.

Supported Data Types

For the block to display video data properly, double- and single-precision floating-point pixel values must be between 0 and 1. Any other data type requires the pixel values between the minimum and maximum values supported by their data type.

Check the specific codecs you are using for supported audio rates.

Port	Supported Data Types	Supports Complex Values?
Image	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16- 32-bit signed integers • 8-, 16- 32-bit unsigned integers	No
R, G, B	Same as Image port	No

To Multimedia File

Port	Supported Data Types	Supports Complex Values?
Audio	• Double-precision floating point • Single-precision floating point • 16-bit signed integers • 32-bit signed integers • 8-bit unsigned integers	No
Y, Cb, Cr	Same as Image port	No

See Also   From Multimedia File   Computer Vision System Toolbox

Purpose   Display video data

Library   Sinks

visionsinks

Description

To Video Display

The To Video Display block sends video data to your computer screen. It provides a Windows only, lightweight, high performance, simple display, which accepts RGB and YCbCr formatted images. This block also generates code.

Note This block supports code generation and is only available on Windows platforms with available file I/O. This excludes Real-Time Windows Target (RTWin). This block performs best on platforms with DirectX Version 9.0 or later and Windows Media Version 9.0 or later.

The generated code for this block relies on prebuilt .dll files. You can run this code outside the MATLAB environment, or redeploy it, but be sure to account for these extra .dll files when doing so. The packNGo function creates a single zip file containing all of the pieces required to run or rebuild this code. See packNGo for more information.

To run an executable file that was generated from a model containing this block, you may need to add precompiled shared library files to your system path. See "Simulink Coder", "Shared Library Dependencies", and "Accelerating Simulink Models" for details.

To Video Display

For the block to display video data properly, double- and single-precision floating-point pixel values must be from 0 to 1. For any other data type, the pixel values must be between the minimum and maximum values supported by their data type.

You can set the display for full screen, normal or, to maintain one-to-one size. When you save the model, the size and position of the display window is saved. Any changes while working with the model should be saved again in order that these preferences are maintained when you run the model. The minimum display width of the window varies depending on your system's font size settings.

This block runs in real-time, and may limit the speed at which video data displays. It operates and renders video data to the display independent of the Simulink model. This design allows buffering to the display, which increases performance. You may notice, the block lags the model by one or more frames.

Rapid Accelerator

When you set your model to run in "Accelerator Mode", and do not select the **Open at Start of Simulation** option, the block will not be included during the run, and therefore the video display will not be visible. For **Rapid Accelerator** mode, menu preferences are saved only when the model is compiled. To change any of the menu options, change the model to run in "Normal Mode", and re-save it. You can then run in Rapid Accelerator mode with the new preferences.

Menu Options

The To Video Display block provides menu options to modify viewing preferences for the display. If however, your model is set to run in "Accelerator Mode", the menu options will not be available.

View menu

Window Size

Select **Full-screen** mode to display your video stream in a full screen window. To exit or return to other applications from full-screen, select the display and use the **Esc** key.

Select **True Size (1:1)** mode to display a one-to-one pixel ratio of input image at simulation start time. The block displays the same information per pixel and does not change size from the input size. You can change the display size after the model starts.

Select **Normal** mode to modify display size at simulation start time.

Open at Start of Simulation

Select `Open at Start of Simulation` from the **View** menu for the display window to appear while running the model. If not selected, you can double click the block to display the window.

Settings menu

Input Color Format

Select the color format of the data stored in the input image.

Select **RGB** for the block to accept a matrix that represents one plane of the RGB video stream. Inputs to the **R**, **G**, or **B** ports must have the same dimension and data type.

Select **YCbCr 4:2:2** for the block to accept a matrix that represents one frame of the YCbCr video stream. The **Y** port accepts an *M*-by-*N* matrix. The **Cb** and **Cr** ports accepts an

M-by-$\dfrac{N}{2}$ matrix.

Image Signal

Specify how the block accepts a color video signal.

Select **One multidimensional signal**, for the block to accept an *M*-by-*N*-by-3 color video signal at one port.

Select **Separate color signals**, for additional ports to appear on the block. Each port accepts one *M*-by-*N* plane of an RGB video stream.

Preferences

Set any preferences available for the Computer Vision System Toolbox. You can set your model to use hardware acceleration, by selecting the **Use hardware acceleration** checkbox in the

To Video Display

Preferences window. To access the preferences, select **Settings > Preferences** . Then select **Computer Vision** from the preferences list.

Help menu

Help

Select **To Video Display** for reference documentation for the block.

Select **Computer Vision System Toolbox** to navigate to the Computer Vision System Toolbox product page.

Select **About Computer Vision System Toolbox** for version information.

Supported Data Types

Port	Supported Data Types
Image	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16, and 32-bit signed integer • 8-, 16, and 32-bit unsigned integer
R, G, B	Same as Image port
YCbCr 4:2:2	Same as Image ports

See Also

Frame Rate Display	Computer Vision System Toolbox software
From Multimedia File	Computer Vision System Toolbox software
To Multimedia File	Computer Vision System Toolbox software
Video To Workspace	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software

Top-hat

Purpose   Perform top-hat filtering on intensity or binary images

Library   Morphological Operations

visionmorphops

Description   The Top-hat block performs top-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element. Top-hat filtering is the equivalent of subtracting the result of performing a morphological opening operation on the input image from the input image itself. This block uses flat structuring elements only.

Port	Input/Output	Supported Data Types	Complex Values Supported
I	Vector or matrix of intensity values	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer	No
Nhood	Matrix or vector of 1s and 0s that represents the neighborhood values	Boolean	No
Output	Scalar, vector, or matrix that represents the filtered image	Same as I port	No

If your input image is a binary image, for the **Input image type** parameter, select Binary. If your input image is an intensity image, select Intensity.

Use the **Neighborhood or structuring element source** parameter to specify how to enter your neighborhood or structuring element values. If you select Specify via dialog, the **Neighborhood or structuring element** parameter appears in the dialog box. If you select Input port, the Nhood port appears on the block. Use this port to enter your neighborhood values as a matrix or vector of 1s and 0s. Choose your structuring element so that it matches the shapes you want to remove from your image. You can only specify a it using the dialog box.

Use the **Neighborhood or structuring element** parameter to define the region the block moves throughout the image. Specify a neighborhood by entering a matrix or vector of 1s and 0s. Specify a structuring element with the strel function from the Image Processing Toolbox. If the structuring element is decomposable into smaller elements, the block executes at higher speeds due to the use of a more efficient algorithm.

Top-hat

Dialog Box

The Top-hat dialog box appears as shown in the following figure.

Input image type

If your input image is a binary image, select Binary. If your input image is an intensity image, select Intensity.

Neighborhood or structuring element source

Specify how to enter your neighborhood or structuring element values. Select Specify via dialog to enter the values in the dialog box. Select Input port to use the Nhood port to specify the neighborhood values. You can only specify a structuring element using the dialog box.

Neighborhood or structuring element

If you are specifying a neighborhood, this parameter must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the strel function from the Image Processing Toolbox. This parameter is visible if, for the **Neighborhood or**

structuring element source parameter, you select Specify via dialog.

See Also

Bottom-hat	Computer Vision System Toolbox software
Closing	Computer Vision System Toolbox software
Dilation	Computer Vision System Toolbox software
Erosion	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
Opening	Computer Vision System Toolbox software
imtophat	Image Processing Toolbox software
strel	Image Processing Toolbox software

Trace Boundaries (To Be Removed)

Purpose   Trace object boundaries in binary images

Library   Analysis & Enhancement

Description

> **Note** This Trace Boundaries block will be removed in a future release. It uses the zero-based, [row column] coordinate system. It is recommended that you replace this block with the updated Trace Boundary block that uses the one-based, [x y] coordinate system.
>
> Refer to "Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms" in the R2011b Release Notes for details related to these changes.
>
> See the Trace Boundaries documentation to view the reference material related to the obsolete block.

Purpose   Trace object boundaries in binary images

Library   Analysis & Enhancement

visionanalysis
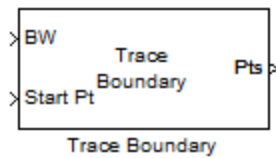
Description

The Trace Boundary block traces object boundaries in binary images, where nonzero pixels represent objects and 0 pixels represent the background.

Port Descriptions

Port	Input/Output	Supported Data Types
BW	Vector or matrix that represents a binary image	Boolean
Start Pt	One-based [x y] coordinates of the boundary starting point.	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integer • 8-, 16-, and 32-bit unsigned integer
Pts	*M*-by-2 matrix of [x y] coordinates of the boundary points, where *M* represents the number of traced boundary pixels. *M* must be less than or equal to the value specified by the **Maximum number**	Same as Start Pts port

Port	Input/Output	Supported Data Types
	of boundary pixels parameter. $$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ . & . \\ . & . \\ . & . \\ x_m & y_m \end{bmatrix}$$	

Dialog Box

The Trace Boundary dialog box appears as shown in the following figure.
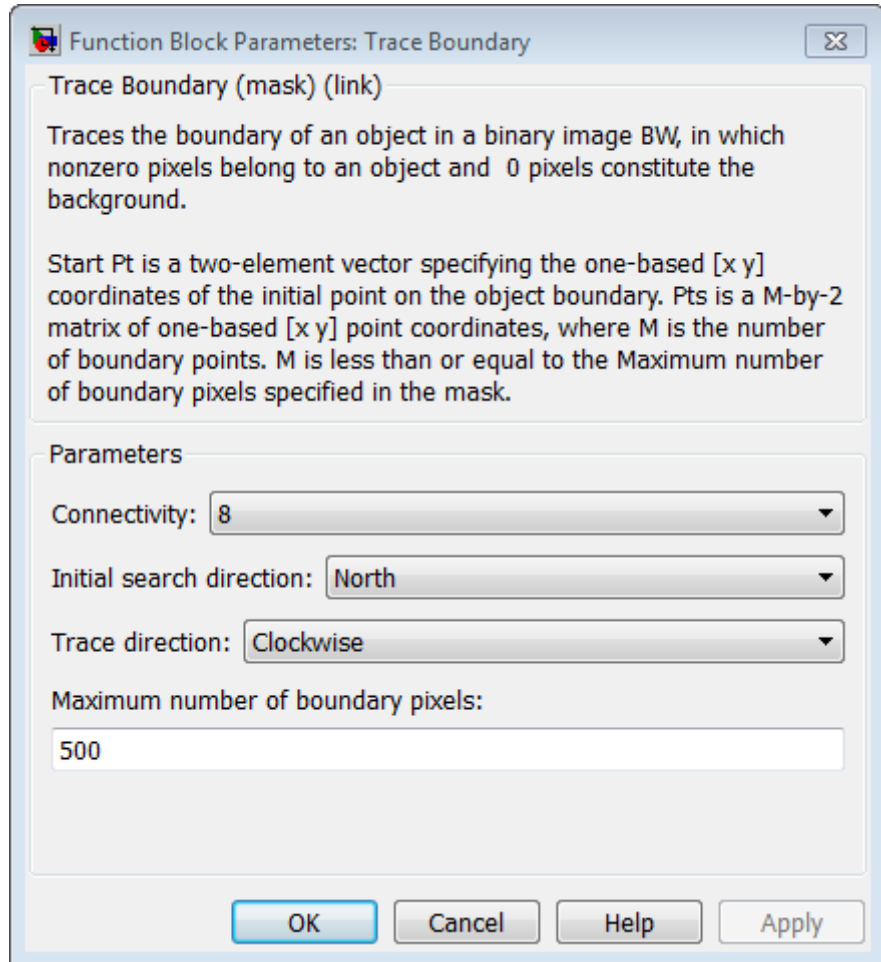
> **Function Block Parameters: Trace Boundary**
>
> **Trace Boundary (mask) (link)**
>
> Traces the boundary of an object in a binary image BW, in which nonzero pixels belong to an object and 0 pixels constitute the background.
>
> Start Pt is a two-element vector specifying the one-based [x y] coordinates of the initial point on the object boundary. Pts is a M-by-2 matrix of one-based [x y] point coordinates, where M is the number of boundary points. M is less than or equal to the Maximum number of boundary pixels specified in the mask.
>
> **Parameters**
>
> Connectivity: 8
>
> Initial search direction: North
>
> Trace direction: Clockwise
>
> Maximum number of boundary pixels:
>
> 500
>
> OK   Cancel   Help   Apply

Connectivity

Specify which pixels are connected to each other. If you want a pixel to be connected to the pixels on the top, bottom, left, and right, select 4. If you want a pixel to be connected to the pixels

on the top, bottom, left, right, and diagonally, select 8. For more information about this parameter, see the Label block reference page.

Initial search direction

Specify the first direction in which to look to find the next boundary pixel that is connected to the starting pixel.

If, for the **Connectivity** parameter, you select 4, the following figure illustrates the four possible initial search directions:
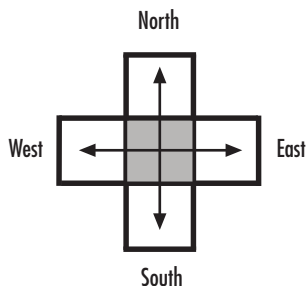
If, for the **Connectivity** parameter, you select 8, the following figure illustrates the eight possible initial search directions:
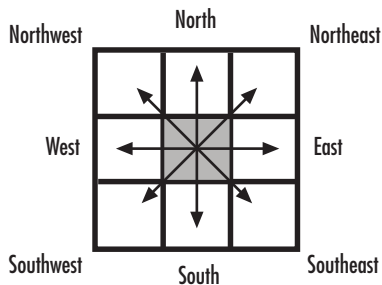
Trace direction

Specify the direction in which to trace the boundary. Your choices are Clockwise or Counterclockwise.

Maximum number of boundary pixels

Specify the maximum number of boundary pixels for each starting point. The block uses this value to preallocate the number of rows of the **Pts** port output matrix so that it can hold all the boundary pixel location values.

Use the **Maximum number of boundary pixels** parameter to specify the maximum number of boundary pixels for the starting point.

See Also

Edge Detection	Computer Vision System Toolbox software
Label	Computer Vision System Toolbox software
bwboundaries	Image Processing Toolbox software
bwtraceboundary	Image Processing Toolbox software

Translate

Purpose   Translate image in 2-D plane using displacement vector

Library   Geometric Transformations

visiongeotforms

Description   Use the Translate block to move an image in a two-dimensional plane using a displacement vector, a two-element vector that represents the number of pixels by which you want to translate your image. The block outputs the image produced as the result of the translation.

> **Note** This block supports intensity and color images on its ports.

Port	Input/Output	Supported Data Types	Complex Values Supported
Image / Input	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
Offset	Vector of values that represent the number of pixels by which to translate the image	Same as I port	No
Output	Translated image	Same as I port	No

The input to the Offset port must be the same data type as the input to the Image port. The output is the same data type as the input to the Image port.

Use the **Output size after translation** parameter to specify the size of the translated image. If you select Full, the block outputs a matrix that contains the entire translated image. If you select Same as input image, the block outputs a matrix that is the same size as the input image and contains a portion of the translated image. Use the **Background fill value** parameter to specify the pixel values outside the image.

Use the **Translation values source** parameter to specify how to enter your displacement vector. If you select Specify via dialog, the **Offset** parameter appears in the dialog box. Use it to enter your displacement vector, a two-element vector, [r c], of real, integer values that represent the number of pixels by which you want to translate your image. The r value represents how many pixels up or down to shift your image. The c value represents how many pixels left or right to shift your image. The axis origin is the top-left corner of your image. For example, if you enter [2.5 3.2], the block moves the image 2.5 pixels downward and 3.2 pixels to the right of its original location. When the displacement vector contains fractional values, the block uses interpolation to compute the output.

Use the **Interpolation method** parameter to specify which interpolation method the block uses to translate the image. If you translate your image in either the horizontal or vertical direction and you select Nearest neighbor, the block uses the value of the nearest pixel for the new pixel value. If you translate your image in either the horizontal or vertical direction and you select Bilinear, the new pixel value is the weighted average of the four nearest pixel values. If you translate your image in either the horizontal or vertical direction and you select Bicubic, the new pixel value is the weighted average of the sixteen nearest pixel values.

The number of pixels the block considers affects the complexity of the computation. Therefore, the nearest-neighbor interpolation is the most computationally efficient. However, because the accuracy of the method is roughly proportional to the number of pixels considered, the bicubic method is the most accurate. For more information, see "Interpolation Methods" in the *Computer Vision System Toolbox User's Guide*.

Translate

If, for the **Output size after translation** parameter, you select `Full`, and for the **Translation values source** parameter, you select `Input port`, the **Maximum offset** parameter appears in the dialog box. Use the **Maximum offset** parameter to enter a two-element vector of real, scalar values that represent the maximum number of pixels by which you want to translate your image. The block uses this parameter to determine the size of the output matrix. If the input to the Offset port is greater than the **Maximum offset** parameter values, the block saturates to the maximum values.

If, for the **Translation values source** parameter, you select `Input port`, the Offset port appears on the block. At each time step, the input to the Offset port must be a vector of real, scalar values that represent the number of pixels by which to translate your image.

Fixed-Point Data Types

The following diagram shows the data types used in the Translate block for bilinear interpolation of fixed-point signals.
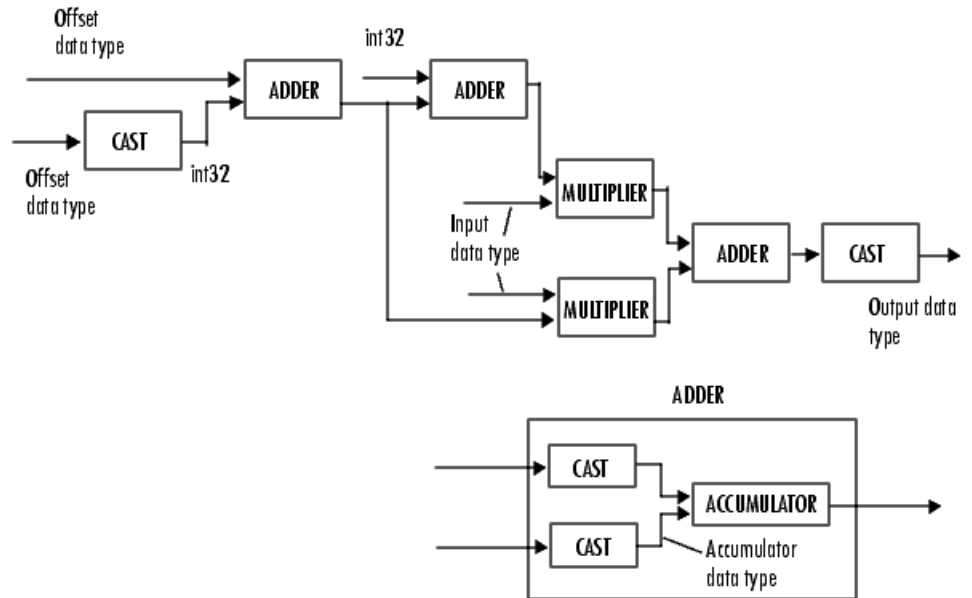
You can set the product output, accumulator, and output data types in the block mask as discussed in the next section.

Translate

Dialog Box

The **Main** pane of the Translate dialog box appears as shown in the following figure.

Function Block Parameters: Translate

Translate

Move an image up or down and/or left or right. You can specify your two-element offset ve
dialog box or the Offset port. The first element represents how many pixels up or down to
If you enter a positive value, the block moves the image downward. The second element
many pixels left or right to shift your image. If you enter a positive value, the block moves
right.

| Main | Data Types |

Parameters

Output size after translation: Full

Offset source: Specify via dialog

Offset: [1.5 2.3]

Background fill value: 0

Interpolation method: Bilinear

OK   Cancel   He

Output size after translation

If you select Full, the block outputs a matrix that contains the translated image values. If you select Same as input image, the block outputs a matrix that is the same size as the input image and contains a portion of the translated image.

Translation values source

Specify how to enter your translation parameters. If you select Specify via dialog, the **Offset** parameter appears in the dialog box. If you select Input port, port O appears on the block. The block uses the input to this port at each time step as your translation values.

Offset source

Enter a vector of real, scalar values that represent the number of pixels by which to translate your image.

Background fill value

Specify a value for the pixels that are outside the image.

Interpolation method

Specify which interpolation method the block uses to translate the image. If you select Nearest neighbor, the block uses the value of one nearby pixel for the new pixel value. If you select Bilinear, the new pixel value is the weighted average of the four nearest pixel values. If you select Bicubic, the new pixel value is the weighted average of the sixteen nearest pixel values.

Maximum offset

Enter a vector of real, scalar values that represent the maximum number of pixels by which you want to translate your image. This parameter must have the same data type as the input to the Offset port. This parameter is visible if, for the **Output size after translation** parameter, you select Full and, for the **Translation values source** parameter, you select Input port.

The **Data Types** pane of the Translate dialog box appears as shown in the following figure.

Translate

Function Block Parameters: Translate

Translate

Move an image up or down and/or left or right. You can specify your two-element offset ve
dialog box or the Offset port. The first element represents how many pixels up or down to
If you enter a positive value, the block moves the image downward. The second element
many pixels left or right to shift your image. If you enter a positive value, the block moves
right.

Main	Data Types

Floating-point inheritance takes precedence over the settings in the 'Data Type' column bel
block input is floating point, all block data types match the input.

Fixed-point operational parameters

Rounding mode: Nearest ▼   Overflow mode: Saturate

Fixed-point data types

	Data Type	Signed	Word length	Fraction
Offset values	Same word length as input ▼			
Product output	Binary point scaling ▼	Yes	32	10
Accumulator	Same as product output ▼			
Output	Same as first input ▼			

☐ Lock data type settings against changes by the fixed-point tools

OK   Cancel   He

Rounding mode

> Select the rounding mode for fixed-point operations.

Overflow mode

> Select the overflow mode for fixed-point operations.

Offset values

> Choose how to specify the word length and the fraction length of the offset values.
>
> - When you select `Same word length as input`, the word length of the offset values match that of the input to the block. In this mode, the fraction length of the offset values is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the offset values.
>
> - When you select `Specify word length`, you can enter the word length of the offset values, in bits. The block automatically sets the fraction length to give you the best precision.
>
> - When you select `Binary point scaling`, you can enter the word length and the fraction length of the offset values, in bits.
>
> - When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the offset values. The bias of all signals in the Computer Vision System Toolbox blocks is 0.
>
> This parameter is visible if, for the **Translation values source** parameter, you select `Specify via dialog`.

Product output

As depicted in the previous figure, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how to designate this product output word and fraction lengths.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Accumulator

As depicted in the previous figure, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how to designate this accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Output

Choose how to specify the word length and fraction length of the output of the block:

- When you select `Same as first input`, these characteristics match those of the first input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. The bias of all signals in the Computer Vision System Toolbox blocks is 0.

Lock data type settings against change by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool in the Simulink documentation.

References   [1] Wolberg, George. *Digital Image Warping*. Washington: IEEE Computer Society Press, 1990.

See Also

Resize	Computer Vision System Toolbox software
Rotate	Computer Vision System Toolbox software
Shear	Computer Vision System Toolbox software

Video From Workspace

Purpose   Import video signal from MATLAB workspace

Library   Sources

visionsources

Description

Video From Workspace
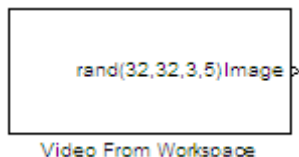
The Video From Workspace block imports a video signal from the MATLAB workspace. If the video signal is a M-by-N-by-T workspace array, the block outputs an intensity video signal, where M and N are the number of rows and columns in a single video frame, and T is the number of frames in the video signal. If the video signal is a M-by-N-by-C-by-T workspace array, the block outputs a color video signal, where M and N are the number of rows and columns in a single video frame, C is the number of color channels, and T is the number of frames in the video stream. In addition to the video signals previously described, this block supports fi objects and variables that are in the structure format returned by the MATLAB `VideoReader` function.

Note If you generate code from a model that contains this block, Simulink Coder takes a long time to compile the code because it puts all of the video data into the `.c` file. Before you generate code, you should convert your video data to a format supported by the From Multimedia File block or the Read Binary File block.

Port	Output	Supported Data Types	Complex Values Supported
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, 32-bit signed integer • 8-, 16-, 32-bit unsigned integer	No
R, G, B	Scalar, vector, or matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports have the same dimensions.	Same as I port	No

For the Computer Vision System Toolbox blocks to display video data properly, double- and single-precision floating-point pixel values must be from 0 to 1. This block does not scale pixel values.

Use the **Signal** parameter to specify the MATLAB workspace variable from which to read. For example, to read an AVI file, use the following syntax:

```
mov = VideoReader('filename.avi')
```

The VideoReader function reads the AVI file into the MATLAB movie structure mov. The MATLAB movie structure might be obsoleted in the future.

If filename.avi has a colormap associated with it, the AVI file must satisfy the following conditions or the block produces an error:

• The colormap must be empty or have 256 values.

Video From Workspace

- The data must represent an intensity image.

- The data type of the image values must be `uint8`.

Use the **Sample time** parameter to set the sample period of the output signal.

When the block has output all of the available signal samples, it can start again at the beginning of the signal, repeat the final value, or generate 0s until the end of the simulation. The **Form output after final value by** parameter controls this behavior:

- When you specify `Setting To Zero`, the block generates zero-valued outputs for the duration of the simulation after generating the last frame of the signal.

- When you specify `Holding Final Value`, the block repeats the final frame for the duration of the simulation after generating the last frame of the signal.

- When you specify `Cyclic Repetition`, the block repeats the signal from the beginning after it reaches the last frame in the signal.

Use the **Image signal** parameter to specify how the block outputs a color video signal. If you select `One multidimensional signal`, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Use the **Output port labels** parameter to label your output ports. Use the spacer character, |, as the delimiter. This parameter is available when the **Image signal** parameter is set to `Separate color signals`.

Dialog Box

The Video From Workspace dialog box appears as shown in the following figure.

Source Block Parameters: Video From Workspace

Video From Workspace (mask) (link)

Outputs video frames from the MATLAB workspace at successive sample times.

If the video signal is a M-by-N-by-T workspace array, the block outputs an intensity video signal, where M and N are the number of rows and columns in a single video frame, and T is the number of frames in the video signal.

If the video signal is a M-by-N-by-C-by-T workspace array, the block outputs a color video signal, where M and N are the number of rows and columns in a single video frame, C is the number of outputs from the block, and T is the number of frames in the video stream.

Parameters

Signal:

rand(32,32,3,5)

Sample time:

1

Form output after final value by: Cyclic repetition

Image signal: One multidimensional signal

OK | Cancel | Help

Video From Workspace

Signal

Specify the MATLAB workspace variable that contains the video signal, or use the `VideoReader` function to specify an AVI filename.

Sample time

Enter the sample period of the output.

Form output after final value by

Specify the output of the block after all of the specified signal samples have been generated. The block can output zeros for the duration of the simulation (`Setting to zero`), repeat the final value (`Holding Final Value`) or repeat the entire signal from the beginning (`Cyclic Repetition`).

Image signal

Specify how the block outputs a color video signal. If you select `One multidimensional signal`, the block outputs an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port outputs one M-by-N plane of an RGB video stream.

Output port labels

Enter the labels for your output ports using the spacer character, |, as the delimiter. This parameter is available when the **Image signal** parameter is set to `Separate color signals`.

See Also

From Multimedia File	Computer Vision System Toolbox software
Image From Workspace	Computer Vision System Toolbox software
Read Binary File	Computer Vision System Toolbox software
To Video Display	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software

Purpose   Export video signal to MATLAB workspace

Library   Sinks

visionsinks

Description   The Video To Workspace block exports a video signal to the MATLAB
workspace. If the video signal is represented by intensity values, it
appears in the workspace as a three-dimensional *M*-by-*N*-by-*T* array,
where *M* and *N* are the number of rows and columns in a single video
frame, and *T* is the number of frames in the video signal. If it is a
color video signal, it appears in the workspace as a four-dimensional
M-by-*N*-by-*C*-by-*T* array, where M and N are the number of rows and
columns in a single video frame, *C* is the number of inputs to the
block, and *T* is the number of frames in the video stream. During code
generation, Simulink Coder does not generate code for this block.

Note This block supports intensity and color images on its ports.

Port	Input	Supported Data Types	Complex Values Supported
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, 32-bit signed integer	No

Video To Workspace

Port	Input	Supported Data Types	Complex Values Supported
		• 8-, 16-, 32-bit unsigned integer	
R, G, B	Scalar, vector, or matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports have the same dimensions.	Same as I port	No

Use the **Variable name** parameter to specify the MATLAB workspace variable to which to write the video signal.

Use the **Number of inputs** parameter to determine the number of inputs to the block. If the video signal is represented by intensity values, enter 1. If it is a color (R, G, B) video signal, enter 3.

Use the **Limit data points to last** parameter to determine the number of video frames, T, you want to export to the MATLAB workspace.

If you want to downsample your video signal, use the **Decimation** parameter to enter your decimation factor.

If your video signal is fixed point and you select the **Log fixed-point data as a fi object** check box, the block creates a fi object in the MATLAB workspace.

Use the **Input port labels** parameter to label your input ports. Use the spacer character, |, as the delimiter. This parameter is available if the **Number of inputs** parameter is greater than 1.

Dialog Box

The Video To Workspace dialog box appears as shown in the following figure.
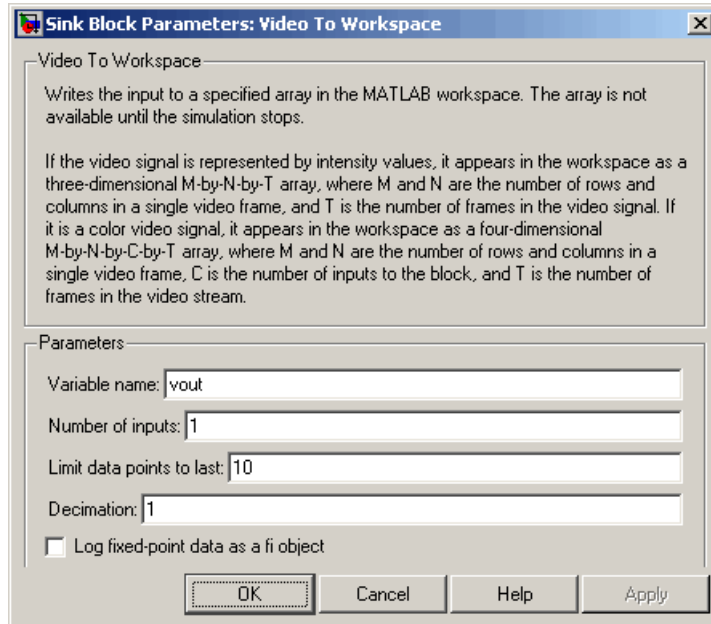
Variable name

Specify the MATLAB workspace variable to which to write the video signal.

Number of inputs

Enter the number of inputs to the block. If the video signal is black and white, enter 1. If it is a color (R, G, B) video signal, enter 3.

Limit data points to last

Enter the number of video frames to export to the MATLAB workspace.

Decimation

Enter your decimation factor.

Log fixed-point data as a fi object

    If your video signal is fixed point and you select this check box, the block creates a fi object in the MATLAB workspace. For more information of fi objects, see the Fixed-Point Designer documentation.

Input port labels

    Enter the labels for your input ports using the spacer character, |, as the delimiter. This parameter is available if the **Number of inputs** parameter is greater than 1.

See Also

To Multimedia File	Computer Vision System Toolbox software
To Video Display	Computer Vision System Toolbox software
Video Viewer	Computer Vision System Toolbox software

Purpose   Display binary, intensity, or RGB images or video streams

Library   Sinks

visionsinks

Description   The Video Viewer block enables you to view a binary, intensity, or RGB image or a video stream. The block provides simulation controls for play, pause, and step while running the model. The block also provides pixel region analysis tools. During code generation, Simulink Coder software does not generate code for this block.

Note The To Video Display block supports code generation.

See the following table for descriptions of both input types.

Input	Description
Image	M-by-N matrix of intensity values or an M-by-N-by-P color video signal where P is the number of color planes.
R/G/B	Scalar, vector, or matrix that represents one plane of the RGB video stream. Inputs to the R, G, or B ports must have the same dimensions and data type.

Select **File > Image Signal** to set the input to either **Image** or **RGB**.

Video Viewer

Toolbar

Playback Toolbar

Video Frames

Video Status   Type of Video   Frame Rate   Time
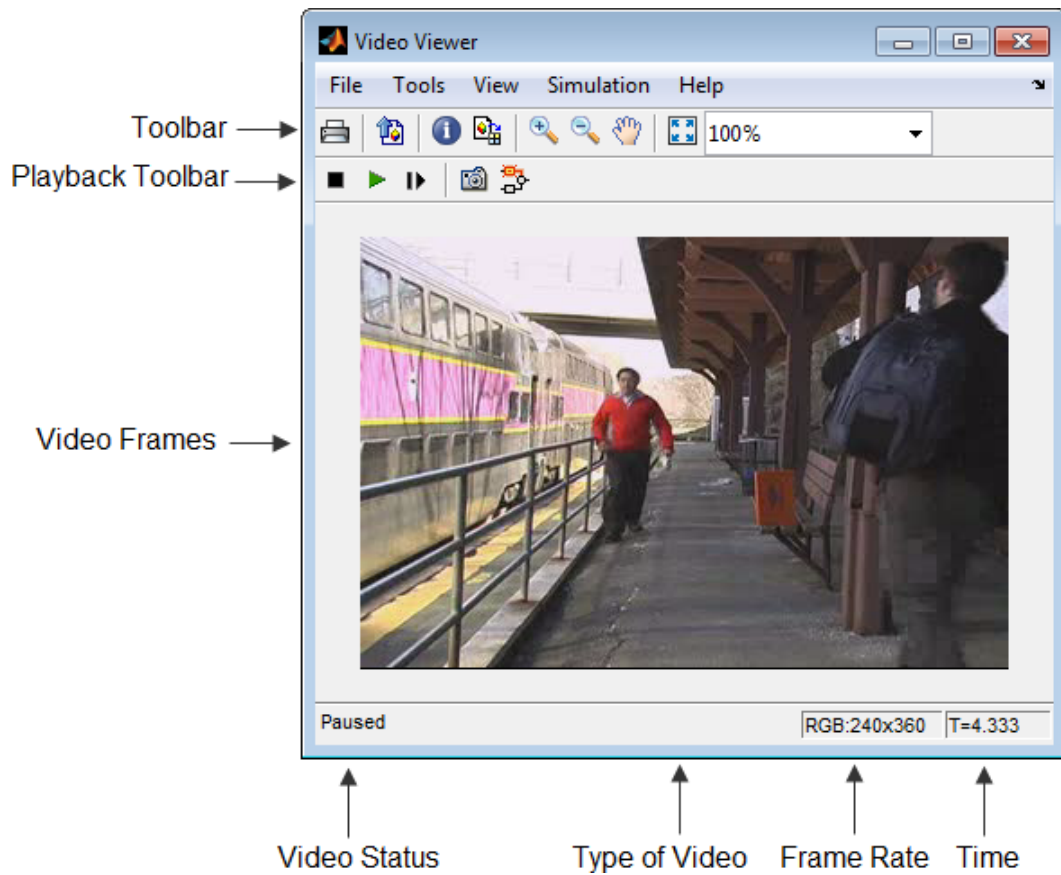
Toolbar

Toolbar

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
🖨	**File > Print**	**Ctrl+P**	Print the current display window. Printing is only available when the display is not changing. You can enable printing by placing the display in snapshot mode, or by pausing or stopping model simulation, or simulating the model in step-forward mode. To print the current window to a figure rather than sending it to your printer, select **File > Print to figure**.
🔼	**File > Export to Image Tool**	**Ctrl+E**	Send the current video frame to the Image Tool. For more information, see "Using the Image Tool to Explore Images" in the Image Processing Toolbox documentation.

Note The Image Tool can only know that the frame is an intensity image if the colormap of the frame is grayscale (gray(256)). Otherwise, the Image Tool assumes the frame is an indexed image and disables the **Adjust Contrast** button.

Toolbar (Continued)

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
(i)	**Tools > Video Information**	V	View information about the video data source.
	Tools > Pixel Region	N/A	Open the Pixel Region tool. For more information about this tool, see the Image Processing Toolbox documentation.
	Tools > Zoom In	N/A	Zoom in on the video display.
	Tools > Zoom Out	N/A	Zoom out of the video display.
	Tools > Pan	N/A	Move the image displayed in the GUI.
	Tools > Maintain Fit to Window	N/A	Scale video to fit GUI size automatically. Toggle the button on or off.
100%	N/A	N/A	Enlarge or shrink the video frame. This option becomes available if you do not select the **Maintain Fit to Window**.

Playback Toolbar

Playback Toolbar

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
■	**Simulation > Stop**	**S**	Stop the video.
▶	**Simulation > Play**	**P**, Space bar	Play the video.
❙❙	**Simulation > Pause**	**P**, Space bar	Pause the video. This button appears only when the video is playing.
❙▶	**Simulation > Step Forward**	Right arrow, **Page Down**	Step forward one frame.
📷	**Simulation > Simulink Snapshot**	N/A	Click this button to freeze the display in the viewer window.
File-menu only	**Simulation > Drop Frames to Improve Performance**	**Ctrl+R**	Enable the viewer to drop video frames to improve performance.
🔸	**View > Highlight Simulink Signal**	**Ctrl+L**	In the model window, highlight the Simulink signal the viewer is displaying.

Video Viewer

Setting Viewer Configuration

The Video Viewer Configuration preferences enables you to change the behavior and appearance of the graphic user interface (GUI) as well as the behavior of the playback shortcut keys.

- To open the Configuration dialog box, select **File > Configuration Set > Edit**.

- To save the configuration settings for future use, select **File > Configuration Set > Save as**.

Core Pane

The Core pane in the Viewer Configuration dialog box controls the GUI's general settings.
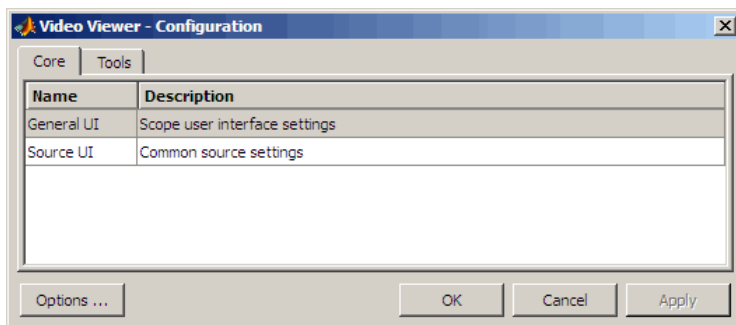
General UI
Click **General UI**, and click the **Options** button to open the General UI Options dialog box.
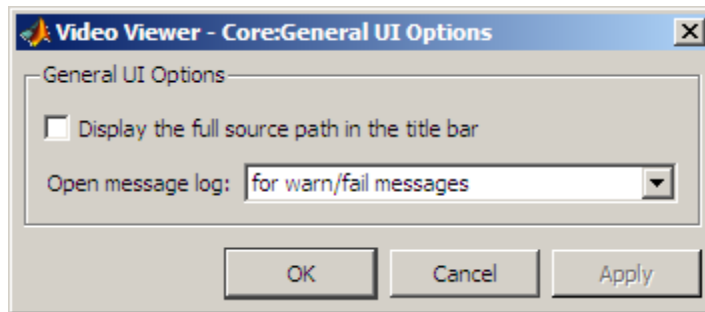
If you select the **Display the full source path in the title bar** check box, the GUI displays the model name and full Simulink path to the video data source in the title bar. Otherwise, it displays a shortened name.

Use the **Open message log:** parameter to control when the Message log window opens. You can use this window to debug issues with video playback. Your choices are `for any new messages`, `for warn/fail messages`, `only for fail messages`, or `manually`.

Tools Pane

The Tools pane in the Viewer Configuration dialog box contains the tools that appear on the Video Viewer GUI. Select the **Enabled** check box next to the tool name to specify which tools to include on the GUI.
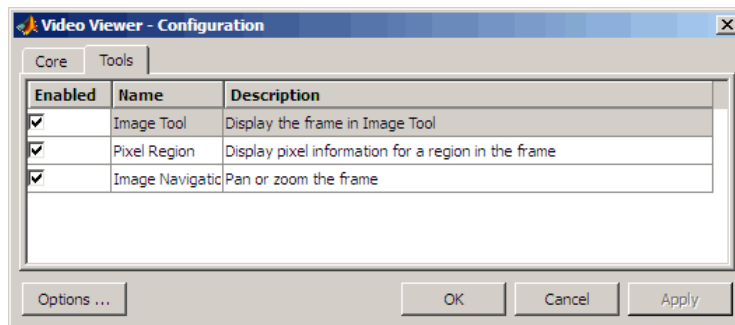
Image Tool

Click **Image Tool**, and then click the **Options** button to open the Image Tool Options dialog box.
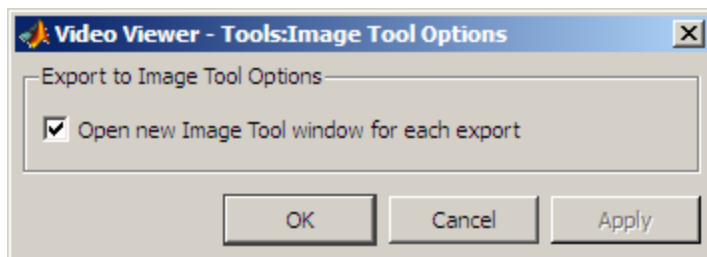
Select the **Open new Image Tool window for export** check box if you want to open a new Image Tool for each exported frame.

Pixel Region
Select the **Pixel Region** check box to display and enable the pixel region GUI button. For more information on working with pixel regions see Getting Information about the Pixels in an Image.

Image Navigation Tools
Select the **Image Navigation Tools** check box to enable the pan-and-zoom GUI button.

Instrumentation Set
Select the **Instrumentation Set** check box to enable the option to load and save viewer settings. The option appears in the **File** menu.

Video Information

The Video Information dialog box lets you view basic information about the video. To open this dialog box, you can select **Tools > Video Information** , click the information button , or press the **V** key.
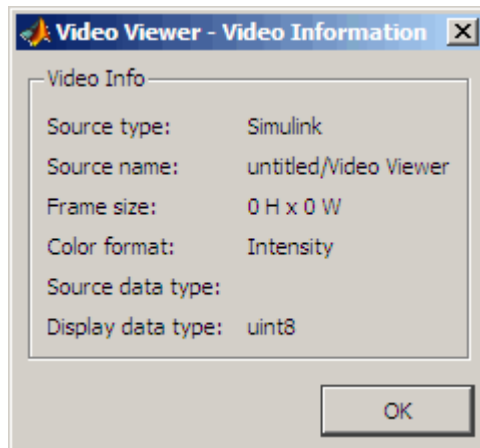
Colormap for Intensity Video

The Colormap dialog box lets you change the colormap of an intensity video. You cannot access the parameters on this dialog box when the GUI displays an RGB video signal. To open this dialog box for an intensity signal, select **Tools > Colormap** or press **C**.
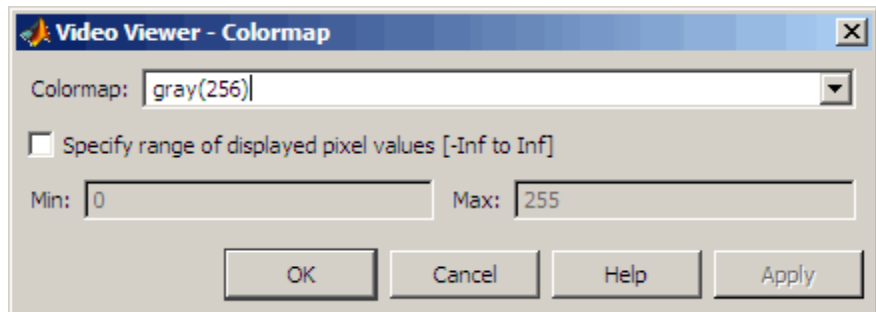
Use the **Colormap** parameter to specify the colormap to apply to the intensity video.

If you know that the pixel values do not use the entire data type range, you can select the **Specify range of displayed pixel values** check

box and enter the range for your data. The dialog box automatically displays the range based on the data type of the pixel values.

Status Bar

A status bar appear along the bottom of the Video Viewer. It displays information pertaining to the video status (running, paused or ready), type of video (Intensity or RGB) and video time.

Message Log

The Message Log dialog provides a system level record of configurations and extensions used. You can filter what messages to display by **Type** and **Category**, view the records, and display record details.

The **Type** parameter allows you to select either `All`, `Info`, `Warn,` or `Fail` message logs. The **Category** parameter allows you to select either `Configuration` or `Extension` message summaries. The `Configuration` messages indicate when a new configuration file is loaded. The `Extension` messages indicate a component is registered. For example, you might see a `Simulink` message, which indicates the component is registered and available for configuration.

Saving the Settings of Multiple Video Viewer GUIs

The Video Viewer GUI enables you to save and load the settings of multiple GUI instances. Thus, you only need to configure the Video Viewer GUIs that are associated with your model once. To save the GUI settings, select **File > Instrumentation Sets > Save Set**. To open the preconfigured GUIs, select **File > Instrumentation Sets > Load Set**.

Supported Data Types

Port	Supported Data Types
Image	• Double-precision floating point
	• Single-precision floating point
	• Boolean
	• 8-, 16-, and 32-bit signed integer
	• 8-, 16-, and 32-bit unsigned integer
R/G/B	Same as Image port

See Also

From Multimedia File	Computer Vision System Toolbox software
mplay	Computer Vision System Toolbox software
To Multimedia File	Computer Vision System Toolbox software
To Video Display	Computer Vision System Toolbox software
Video To Workspace	Computer Vision System Toolbox software
implay	Image Processing Toolbox

Write AVI File (To Be Removed)

Purpose   Write video frames to uncompressed AVI file

Library   Sinks

Description
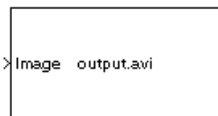
> | Image   output.avi
>
> Write AVI File

Note The Write AVI File block is obsolete. It may be removed in a future version of the Computer Vision System Toolbox blocks. Use the replacement block To Multimedia File.

See the Write AVI File documentation to view the reference material related to the obsolete block.

Purpose   Write binary video data to files

Library   Sinks

                  visionsinks

Description   The Write Binary File block takes video data from a Simulink model
                  and exports it to a binary file.

                  This block produces a raw binary file with no header information. It
                  has no encoded information providing the data type, frame rate or
                  dimensionality. The video data for this block appears in row major
                  format.

                  Note This block supports code generation only for platforms that have
                  file I/O available. You cannot use this block to do code generation with
                  Real-Time Windows Target (RTWin).

Port	Input	Supported Data Types	Complex Values Supported
Input	Matrix that represents the luma (Y') and chroma (Cb and Cr) components of a video stream	• 8-, 16- 32-bit signed integer • 8-, 16- 32-bit unsigned integer	No

Four Character Code Video Formats

Four Character Codes (FOURCC) identify video formats. For more
information about these codes, see http://www.fourcc.org.

Use the **Four character code** parameter to identify the video format.

Custom Video Formats

You can use the Write Binary File block to create a binary file that
contains video data in a custom format.

Write Binary File

- Use the **Bit stream format** parameter to specify whether you want your data in planar or packed format.

- Use the **Number of input components** parameter to specify the number of components in the video stream. This number corresponds to the number of block input ports.

- Select the **Inherit size of components from input data type** check box if you want each component to have the same number of bits as the input data type. If you clear this check box, you must specify the number of bits for each component.

- Use the **Component** parameters to specify the component names.

- Use the **Component order in binary file** parameter to specify how to arrange the components in the binary file.

- Select the **Interlaced video** check box if the video stream represents interlaced video data.

- Select the **Write signed data to output file** check box if your input data is signed.

- Use the **Byte order in binary file** parameter to specify whether the byte ordering in the output binary file is little endian or big endian.

Dialog Box

The Write Binary File dialog box appears as shown in the following figure.
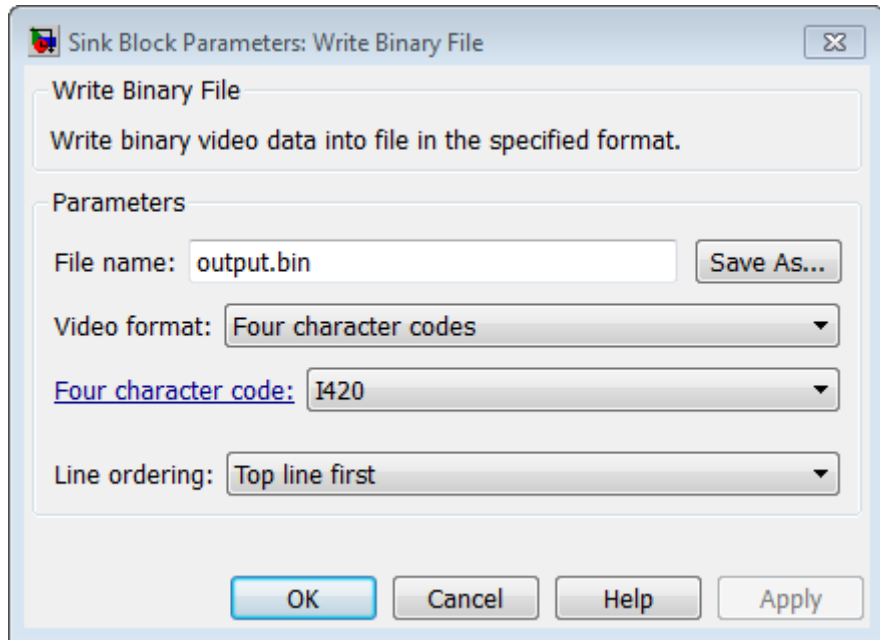
File name
> Specify the name of the binary file. To specify a different file or location, click the **Save As...** button.

Video format
> Specify the format of the binary video data as either Four character codes or Custom. See "Four Character Code Video Formats" on page 1-643 or "Custom Video Formats" on page 1-643 for more details.

Four character code
> From the list, select the binary file format.

Write Binary File

Line ordering
> Specify how the block fills the binary file. If you select `Top line first`, the block first fills the binary file with the first row of the video frame. It then fills the file with the other rows in increasing order. If you select `Bottom line first`, the block first fills the binary file with the last row of the video frame. It then fills the file with the other rows in decreasing order.

Bit stream format
> Specify whether you want your data in planar or packed format.

Number of input components
> Specify the number of components in the video stream. This number corresponds to the number of block input ports.

Inherit size of components from input data type
> Select this check box if you want each component to have the same number of bits as the input data type. If you clear this check box, you must specify the number of bits for each component.

Component
> Specify the component names.

Component order in binary file
> Specify how to arrange the components in the binary file.

Interlaced video
> Select this check box if the video stream represents interlaced video data.

Write signed data to output file
> Select this check box if your input data is signed.

Byte order in binary file
> Use this parameter to specify whether the byte ordering in the output binary file is little endian or big endian.

See Also

| Read Binary File | Computer Vision System Toolbox |
| To Multimedia File | Computer Vision System Toolbox |

Alphabetical List

binaryFeatures

Purpose	Object for storing binary feature vectors

Description   This object provides the ability to pass data between the
                    `extractFeatures` and `matchFeatures` functions. It can also be used to
                    manipulate and plot the data returned by `extractFeatures`.

Construction   `features= binaryFeatures(featureVectors)` constructs
                    a `binaryFeatures` object from the *M*-by-*N* input matrix,
                    `featureVectors`. This matrix contains *M* feature vectors stored in
                    N uint8 containers.

                    `features = binaryFeatures(featureVectors,Name,Value)` uses
                    additional options specified by one or more `Name,Value` pair
                    arguments.

Input Arguments

featureVectors

Input feature vectors, specified as an *M*-by-*N* input matrix. This
matrix contains *M* binary feature vectors stored in *N* uint8
containers.

Read-only Properties

Features

Feature vectors, stated as an *M*-by-*N* input matrix. This matrix
consists of *M* feature vectors stored in *N* uint8 containers.

NumBits

Number of bits per feature, which is the number of uint8 feature
vector containers times 8.

NumFeatures

Number of feature vectors contained in the `binaryFeatures`
object.

Examples   **Match Two Sets of Binary Feature Vectors**

Input feature vectors.

```
features1 = binaryFeatures(uint8([1 8 7 2; 8 1 7 2]));
features2 = binaryFeatures(uint8([8 1 7 2; 1 8 7 2]));
```

Match the vectors using the Hamming distance.

```
[indexPairs matchMetric] = matchFeatures(features1, features2)
```

See Also   extractFeatures | matchFeatures

MSERRegions

Purpose   Object for storing MSER regions

Description   This object describes MSER regions and corresponding ellipses that
                   have the same second moments as the regions. It passes data between
                   the `detectMSERFeatures` and `extractFeatures` functions. The object
                   can also be used to manipulate and plot the data returned by these
                   functions.

Tips   Although `MSERRegions` may hold many regions, it is a scalar object.
                   Therefore, `NUMEL(MSERRegions)` always returns 1. This value may differ
                   from `LENGTH(MSERRegions)`, which returns the true number of regions
                   held by the object.

Construction   `regions = MSERRegions(pixellist)` constructs an MSER regions
                   object, `regions`, from an *M*-by-1 cell array of regions,`pixellist`. Each
                   cell contains a *P*-by-2 array of [x y] coordinates for the detected MSER
                   regions, where *P* varies based on the number of pixels in a region.

Input Arguments

pixellist

   M-by-2 cell array of [x y] coordinates of the detected MSER
   regions.

Properties   The following properties are read-only and are calculated once the input
                   pixel list is specified.

Count

   Number of stored regions

   Default: 0

Location

   An *M*-by-2 array of [x y] centroid coordinates of ellipses that have
   the same second moments as the MSER regions.

Axes

> A two-element vector, [majorAxis minorAxis]. This vector specifies the major and minor axis of the ellipse that have the same second moments as the MSER regions.

Orientation

> A value in the range from -pi/2 to +pi/2 radians. This value represents the orientation of the ellipse as measured from the *X*-axis to the major axis of the ellipse. You can use this property for visualization purposes.

Methods

isempty	Returns true for empty object
length	Number of stored points
plot	Plot MSER regions
size	Size of the MSERRegions object

Examples

Detect MSER Features in an Image

Detect MSER features. Display centroids and axes of detected regions.

Load an image.

```
I = imread('cameraman.tif');
```

Store regions.

```
regions = detectMSERFeatures(I);
```

Display the centroids and axes of detected regions.

```
imshow(I); hold on;
plot(regions);
```

Display MSER Feature Regions from the MSERRegions Object

Detect and display the first 10 regions contained in the MSERRegions object

Detect MSER features.

```
I = imread('cameraman.tif');
regions = detectMSERFeatures(I);
```

Display the first 10 regions in the MSERRegions object.

```
imshow(I); hold on;
plot(regions(1:10),'showPixelList', true);
```

Combine MSER Region Detector with SURF Descriptors

Extract and display SURF descriptors at locations identified by MSER detector.

Read image.

```
I = imread('cameraman.tif');
```

Detect MSER features.

```
regionsObj = detectMSERFeatures(I);
```

Extract and display SURF descriptors.

```
[features, validPtsObj] = extractFeatures(I, regionsObj);
imshow(I); hold on;
plot(validPtsObj,'showOrientation',true);
```

References   [1] Nister, D., and H. Stewenius, "Linear Time Maximally Stable Extremal Regions", *Lecture Notes in Computer Science*. 10th European Conference on Computer Vision, Marseille, France: 2008, no. 5303, pp. 183–196.

[2] Matas, J., O. Chum, M. Urba, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions. "*Proceedings of British Machine Vision Conference*, pages 384-396, 2002.

See Also   detectMSERFeatures | detectSURFFeaturesSURFPoints **|**
vision.EdgeDetector **|** | extractFeatures | matchFeatures

**Related
Examples**
- "Detect MSER Interest Points" on page 3-35
- "Detect SURF Interest Points in a Grayscale Image" on page 3-40

MSERRegions.isempty

Purpose   Returns true for empty object

Syntax   isempty(MSERRegionsObj)

Description   isempty(MSERRegionsObj) returns a true value, if the MSERRegionsObj object is empty.

Purpose   Number of stored points

Syntax   `length(MSERRegionsObj)`

Description   `length(MSERRegionsObj)` returns the number of stored points in the `MSERRegionsObj` object.

MSERRegions.plot

Purpose   Plot MSER regions

Syntax
```
MSERRegions.plot
MSERRegions.plot(AXES_HANDLE,...)
MSERRegions.plot(AXES_HANDLE,Name,Value)
```

Description   `MSERRegions.plot` plots MSER points in the current axis.

`MSERRegions.plot(AXES_HANDLE,...)` plots using axes with the handle `AXES_HANDLE`.

`MSERRegions.plot(AXES_HANDLE,Name,Value)` Additional control for the `plot` method requires specification of parameters and corresponding values. An additional option is specified by one or more `Name,Value` pair arguments.

Input Arguments

AXES_HANDLE

Handle for plot method to use for display. You can set the handle using `gca`.

Name-Value Pair Arguments

showEllipses

Display ellipses around feature. Set this value to `true` or `false`. When you set this value to `true`, the object draws an ellipse with the same 2nd order moments as the region.

When you set this value to `false`, only the ellipses centers are plotted.

Default: `true`

showOrientation

Display a line corresponding to the region's orientation. Set this value to `true` or `false`. When you set this value to `true`, the object draws a line corresponding to the region's orientation. The

object draws the line from the ellipse's centroid to the edge of the ellipse, which indicates the region's major axis.

Default: false

showPixelList

Display the MSER regions. Set this value to true or false. When you set this value to true, the object plots the MSER regions using the JET colormap.

Examples

Plot MSER Regions

Extract MSER features and plot the regions

1

Read image and extract MSER features

```
I = imread('cameraman.tif');
regions = detectMSERFeatures(I);
imshow(I);hold on;
plot(regions);
```

2

Plot MSER Regions

```
figure; imshow(I);hold on;
plot(regions,'showPixelList',true, 'showEllipses',false);
hold off;
```

MSERRegions.size

Purpose   Size of the MSERRegions object

Syntax   `size(MSERRegionsObj)`

Description   `size(MSERRegionsObj)` returns the size of the `MSERRegionsObj` object.

Purpose　　Object for storing corner points

Description　　This object stores information about feature points detected from a 2-D grayscale image.

Tips　　Although `cornerPoints` may hold many points, it is a scalar object. Therefore, `NUMEL(cornerPoints)` always returns `1`. This value may differ from `LENGTH(cornerPoints)`, which returns the true number of points held by the object.

Construction　　`points = cornerPoints(Location)` constructs a `cornerPoints` object from an *M*-by-2 array of [x y] point coordinates, `Location`.

`points = cornerPoints(Location,Name,Value)` constructs a `cornerPoints` object with optional input properties specified by one or more `Name,Value` pair arguments. Each additional property can be specified as a scalar or a vector whose length matches the number of coordinates in `Location`.

Input Arguments

Location

M-by-2 array of [x y] point coordinates.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Count'

Number of points held by the object.

Default: `0`

cornerPoints

'Metric'

> Value describing strength of detected feature.

> **Default:** `0.0`

Methods

isempty	Returns true for empty object
plot	Plot corner points
selectStrongest	Return points with strongest metrics
size	Size of the cornerPoints object

Examples

Plot the Strongest Features from Detected Feature Points

Read image.

```
I = imread('cameraman.tif');
```

Detect feature points.

```
points = detectHarrisFeatures(I);
```

Display the 10 strongest points.

```
strongest = points.selectStrongest(10);
imshow(I); hold on;
plot(strongest);
```

Display the [*x y*] coordinates.

```
strongest.Location
```

Create a `cornerPoints` Object and Display Points

1

Create a checkerboard image.

```
I = checkerboard(50,2,2);
```

2

Load locations.

```
location = [51    51    51   100   100   100   151   151   151;...
            50   100   150    50   101   150    50   100   150]';
```

3

Save points.

```
points = cornerPoints(location);
```

4

Display points on image.

```
imshow(I); hold on;
plot(points);
```

See Also   detectHarrisFeatures | detectFASTFeatures |
                 detectMinEigenFeatures | extractFeatures |
                 matchFeaturesMSERRegions |

cornerPoints.isempty

Purpose   Returns true for empty object

Syntax   isempty(cornerPointsObj)

Description isempty(cornerPointsObj) returns a true value, if the corner points object is empty.

Purpose   Number of stored points

Syntax   length(cornerPointsObj)

Description length(cornerPointsObj) returns the number of stored points in the
                corner points object.

cornerPoints.plot

Purpose	Plot corner points
Syntax	`cornerPoints.plot`
Description	`cornerPoints.plot` plots corner points in the current axis.

Input Arguments

AXES_HANDLE

Handle for plot method to use for display. You can set the handle using `gca`.

Examples

Read an image.

```
I = imread('cameraman.tif');
```

Detect corner features.

```
featurePoints = detectHarrisFeatures(I);
```

Plot feature image with detected features.

```
imshow(I); hold on;
plot(featurePoints);
```

Purpose   Return points with strongest metrics

Syntax   cornerPoints = cornerPoints.selectStrongest(N)

Description   cornerPoints = cornerPoints.selectStrongest(N) returns N
                   number of points with strongest metrics.

Examples   Create object holding 50 points.

                   points = cornerPoints(ones(50,2), 'Metric', 1:50);

                   Keep 2 strongest features.

                   points = points.selectStrongest(2)

cornerPoints.size

Purpose   Size of the cornerPoints object

Syntax   `size(cornerPointsObj)`

Description   `size(cornerPointsObj)` returns the size of the corner points object.

Purpose   Object for storing SURF interest points

Description   This object provides the ability to pass data between the
                     `detectSURFFeatures` and `extractFeatures` functions. It can also be
                     used to manipulate and plot the data returned by these functions.
                     You can use the object to fill the points interactively. You can use
                     this approach in situations where you might want to mix a non-SURF
                     interest point detector with a SURF descriptor.

Tips   Although `SURFPoints` may hold many points, it is a scalar object.
                     Therefore, `NUMEL(surfPoints)` always returns 1. This value may differ
                     from `LENGTH(surfPoints)`, which returns the true number of points
                     held by the object.

Construction   `points = SURFPoints(Location)` constructs a `SURFPoints` object from
                     an *M*-by-2 array of [x y] point coordinates, `Location`.

                     `points = SURFPoints(Location,Name,Value)` constructs a
                     `SURFPoints` object with optional input properties specified by one or
                     more `Name,Value` pair arguments. Each additional property can be
                     specified as a scalar or a vector whose length matches the number of
                     coordinates in `Location`.

Input Arguments

Location

   M-by-2 array of [x y] point coordinates.

Properties   #### Count

   Number of points held by the object.

   Default: 0

Scale

   Specifies scale at which the interest points were detected. This
   value must be greater than or equal to `1.6`.

Default: `1.6`

Metric

Value describing strength of detected feature. The SURF algorithm uses a determinant of an approximated Hessian.

Default: `0.0`

SignOfLaplacian

Sign of the Laplacian determined during the detection process. This value must be an integer, `-1`, `0`, or `1`. You can use this parameter to accelerate the feature matching process.

Blobs with identical metric values but different signs of Laplacian can differ by their intensity values. For example, a white blob on a blackground versus a black blob on a white background. You can use this parameter to quickly eliminate blobs that do not match.

For non-SURF detectors, this property is not relevant. For example, for corner features, you can simply use the default value of `0`.

Default: `0`

Orientation

Describes the orientation of the detected feature. This value must be specified as an angle, in radians. The angle is measured from the *X*-axis with the origin at the point given by the `Location` property. Typically, this value gets set during the descriptor extraction process. The `extractFeatures` function modifies the default value of `0`. Do not set this property manually. Rely instead on the call to `extractFeatures` to fill in this value. The `Orientation` is mainly useful for visualization purposes.

Default: `0.0`

Methods

isempty	Returns true for empty object
length	Number of stored points
plot	Plot SURF points
selectStrongest	Return points with strongest metrics
size	Size of the SURFPoints object

Examples

Detect SURF Features in an Image

Detect, store, and display the last 5 SURF points of an image

```
I = imread('cameraman.tif');

points = detectSURFFeatures(I);

Display the last 5 points
imshow(I); hold on;
plot(points(end-4:end));
```

Mix FAST Corner Detector with SURF Descriptors

Using corner detection, create a SURF points object and then extract SURF descriptors.

1

Load an image.

```
I = imread('cameraman.tif');
```

2

Construct corner detector System object.

```
hcornerdet = vision.CornerDetector('Method', 'Local intensity comparis
```

3

Invoke the object on image, I.

```
ptsArray = step(hcornerdet, I);
```

4

Transform the array of [x y] coordinates into a SURFPoints object.

```
ptsObj = SURFPoints; % start with an empty object
ptsObj = ptsObj.append(ptsArray, 'Scale', 2);
```

5

Extract SURF descriptors and visualize the first 5 descriptors.

```
[features, validPtsObj] = extractFeatures(I, ptsObj);
imshow(I); hold on;
plot(validPtsObj(1:5),'showOrientation',true);
```

References

[1] Bradski, G. and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*, O'Reilly, Sebastopol, CA, 2008.

[2] Matas, J., O. Chum, M. Urba, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions." Proc. of British Machine Vision Conference, pages 384-396, 2002.

See Also

detectMSERFeatures | extractFeatures | matchFeatures | detectSURFFeaturesMSERRegions |

Related Examples

- "Detect SURF Interest Points in a Grayscale Image" on page 3-40
- "Display MSER Feature Regions from the MSERRegions Object" on page 2-6
- "Detect MSER Interest Points" on page 3-35
- "Detect MSER Features in an Image" on page 2-5

Purpose   Returns true for empty object

Syntax   isempty(SURFPointsObj)

Description isempty(SURFPointsObj) returns a true value, if the SURFPointsObj object is empty.

SURFPoints.length

Purpose   Number of stored points

Syntax   length(SURFPointsObj)

Description   length(SURFPointsObj) returns the number of stored points in the
                   SURFPointsObj object.

Purpose   Plot SURF points

Syntax   surfPoints.plot
                 surfPoints.plot(AXES_HANDLE,...)
                 surfPoints.plot(AXES_HANDLE,Name,Value)

Description surfPoints.plot plots SURF points in the current axis.

surfPoints.plot(AXES_HANDLE,...) plots using axes with the handle AXES_HANDLE.

surfPoints.plot(AXES_HANDLE,Name,Value) Additional control for the plot method requires specification of parameters and corresponding values. An additional option is specified by one or more Name,Value pair arguments.

Input Arguments

AXES_HANDLE

Handle for plot method to use for display. You can set the handle using gca.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

showScale

Display proportional circle around feature. Set this value to true or false. When you set this value to true, the object draws a circle proportional to the scale of the detected feature, with the feature point located at it's center. When you set this value to false, the object turns the display of the circle off.

The algorithm represents the scale of the feature with a circle of 6*Scale radius. The SURF algorithm uses this equivalent size of circular area to compute the orientation of the feature.

SURFPoints.plot

Default: `true`

showOrientation

Display a line corresponding to feature point orientation. Set this value to `true` or `false`. When you set this value to `true`, the object draws a line corresponding to the point's orientation. The object draws the line from the feature point location to the edge of the circle, indicating the scale.

Default: `false`

Examples

```
% Extract SURF features
I = imread('cameraman.tif');
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I, points);

% Visualize 10 strongest SURF features, including their
% scales and orientation which were determined during the
% descriptor extraction process.
imshow(I); hold on;
strongestPoints = valid_points.selectStrongest(10);
strongestPoints.plot('showOrientation',true);
```

Purpose   Return points with strongest metrics

Syntax   `surfPoints = surfPoints.selectStrongest(N)`

Description   `surfPoints = surfPoints.selectStrongest(N)` returns N number of points with strongest metrics.

Examples

```
% Create object holding 50 points
  points = SURFPoints(ones(50,2), 'Metric', 1:50);

% Keep 2 strongest features
  points = points.selectStrongest(2)
```

SURFPoints.size

Purpose   Size of the SURFPoints object

Syntax   size(SURFPointsObj)

Description size(SURFPointsObj) returns the size of the SURFPointsObj object.

Purpose   Combine images, overlay images, or highlight selected pixels

Description   The `AlphaBlender` object combines two images, overlays one image over
                   another, or highlights selected pixels.

                   Use the step syntax below with input images, `I1` and `I2`, the alpha
                   blender object, `H`, and any optional properties.

                   `Y = step(H,I1,I2)` performs the alpha blending operation on images
                   `I1` and `I2`.

                   `Y = step(H,I1,I2,OPACITY)` uses *OPACITY* input to combine pixel
                   values of `I1` and `I2` when you set the `Operation` property to `Blend` and
                   the `OpacitySource` property to `Input port`.

                   `Y = step(H,I1,I2,MASK)` uses *MASK* input to overlay `I2` over `I1` when
                   you set the `Operation` property to `Binary mask` and the `MaskSource`
                   property to `Input port`'.

                   `Y = step(H,I1,MASK)` uses *MASK* input to determine which pixels in `I1`
                   are set to the maximum value supported by their data type when you
                   set the `Operation` property to `Highlight selected pixels` and the
                   `MaskSource` property to `Input port`.

                   `Y = step(H,I1,I2,...,LOCATION)` uses *LOCATION* input to specify
                   the upper-left corner position of `I2` when you set the `LocationSource`
                   property to `Input port`.

Construction   `H = vision.AlphaBlender` returns an alpha blending System object™,
                   `H`, that combines the pixel values of two images, using an opacity factor
                   of `0.75`.

                   `H = vision.AlphaBlender(*Name*,*Value*)` returns an alpha blending
                   object, `H`, with each specified property set to the specified value. You can
                   specify additional name-value pair arguments in any order as (*Name1*,
                   Value1,...,*NameN*,*ValueN*).

Properties   **Operation**

                       Operation to perform

Specify the operation that the object performs as `Blend`, `Binary mask`, or `Highlight selected pixels`. If you set this property to `Blend`, the object linearly combines the pixels of one image with another image. If you set this property to `Binary mask`, the object overwrites the pixel values of one image with the pixel values of another image. If you set this property to `Highlight selected pixel`, the object uses the binary image input, `MASK`, to determine which pixels are set to the maximum value supported by their data type.

OpacitySource

Source of opacity factor

Specify how to determine any opacity factor as `Property` or `Input port`. This property applies when you set the `Operation` property to `Blend`. The default is `Property`.

Opacity

Amount by which the object scales each pixel value before combining them

Specify the amount by which the object scales each pixel value before combining them. Determine this value before combining pixels as a scalar value used for all pixels, or a matrix of values that defines the factor for each pixel. This property applies when you set the `OpacitySource` property to `Property`. This property is tunable. The default is `0.75`.

MaskSource

Source of binary mask

Specify how to determine any masking factors as `Property` or `Input port`. This property applies when you set the `Operation` property to `Binary mask`. The default is `Property`.

Mask

Which pixels are overwritten

Specify which pixels are overwritten as a binary scalar `0` or `1` used for all pixels, or a matrix of `0`s and `1`s that defines the factor for each pixel. This property applies when you set the `MaskSource` property to `Property`. This property is tunable. The default is `1`.

LocationSource

Source of location of the upper-left corner of second input image

Specify how to enter location of the upper-left corner of second input image as `Property` or `Input port`. The default is `Property`.

Location

Location [x y] of upper-left corner of second input image relative to first input image

Specify the row and column position of upper-left corner of the second input image relative to upper-left corner of first input image as a two-element vector. This property applies when you set the `LocationSource` property to `Property`. The default is `[1 1]`. This property is tunable.

See "Coordinate Systems" for a discussion on pixel coordinates and spatial coordinates, which are the two main coordinate systems used in the Computer Vision System Toolbox.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

OpacityDataType

Opacity word and fraction lengths

Specify the opacity factor fixed-point data type as `Same word length as input` or `Custom`. The default is `Same word length as input`.

CustomOpacityDataType

Opacity word and fraction lengths

Specify the opacity factor fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OpacityDataType` property to `Custom`. The default is `numerictype([],16)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Same as first input` or `Custom`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Same as first input`, or `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Same as first input` or `Custom`. The default is `Same as first input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],32,10)`.

Methods

clone	Create alpha blender object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Blend images, overlay images, or highlight selected pixels

vision.AlphaBlender

Examples

Blend two images:

```
I1 = im2single(imread('blobs.png'));
I2 = im2single(imread('circles.png'));
halphablend = vision.AlphaBlender;
J = step(halphablend,I1,I2);
imshow(J)
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Compositing block reference page. The object properties correspond to the block parameters.

See Also

vision.TextInserter | vision.ShapeInserter | vision.MarkerInserter

Purpose   Create alpha blender object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an AlphaBlender System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.AlphaBlender.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                     the step method.

                     The getNumInputs method returns a positive integer that is the number
                     of expected inputs (not counting the object itself) to the step method.
                     This value will change if you alter any properties that turn inputs on or
                     off. You must call the step method with the number of input arguments
                     equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.AlphaBlender.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the AlphaBlender
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note   You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.AlphaBlender.step

Purpose   Blend images, overlay images, or highlight selected pixels

Syntax
```
Y = step(H,I1,I2)
Y = step(H,I1,I2,OPACITY)
Y = step(H,I1,I2,MASK)
Y = step(H,I1,MASK)
Y = step(H,I1,I2,...,LOCATION)
```

Description   `Y = step(H,I1,I2)` performs the alpha blending operation on images *I1* and *I2*.

`Y = step(H,I1,I2,OPACITY)` uses *OPACITY* input to combine pixel values of *I1* and *I2* when you set the `Operation` property to `Blend` and the `OpacitySource` property to `Input port`.

`Y = step(H,I1,I2,MASK)` uses *MASK* input to overlay *I2* over *I1* when you set the `Operation` property to `Binary mask` and the `MaskSource` property to `Input port`'.

`Y = step(H,I1,MASK)` uses *MASK* input to determine which pixels in *I1* are set to the maximum value supported by their data type when you set the `Operation` property to `Highlight selected pixels` and the `MaskSource` property to `Input port`.

`Y = step(H,I1,I2,...,LOCATION)` uses *LOCATION* input to specify the upper-left corner position of *I2* when you set the `LocationSource` property to `Input port`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose	Compute 2-D autocorrelation of input matrix

Description	The `Autocorrelator` object computes 2-D autocorrelation of input matrix.

Construction

`H = vision.Autocorrelator` returns a System object, `H`, that performs two-dimensional auto-correlation of an input matrix.

`H = vision.Autocorrelator(Name,Value)` returns a 2-D autocorrelation System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`, `Value1`,...,`NameN`,`ValueN`).

Properties

Fixed-Point Properties

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Same as input`, or `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,30)`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],16,15)`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,30)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input` or `Custom`. The default is `Same as input`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Same as input`, `Custom`. The default is `Same as input`.

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

Methods	clone	Create 2-D autocorrelator object with same property values
	getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Compute cross-correlation of two input matrices

Examples Compute the 2D autocorrelation of a matrix.

```
hac2d = vision.Autocorrelator;
x = [1 2;2 1];
Y = step(hac2d, x)
```

Algorithms This object implements the algorithm, inputs, and outputs described on the 2-D Autocorrelation block reference page. The object properties correspond to the block parameters.

See Also vision.Crosscorrelator

vision.Autocorrelator.clone

Purpose   Create 2-D autocorrelator object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an Autocorrelator System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.Autocorrelator.getNumOutputs

Purpose Number of outputs from step method

Syntax N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the Autocorrelator
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.Autocorrelator.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute cross-correlation of two input matrices

Syntax   Y = step(H,X)

Description Y = step(H,X) returns the 2-D autocorrelation, *Y*, of input matrix *X*.

Calling step on an unlocked System object will lock that object.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.Autothresholder

Purpose Convert intensity image to binary image

Description Convert intensity images to binary images. Autothresholding uses Otsu's method, which determines the threshold by splitting the histogram of the input image to minimize the variance for each of the pixel groups.

Construction `H = vision.Autothresholder` returns a System object, `H`, that automatically converts an intensity image to a binary image.

`H = vision.Autothresholder(`*Name*`,`*Value*`)` returns an autothreshold object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties **Operator**

Threshold operator on input matrix values

Specify the condition the object places on the input matrix values as one of `>` | `<=` . The default is `>`.

If you set this property to `>`, and the step method inputs a value greater than the threshold value, the object outputs `1`. If the step method inputs a value less than the threshold value, the object outputs `0`.

If you set this property to `<=`, and the step method inputs a value less than or equal to the threshold value, the object outputs `1`. If the step method inputs a value greater than the threshold value, the object outputs `0`.

ThresholdOutputPort

Enable threshold output

Set this property to `true` to enable the output of the calculated threshold values.

The default is `false`.

EffectivenessOutputPort

Enable threshold effectiveness output

Set this property to true to enable the output of the effectiveness of the thresholding. The default is `false`. This effectiveness metric ranges from `0` to `1`. If every pixel has the same value, the object sets effectiveness metric to `0`. If the image has two pixel values, or the histogram of the image pixels is symmetric, the object sets the effectiveness metric to `1`.

InputRangeSource

Source of input data range

Specify the input data range as one of `Auto | Property`. The default is `Auto`. If you set this property to `Auto`, the object assumes an input range between `0` and `1`, inclusive, for floating point data types. For all other data types, the object sets the input range to the full range of the data type. To specify a different input data range, set this property to `Property`.

InputRange

Input data range

Specify the input data range as a two-element numeric row vector. First element of the input data range vector represents the minimum input value while the second element represents the maximum value. This property applies when you set the `InputRangeSource` property to `Property`.

InputRangeViolationAction

Behavior when input values are out of range

Specify the object's behavior when the input values are outside the expected data range as one of `Ignore | Saturate`. The default is `Saturate`. This property applies when you set the `InputRangeSource` property to `Property`.

ThresholdScaleFactor

Threshold scale factor

Specify the threshold scale factor as a numeric scalar greater than 0. The default is 1. The object multiplies this scalar value with the threshold value computed by Otsu's method. The result becomes the new threshold value. The object does not do threshold scaling. This property is tunable.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero. The default isFloor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. The default isWrap.

Product1DataType

Product-1 word and fraction lengths

This is a constant property with value Custom.

CustomProduct1DataType

Product-1 word and fraction lengths

Specify the product-1 fixed-point type as a signed numerictype object with a Signedness of Auto. The default isnumerictype([],32).

Accumulator1DataType

Accumulator-1 word and fraction lengths

Specify the accumulator-1 fixed-point data type as Same as product 1, Custom. The default isSame as product 1.

CustomAccumulator1DataType

Accumulator-1 word and fraction lengths

Specify the accumulator-1 fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Accumulator1DataType` property to `Custom`. The default is `numerictype([],32)`.

Product2DataType

Product-2 word and fraction lengths

This is a constant property with value `Custom`.

CustomProduct2DataType

Product-2 word and fraction lengths

Specify the product-2 fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. The default is `numerictype([],32)`.

Accumulator2DataType

Accumulator-2 word and fraction lengths

Specify the accumulator-2 fixed-point data type as `Same as product 2`, `Custom`. The default is `Same as product 2`.

CustomAccumulator2DataType

Accumulator-2 word and fraction lengths

Specify the accumulator-2 fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Accumulator2DataType` property to `Custom`. The default is `numerictype([],32)`.

Product3DataType

Product-3 word and fraction lengths

This is a constant property with value `Custom`.

CustomProduct3DataType

Product-3 word and fraction lengths

Specify the product-3 fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. The default is `numerictype([],32)`.

Accumulator3DataType

Accumulator-3 word and fraction lengths

Specify the accumulator-3 fixed-point data type as `Same as product 3`, `Custom`. This property applies when you set the `EffectivenessOutputPort` property to `true`. The default is `Same as product 3`.

CustomAccumulator3DataType

Accumulator-3 word and fraction lengths

Specify the accumulator-3 fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `EffectivenessOutputPort` property to `true`, and when you set the `Accumulator3DataType` property to `Custom`. The default is `numerictype([],32)`.

Product4DataType

Product-4 word and fraction lengths

Specify the product-4 fixed-point data type as `Same as input`, or `Custom`. The default is `Custom`.

CustomProduct4DataType

Product-4 word and fraction lengths

Specify the product-4 fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Product4DataType` property to `Custom`. The default value is `numerictype([],32,15)`.

Accumulator4DataType

Accumulator-4 word and fraction lengths

Specify the accumulator-4 fixed-point data type as `Same as product 4` | `Custom`. The default is `Same as product 4`.

CustomAccumulator4DataType

Accumulator-4 word and fraction lengths

Specify the accumulator-4 fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Accumulator4DataType` property to `Custom`. The default is `numerictype([],16,4)`.

QuotientDataType

Quotient word and fraction lengths

Specify the quotient fixed-point data type as `Custom`.

CustomQuotientDataType

Quotient word and fraction lengths

Specify the quotient fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `QuotientDataType` property to `Custom`. The default is `numerictype([],32)`.

EffectivenessDataType

Effectiveness metric word and fraction lengths

This is a constant property with value `Custom`. This property applies when you set the `EffectivenessOutputPort` property to `true`.

CustomEffectivenessDataType

Effectiveness metric word and fraction lengths

Specify the effectiveness metric fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `EffectivenessOutputPort` property to `true`. The default is `numerictype([],16)`.

vision.Autothresholder

Methods	clone	Create autothresholder object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Convert input intensity image to binary image

Examples

Convert an image of peppers to binary image.

```
img = im2single(rgb2gray(imread('peppers.png')));
imshow(img);
hautoth = vision.Autothresholder;
bin = step(hautoth,img);
pause(2);
figure;imshow(bin);
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Autothreshold block reference page. The object properties correspond to the block parameters, except:

> You can only specify a value of Ignore or Saturate for the InputRangeViolationAction property of the System object. The object does not support the Error and Warn and Saturate options that the corresponding **When data range is exceeded** block parameter offers.

See Also

vision.ColorSpaceConverter

Purpose   Create autothresholder object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Autothresholder System object *C*, with the
                   same property values as *H*. The clone method creates a new unlocked
                   object.

vision.Autothresholder.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                     method.

                     The getNumOutputs method returns a positive integer that is the
                     number of outputs from the step method. This value will change if you
                     alter any properties that turn outputs on or off.

vision.Autothresholder.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
Autothresholder System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Convert input intensity image to binary image

Syntax
```
BW = step(H,I)
[BW,TH] = step(H,I)
[...,EMETRIC] = step(H,I)
```

Description   `BW = step(H,I)` converts input intensity image, `I`, to a binary image, `BW`.

`[BW,TH] = step(H,I)` also returns the threshold, `TH`, when the you set the `ThresholdOutputPort` property to true.

`[...,EMETRIC] = step(H,I)` also returns `EMETRIC`, a metric indicating the effectiveness of thresholding the input image when you set the `EffectivenessOutputPort` property to `true`. When the step method inputs an image having a single gray level, the object can attain the lower bound of the metric (zero). When the step method inputs a two-valued image, the object can attain the upper bound (one).

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose　　　Read video data from binary files

Description　　The `BinaryFileReader` object reads video data from binary files.

Construction　`H = vision.BinaryFileReader` returns a System object, `H`, that reads binary video data from the specified file in `I420` Four Character Code (FOURCC) video format.

`H = vision.BinaryFileReader(`*Name*`,`*Value*`)` returns a binary file reader System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

`H = vision.BinaryFileReader(FILE,`*Name*`,`*Value*`)` returns a binary file reader System object, `H`, with the `Filename` property set to `FILE` and other specified properties set to the specified values.

Properties　　**Filename**

Name of binary file to read from

Specify the name of the binary file as a string. The full path for the file needs to be specified only if the file is not on the MATLAB path. The default is `vipmen.bin`.

VideoFormat

Format of binary video data

Specify the format of the binary video data as `Four character codes`, or `Custom`. The default is `Four character codes`.

FourCharacterCode

Four Character Code video format

Specify the binary file format from the available list of Four Character Code video formats. For more information on Four Character Codes, see http://www.fourcc.org. This property applies when you set the `VideoFormat` property to `Four character codes`.

BitstreamFormat

Format of data as planar or packed

Specify the data format as `Planar` or `Packed`. This property applies when you set the `VideoFormat` property to `Custom`. The default is `Planar`.

OutputSize

Size of output matrix

Specify the size of the output matrix. This property applies when you set the `BitstreamFormat` property to `Packed`.

VideoComponentCount

Number of video components in video stream

Specify the number of video components in the video stream as `1`, `2`, `3` or `4`. This number corresponds to the number of video component outputs. This property applies when you set the `VideoFormat` property to `Custom`. The default is `3`.

VideoComponentBits

Bit size of video components

Specify the bit sizes of video components as an integer valued vector of length N, where N is the value of the `VideoComponentCount` property. This property applies when you set the `VideoFormat` property to `Custom`. The default is `[8 8 8]`.

VideoComponentSizes

Size of output matrix

Specify the size of the output matrix. This property must be set to an N-by-2 array, where N is the value of the `VideoComponentCount` property. Each row of the matrix corresponds to the size of that video component, with the first element denoting the number of rows and the second element denoting the number of columns. This property applies when you

set the `VideoFormat` property to `Custom` and the `BitstreamFormat` property to `Planar`. The default is `[120 160; 60 80; 60 80]`.

VideoComponentOrder

Specify how to arrange the components in the binary file. This property must be set to a vector of length *N*, where *N* is set according to how you set the `BitstreamFormat` property. When you set the `BitStreamFormat` property to `Planar`, you must set *N* equal to the value of the `VideoComponentCount` property. Otherwise, you can set *N* equal to or greater than the value of the `VideoComponentCount` property.

This property applies when you set the `VideoFormat` property to `Custom`. The default is `[1 2 3]`.

InterlacedVideo

Whether data stream represents interlaced video

Set this property to `true` if the video stream represents interlaced video data. This property applies when you set the `VideoFormat` property to `Custom`. The default is `false`.

LineOrder

How to fill binary file

Specify how to fill the binary file as `Top line first`, or `Bottom line first`. If this property is set to `Top line first`, the System object first fills the binary file with the first row of the video frame. If it is set to `Bottom line first`, the System object first fills the binary file with the last row of the video frame. The default is `Top line first`.

SignedData

Whether input data is signed

Set this property to `true` if the input data is signed. This property applies when you set the `VideoFormat` property to `Custom`. The default is `false`.

vision.BinaryFileReader

ByteOrder

Byte ordering as little endian or big endian

Specify the byte ordering in the output binary file as `Little endian`, `Big endian`. This property applies when you set the `VideoFormat` property to `Custom`. The default is `Little endian`.

PlayCount

Number of times to play the file

Specify the number of times to play the file as a positive integer or `inf`. The default is `1`.

Methods

clone	Create binary file reader object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isDone	End-of-file status (logical)
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset to beginning of file
step	Read video components from a binary file

Examples

Read in a binary video file and play it back on the screen

```
hbfr = vision.BinaryFileReader('ecolicells.bin');
hbfr.VideoFormat = 'Custom';
hbfr.VideoComponentCount = 1;
```

```
hbfr.VideoComponentBits = 16;
hbfr.VideoComponentSizes = [442 538];
hbfr.VideoComponentOrder = 1;

hvp = vision.VideoPlayer;
while ~isDone(hbfr)
  y = step(hbfr);
  step(hvp,y);
end

release(hbfr); % close the input file
release(hvp); % close the video display
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Read Binary File block reference page. The object properties correspond to the block parameters.

See Also   vision.VideoFileReader | vision.BinaryFileWriter

vision.BinaryFileReader.clone

Purpose   Create binary file reader object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a BinaryFileReader System object C, with the
                   same property values as H. The clone method creates a new unlocked
                   object with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.BinaryFileReader.getNumOutputs

Purpose Number of outputs from step method

Syntax N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   End-of-file status (logical)

Syntax   `TF = isDone(H)`

Description   `TF = isDone(H)` returns `true` if the `BinaryFileReader` System object, `H`, has reached the end of the binary file. If `PlayCount` property is set to a value greater than `1`, this method will return `true` every time the end is reached.

vision.BinaryFileReader.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
BinaryFileReader System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Reset to beginning of file

Syntax   reset(H)

Description   reset(H) System object *H* to the beginning of the specified file.

vision.BinaryFileReader.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles
or hardware connections) and allows all properties and input
characteristics to be changed.

> **Note** You can use the release method on a System object in code
> generated from MATLAB, but once you release its resources, you cannot
> use that System object again.

Purpose   Read video components from a binary file

Syntax
```
[Y,Cb,Cr] = step(H)
Y = step(H)
[Y,Cb] = step(H)
[Y,Cb,Cr] = step(H)
[Y,Cb,Cr,Alpha] = step(H)
[..., EOF] = step(H)
```

Description   [Y,Cb,Cr] = step(H) reads the luma, *Y* and chroma, *Cb* and *Cr* components of a video stream from the specified binary file when you set the VideoFormat property to 'Four character codes'.

Y = step(H) reads the video component *Y* from the binary file when you set the VideoFormat property to Custom and the VideoComponentCount property to 1.

[Y,Cb] = step(H) reads video the components *Y* and *Cb* from the binary file when you set the VideoFormat property to Custom and the VideoComponentCount property to 2.

[Y,Cb,Cr] = step(H) reads the video components *Y*, *Cb* and *Cr* when you set the VideoFormat property to Custom, and the VideoComponentCount property to 3.

[Y,Cb,Cr,Alpha] = step(H) reads the video components *Y*, *Cb*, *Cr* and *Alpha* when you set the VideoFormat property to Custom and the VideoComponentCount property to 4.

[..., EOF] = step(H) also returns the end-of-file indicator, *EOF*. *EOF* is set to true each time the output contains the last video frame in the file.

> **Note** The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose   Write binary video data to files

Description   The `BinaryFileWriter` object writes binary video data to files.

Construction   `H = vision.BinaryFileWriter` returns a System object, `H`, that writes
binary video data to an output file, `output.bin` in the `I420` Four
Character Code format.

`H = vision.BinaryFileWriter(`*Name*`,`*Value*`)` returns a binary file
writer System object, `H`, with each specified property set to the specified
value. You can specify additional name-value pair arguments in any
order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

`H = vision.BinaryFileWriter(FILE,`*Name*`,`*Value*`)` returns a binary
file writer System object, `H`, with the `Filename` property set to `FILE` and
other specified properties set to the specified values.

Properties   **Filename**

Name of binary file to write to

Specify the name of the binary file as a string. The default is the
file `output.bin`.

VideoFormat

Format of binary video data

Specify the format of the binary video data as `Four character
codes`, or `Custom`. The default is `Four character codes`.

FourCharacterCode

Four Character Code video format

Specify the binary file format from the available list of Four
Character Code video formats. For more information on Four
Character Codes, see http://www.fourcc.org. This property applies
when you set the `VideoFormat` property to `Four character
codes`.

BitstreamFormat

Format of data as planar or packed

Specify the data format as `Planar` or `Packed`. This property applies when you set the `VideoFormat` property to `Custom`. The default is `Planar`.

VideoComponentCount

Number of video components in video stream

Specify the number of video components in the video stream as `1`, `2`, `3` or `4`. This number corresponds to the number of video component outputs. This property applies when you set the `VideoFormat` property to `Custom`. The default is `3`.

VideoComponentBitsSource

How to specify the size of video components

Indicate how to specify the size of video components as `Auto` or `Property`. If this property is set to `Auto`, each component will have the same number of bits as the input data type. Otherwise, the number of bits for each video component is specified using the `VideoComponentBits` property. This property applies when you set the `VideoFormat` property to `Custom`. The default is `Auto`.

VideoComponentBits

Bit size of video components

Specify the bit size of video components using a vector of length N, where N is the value of the `VideoComponentCount` property. This property applies when you set the `VideoComponentBitsSource` property to `Property`. The default is `[8 8 8]`.

VideoComponentOrder

Arrange video components in binary file

Specify how to arrange the components in the binary file. This property must be set to a vector of length *N*, where *N* is set according to how you set the `BitstreamFormat` property. When you set the `BitStreamFormat` property to `Planar`, you must set

N equal to the value of the `VideoComponentCount` property. Otherwise, you can set *N* equal to or greater than the value of the `VideoComponentCount` property.

This property applies when you set the `VideoFormat` property to `Custom`. The default is `[1 2 3]`.

InterlacedVideo

Whether data stream represents interlaced video

Set this property to true if the video stream represents interlaced video data. This property applies when you set the `VideoFormat` property to `Custom`. The default is `false`.

LineOrder

How to fill binary file

Specify how to fill the binary file as `Top line first`, or `Bottom line first`. If this property is set to `Top line first`, the object first fills the binary file with the first row of the video frame. Otherwise, the object first fills the binary file with the last row of the video frame. The default is `Top line first`.

SignedData

Whether input data is signed

Set this property to true if the input data is signed. This property applies when you set the `VideoFormat` property to `Custom`. The default is `false`.

ByteOrder

Byte ordering as little endian or big endian

Specify the byte ordering in the output binary file as `Little endian`, or `Big endian`. This property applies when you set the `VideoFormat` property to `Custom`. The default is `Little endian`.

vision.BinaryFileWriter

Methods

clone	Create binary file writer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Write video frame to output file

Examples   Write video to a binary video file

```
filename = fullfile(tempdir,'output.bin');
hbfr = vision.BinaryFileReader;
hbfw = vision.BinaryFileWriter(filename);

while ~isDone(hbfr)
  [y,cb,cr] = step(hbfr);
  step(hbfw,y,cb,cr);
end

release(hbfr); % close the input file
release(hbfw); % close the output file
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Write Binary File block reference page. The object properties correspond to the block parameters.

See Also   vision.VideoFileWriter | vision.BinaryFileReader

Purpose   Create binary file writer object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a BinaryFileWriter System object C, with the same property values as *H*. The clone method creates a new unlocked object.

vision.BinaryFileWriter.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                     method.

                     The getNumOutputs method returns a positive integer that is the
                     number of outputs from the step method. This value will change if you
                     alter any properties that turn outputs on or off.

vision.BinaryFileWriter.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
BinaryFileWriter System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose　　Allow property value and input characteristics changes

Syntax　　`release(H)`

Description　　`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note　You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.BinaryFileWriter.step

Purpose   Write video frame to output file

Syntax   step(H,Y,Cb,Cr)
                 step(H,Y)
                 step(H,Y,Cb)
                 step(H,Y,Cb,Cr)
                 step(H,Y,Cb,Cr,Alpha)

Description step(H,Y,Cb,Cr) writes one frame of video to the specified output file.
                 Y , Cb, Cr represent the luma (Y) and chroma (Cb and Cr) components
                 of a video stream. This option applies when you set the VideoFormat
                 property to Four character codes.

                 step(H,Y) writes video component Y to the output file when the
                 VideoFormat property is set to Custom and the VideoComponentCount
                 property is set to 1.

                 step(H,Y,Cb) writes video components Y and Cb to the output file when
                 the VideoFormat property is Custom and the VideoComponentCount
                 property is set to 2.

                 step(H,Y,Cb,Cr) writes video components Y , Cb and Cr to the
                 output file when the VideoFormat property is set to Custom and the
                 VideoComponentCount property is set to 3.

                 step(H,Y,Cb,Cr,Alpha) writes video components Y , Cb, Cr and Alpha
                 to the output file when the VideoFormat property is set to Custom, and
                 the VideoComponentCount property is set to 4.

                 Note The object performs an initialization the first time the step
                 method is executed. This initialization locks nontunable properties and
                 input specifications, such as dimensions, complexity, and data type
                 of the input data. If you change a nontunable property or an input
                 specification, the System object issues an error. To change nontunable
                 properties or inputs, you must first call the release method to unlock
                 the object.

Purpose   Properties of connected regions

Description   The BlobAnalysis object computes statistics for connected regions in a binary image.

Use the step syntax below with input binary image, BW, blob analysis object, H, and any optional properties. The step method computes and returns statistics of the input binary image depending on the property values specified. The order of the returned values when there are multiple outputs are in the order they are described below:

AREA= step(H,BW) computes the AREA of the blobs found in input binary image BW when the AreaOutputPort property is set to true.

[...,CENTROID] = step(H,BW) computes the CENTROID of the blobs found in input binary image BW when the CentroidOutputPort property is set to true.

[...,BBOX] = step(H,BW) computes the bounding box BBOX of the blobs found in input binary image BW when the BoundingBoxOutputPort property is set to true.

[...,MAJORAXIS] = step(H,BW) computes the major axis length MAJORAXIS of the blobs found in input binary image BW when the MajorAxisLengthOutputPort property is set to true.

[...,MINORAXIS] = step(H,BW) computes the minor axis length MINORAXIS of the blobs found in input binary image BW when the MinorAxisLengthOutputPort property is set to true.

[...,ORIENTATION] = step(H,BW) computes the ORIENTATION of the blobs found in input binary image BW when the OrientationOutputPort property is set to true.

[...,ECCENTRICITY] = step(H,BW) computes the ECCENTRICITY of the blobs found in input binary image BW when the EccentricityOutputPort property is set to true.

[...,EQDIASQ] = step(H,BW) computes the equivalent diameter squared EQDIASQ of the blobs found in input binary image BW when the EquivalentDiameterSquaredOutputPort property is set to true.

[...,EXTENT] = step(H,BW) computes the EXTENT of the blobs found in input binary image BW when the ExtentOutputPort property is set to true.

[...,PERIMETER] = step(H,BW) computes the PERIMETER of the blobs found in input binary image BW when the PerimeterOutputPort property is set to true.

[...,LABEL] = step(H,BW) returns a label matrix LABEL of the blobs found in input binary image BW when the LabelMatrixOutputPort property is set to true.

[AREA,CENTROID,BBOX] = step(H,BW) returns the area, centroid and the bounding box of the blobs, when the AreaOutputPort, CentroidOutputPort and BoundingBoxOutputPort properties are set to true. You can use this to calculate multiple statistics.

Input/Output	Format	Supported Data Types
BW	Vector or matrix that represents a binary image	Boolean
AREA	Vector that represents the number of pixels in labeled regions	32-bit signed integer
CENTROID	M-by-2 matrix of centroid coordinates, where M represents the number of blobs	• Double-precision floating point • Single-precision floating point • Fixed point

Input/Output	Format	Supported Data Types
BBOX	*M*-by-4 matrix of [x y width height] bounding box coordinates, where *M* represents the number of blobs and [x y] represents the upper left corner of the bounding box.	32-bit signed integer
MAJORAXIS	Vector that represents the lengths of major axes of ellipses	• Double-precision floating point • Single-precision floating point
MINORAXIS	Vector that represents the lengths of minor axes of ellipses	Same as MajorAxis port
ORIENTATION	Vector that represents the angles between the major axes of the ellipses and the *x*-axis.	Same as MajorAxis port
ECCENTRICITY	Vector that represents the eccentricities of the ellipses	Same as MajorAxis port
EQDIASQ	Vector that represents the equivalent diameters squared	Same as Centroid port

Input/Output	Format	Supported Data Types
EXTENT	Vector that represents the results of dividing the areas	Same as Centroid port
PERIMETER	Vector containing an estimate of the perimeter length, in pixels, for each blob	Same as Centroid port
LABEL	Label matrix	8-, 16-, or 32-bit unsigned integer

Construction   H = vision.BlobAnalysis returns a blob analysis System object, H, used to compute statistics for connected regions in a binary image.

H = vision.BlobAnalysis(*Name*,*Value*) returns a blob analysis object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **AreaOutputPort**

Return blob area

Setting this property to true outputs the area of the blobs. The default is true.

CentroidOutputPort

Return coordinates of blob centroids

Set this property to true to output the coordinates of the centroid of the blobs. The default is true.

BoundingBoxOutputPort

Return coordinates of bounding boxes

Set this property to `true` to output the coordinates of the bounding boxes. The default is `true`.

MajorAxisLengthOutputPort

Return vector whose values represent lengths of ellipses' major axes

Set this property to `true` to output a vector whose values represent the lengths of the major axes of the ellipses that have the same normalized second central moments as the labeled regions. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.

MinorAxisLengthOutputPort

Return vector whose values represent lengths of ellipses' minor axes

Set this property to `true` to output a vector whose values represent the lengths of the minor axes of the ellipses that have the same normalized second central moments as the labeled regions. This property is available when the `OutputDataType` property is `double` or `single`. The default is `false`.

OrientationOutputPort

Return vector whose values represent angles between ellipses' major axes and x-axis

Set this property to true to output a vector whose values represent the angles between the major axes of the ellipses and the x-axis. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.

EccentricityOutputPort

Return vector whose values represent ellipses' eccentricities

Set this property to `true` to output a vector whose values represent the eccentricities of the ellipses that have the same

second moments as the region. This property applies when you set the `OutputDataType` property to `double` or `single`. The default is `false`.

EquivalentDiameterSquaredOutputPort

Return vector whose values represent equivalent diameters squared

Set this property to `true` to output a vector whose values represent the equivalent diameters squared. The default is `false`.

ExtentOutputPort

Return vector whose values represent results of dividing blob areas by bounding box areas

Set this property to `true` to output a vector whose values represent the results of dividing the areas of the blobs by the area of their bounding boxes. The default is `false`.

PerimeterOutputPort

Return vector whose values represent estimates of blob perimeter lengths

Set this property to `true` to output a vector whose values represent estimates of the perimeter lengths, in pixels, of each blob. The default is `false`.

OutputDataType

Output data type of statistics

Specify the data type of the output statistics as `double`, `single`, or `Fixed point`. Area and bounding box outputs are always an `int32` data type. `Major axis length`, `Minor axis length`, `Orientation` and `Eccentricity` do not apply when you set this property to `Fixed point`. The default is `double`.

Connectivity

Which pixels are connected to each other

Specify connectivity of pixels as 4 or 8. The default is 8.

LabelMatrixOutputPort

Return label matrix

Set this property to true to output the label matrix. The default is false.

MaximumCount

Maximum number of labeled regions in each input image

Specify the maximum number of blobs in the input image as a positive scalar integer. The maximum number of blobs the object outputs depends on both the value of this property, and on the size of the input image. The number of blobs the object outputs may be limited by the input image size. The default is 50.

MinimumBlobArea

Minimum blob area in pixels

Specify the minimum blob area in pixels. The default is 0. This property is tunable.

MaximumBlobArea

Maximum blob area in pixels

Specify the maximum blob area in pixels. The default is intmax('uint32'). This property is tunable.

ExcludeBorderBlobs

Exclude blobs that contain at least one border pixel

Set this property to true if you do not want to label blobs that contain at least one border pixel. The default is false.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`. This property applies when you set the OutputDataType property to `Fixed point`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the OutputDataType property to `Fixed point`.

ProductDataType

Product word and fraction lengths

This property is constant and is set to `Custom`. This property applies when you set the OutputDataType property to `Fixed point` and the `EquivalentDiameterSquaredOutputPort` property to `true`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the OutputDataType property to `Fixed point` and the `EquivalentDiameterSquaredOutputPort` property to `true`. The default is `numerictype([],32,16)`.

AccumulatorDataType

Accumulator word and fraction lengths

This property is constant and is set to `Custom`. This property applies when you set the OutputDataType property to `Fixed point`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Fixed point`. The default is `numerictype([],32,0)`.

CentroidDataType

Centroid word and fraction lengths

Specify the centroid output's fixed-point data type as `Same as accumulator`, `Custom`. This property applies when you set the `OutputDataType` property to `Fixed point` and the `CentroidOutputPort` property to `true`. The default is `Custom`.

CustomCentroidDataType

Centroid word and fraction lengths

Specify the centroid output's fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Fixed point` and the `CentroidDataType` property to `Custom` and when the `CentroidOutputPort` property to `true`. The default is `numerictype([],32,16)`.

EquivalentDiameterSquaredDataType

Equivalent diameter squared word and fraction lengths

Specify the equivalent diameters squared output's fixed-point data type as `Same as accumulator`, `Same as product`, `Custom`. This property applies when you set the `OutputDataType` property to `Fixed point` and the `EquivalentDiameterSquaredOutputPort` property to `true`. The default is `Same as product`.

CustomEquivalentDiameterSquaredDataType

Equivalent diameter squared word and fraction lengths

Specify the equivalent diameters squared output's fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Fixed point` and the

EquivalentDiameterSquaredDataType property to Custom and when the EquivalentDiameterSquaredOutputPort property to true. The default is numerictype([],32,16).

ExtentDataType

Extent word and fraction lengths

Specify the extent output's fixed-point data type as Same as accumulator or Custom. This property applies when you set the OutputDataType property to Fixed point and the ExtentOutputPort property to true.

CustomExtentDataType

Extent word and fraction lengths

Specify the extent output's fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Fixed point, the ExtentDataType property to Custom and the ExtentOutputPort property to true. The default is numerictype([],16,14).

PerimeterDataType

Perimeter word and fraction lengths

Specify the perimeter output's fixed-point data type as Same as accumulator or Custom. This property applies when you set the OutputDataType property to Fixed point and the PerimeterOutputPort property to true. The default is Custom.

CustomPerimeterDataType

Perimeter word and fraction lengths

Specify the perimeter output's fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Fixed point, the PerimeterDataType property to Custom and the PerimeterOutputPort property to true. The default is numerictype([],32,16).

Methods	clone	Create blob analysis object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Compute and returns statistics of input binary image

Examples   Find the centroid of a blob.

```
hblob = vision.BlobAnalysis;
hblob.AreaOutputPort = false;
hblob.BoundingBoxOutputPort = false;
img = logical([0 0 0 0 0 0; ...
        0 1 1 1 1 0; ...
        0 1 1 1 1 0; ...
        0 1 1 1 1 0; ...
        0 0 0 0 0 0]);
centroid = step(hblob, img); % [x y] coordinates of the centroid
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Blob Analysis block reference page. The object properties correspond to the block parameters, except:

• The **Warn if maximum number of blobs is exceeded** block parameter does not have a corresponding object property. The object does not issue a warning.

- The **Output blob statistics as a variable-size signal** block parameter does not have a corresponding object property.

See Also   `vision.Autothresholder` | `vision.ConnectedComponentLabeler`

Purpose   Create blob analysis object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a BlobAnalysis System object *C*, with the same
                   property values as *H*. The clone method creates a new unlocked object.

vision.BlobAnalysis.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.BlobAnalysis.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the BlobAnalysis
                  System objects.

                  The isLocked method returns a logical value that indicates whether
                  input attributes and nontunable properties for the object are locked. The
                  object performs an internal initialization the first time the step method
                  is executed. This initialization locks nontunable properties and input
                  specifications, such as dimensions, complexity, and data type of the
                  input data. After locking, the isLocked method returns a true value.

Purpose　　　　Allow property value and input characteristics changes

Syntax　　　　`release(H)`

Description　　`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note　You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.BlobAnalysis.step

Purpose   Compute and returns statistics of input binary image

Syntax
```
AREA = step(H,BW)
[...,CENTROID] = step(H,BW)
[...,BBOX] = step(H,BW)
[...,MAJORAXIS] = step(H,BW)
[...,MINORAXIS] = step(H,BW)
[...,ORIENTATION] = step(H,BW)
[...,ECCENTRICITY] = step(H,BW)
[...,EQDIASQ] = step(H,BW)
[...,EXTENT] = step(H,BW)
[...,PERIMETER] = step(H,BW)
[...,LABEL] = step(H,BW)
[...,NUMBLOBS] = step(H,BW)
[AREA,CENTROID,BBOX] = step(H,BW)
```

Description   AREA = step(H,BW) computes the AREA of the blobs found in input
binary image BW when the AreaOutputPort property is set to true.

[...,CENTROID] = step(H,BW) computes the CENTROID of the blobs
found in input binary image BW when the CentroidOutputPort property
is set to true.

[...,BBOX] = step(H,BW) computes the bounding box BBOX of the
blobs found in input binary image BW when the BoundingBoxOutputPort
property is set to true.

[...,MAJORAXIS] = step(H,BW) computes the major axis length
MAJORAXIS of the blobs found in input binary image BW when the
MajorAxisLengthOutputPort property is set to true.

[...,MINORAXIS] = step(H,BW) computes the minor axis length
MINORAXIS of the blobs found in input binary image BW when the
MinorAxisLengthOutputPort property is set to true.

[...,ORIENTATION] = step(H,BW) computes the ORIENTATION of the
blobs found in input binary image BW when the OrientationOutputPort
property is set to true.

[...,ECCENTRICITY] = step(H,BW) computes the ECCENTRICITY of the blobs found in input binary image BW when the EccentricityOutputPort property is set to true.

[...,EQDIASQ] = step(H,BW) computes the equivalent diameter squared EQDIASQ of the blobs found in input binary image BW when the EquivalentDiameterSquaredOutputPort property is set to true.

[...,EXTENT] = step(H,BW) computes the EXTENT of the blobs found in input binary image BW when the ExtentOutputPort property is set to true.

[...,PERIMETER] = step(H,BW) computes the PERIMETER of the blobs found in input binary image BW when the PerimeterOutputPort property is set to true.

[...,LABEL] = step(H,BW) returns a label matrix LABEL of the blobs found in input binary image BW when the LabelMatrixOutputPort property is set to true.

[...,NUMBLOBS] = step(H,BW) returns the number of blobs found in the input binary image BW when the NumBlobsOutputPort property is set to true.

[AREA,CENTROID,BBOX] = step(H,BW) returns the area, centroid and the bounding box of the blobs, when the AreaOutputPort, CentroidOutputPort and BoundingBoxOutputPort properties are set to true. You can use this to calculate multiple statistics.

The step method computes and returns statistics of the input binary image depending on the property values specified. The different options can be used simultaneously. The order of the returned values when there are multiple outputs are in the order they are described.

> **Note** The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose   Estimate motion between images or video frames

Description   The BlockMatcher object estimates motion between images or video frames.

Construction   H = vision.BlockMatcher returns a System object, H, that estimates motion between two images or two video frames. The object performs this estimation using a block matching method by moving a block of pixels over a search region.

H = vision.BlockMatcher(*Name*,*Value*) returns a block matcher System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **ReferenceFrameSource**

Reference frame source

Specify the source of the reference frame as one of Input port | Property. When you set the ReferenceFrameSource property to Input port a reference frame input must be specified to the step method of the block matcher object. The default is Property.

ReferenceFrameDelay

Number of frames between reference and current frames

Specify the number of frames between the reference frame and the current frame as a scalar integer value greater than or equal to zero. This property applies when you set the ReferenceFrameSource property to Property.

The default is 1.

SearchMethod

Best match search method

Specify how to locate the block of pixels in frame $k+1$ that best matches the block of pixels in frame k. You can specify the search

method as `Exhaustive` or `Three-step`. If you set this property to `Exhaustive`, the block matcher object selects the location of the block of pixels in frame $k+1$. The block matcher does so by moving the block over the search region one pixel at a time, which is computationally expensive.

If you set this property to `Three-step`, the block matcher object searches for the block of pixels in frame $k+1$ that best matches the block of pixels in frame k using a steadily decreasing step size. The object begins with a step size approximately equal to half the maximum search range. In each step, the object compares the central point of the search region to eight search points located on the boundaries of the region and moves the central point to the search point whose values is the closest to that of the central point. The object then reduces the step size by half, and begins the process again. This option is less computationally expensive, though sometimes it does not find the optimal solution.

The default is `Exhaustive`.

BlockSize

Block size

Specify the size of the block in pixels.

The default is `[17 17]`.

Overlap

Input image subdivision overlap

Specify the overlap (in pixels) of two subdivisions of the input image.

The default is `[0 0]`.

MaximumDisplacement

Maximum displacement search

Specify the maximum number of pixels that any center pixel in a block of pixels can move, from image to image or from frame to

frame. The block matcher object uses this property to determine the size of the search region.

The default is [7 7].

MatchCriteria

Match criteria between blocks

Specify how the System object measures the similarity of the block of pixels between two frames or images. Specify as one of Mean square error (MSE) | Mean absolute difference (MAD). The default is Mean square error (MSE).

OutputValue

Motion output form

Specify the desired form of motion output as one of Magnitude-squared | Horizontal and vertical components in complex form. The default is Magnitude-squared.

Fixed-Point Properties

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Same as input | Custom. The default is Custom. This property applies when you set the MatchCriteria property to Mean square error (MSE).

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the MatchCriteria property to Mean square error (MSE) and the ProductDataType property to Custom.

The default is numerictype([],32,0).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Custom`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`.

The default is `numerictype([],32,0)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Custom`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. The `numerictype` object should be unsigned if the `OutputValue` property is `Magnitude-squared` and, signed if it is `Horizontal and vertical components in complex form`.

The default is `numerictype([],8)`.

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Saturate`

Methods

clone	Create block matcher object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Compute motion of input image

Examples

Estimate Motion

Estimate and draw lines of motion for two images.

1

Read in an image, convert it to grayscale, and translate it to create second image.

```
img1 = im2double(rgb2gray(imread('onion.png')));
htran = vision.GeometricTranslator('Offset', [5 5], ...
   'OutputSize', 'Same as input image');
hbm = vision.BlockMatcher( ...
   'ReferenceFrameSource','Input port','BlockSize',[35 35]);
hbm.OutputValue = ...
   'Horizontal and vertical components in complex form';
halphablend = vision.AlphaBlender;
```

2

Offset the first image by [5 5] pixels to create second image.

```
img2 = step(htran, img1);
```

3

Compute motion of the two images.

```
motion = step(hbm, img1, img2);
```

4

Blend two images.

```
img12 = step(halphablend, img2, img1);
```

5

Show the directions.

```
[X Y] = meshgrid(1:35:size(img1, 2), 1:35:size(img1, 1));
imshow(img12); hold on;
quiver(X(:), Y(:), real(motion(:)), imag(motion(:)), 0); hold off;
```

Algorithms   This object implements the algorithm, inputs, and outputs described
on the Block Matching block reference page. The object properties
correspond to the block parameters.

See Also   vision.OpticalFlow

Purpose   Create block matcher object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a BlockMatcher System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.BlockMatcher.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.BlockMatcher.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the BlockMatcher
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.BlockMatcher.step

Purpose   Compute motion of input image

Syntax   V = step(H,I)
                     C = step(H,I)
                     Y = step(H,I,IREF)

Description   V = step(H,I) computes the motion of input image *I* from one video
                    frame to another, and returns *V* as a matrix of velocity magnitudes.

                    C = step(H,I) computes the motion of input image *I* from one video
                    frame to another, and returns *C* as a complex matrix of horizontal
                    and vertical components, when you set the OutputValue property to
                    Horizontal and vertical components in complex form.

                    Y = step(H,I,IREF) computes the motion between input image *I*
                    and reference image *IREF* when you set the ReferenceFrameSource
                    property to Input port.

                    Note The object performs an initialization the first time the step
                    method is executed. This initialization locks nontunable properties and
                    input specifications, such as dimensions, complexity, and data type
                    of the input data. If you change a nontunable property or an input
                    specification, the System object issues an error. To change nontunable
                    properties or inputs, you must first call the release method to unlock
                    the object.

Purpose   Trace object boundary

Description   The boundary tracer object traces object boundaries in binary images.

Use the step syntax below with input image BW, starting point STARTPT, boundary tracer object, H, and any optional properties.

PTS = step(H,BW,STARTPT) traces the boundary of an object in a binary image, BW. The input matrix, STARTPT, specifies the starting point for tracing the boundary. STARTPT is a two-element vector of [x y] coordinates of the initial point on the object boundary. The step method outputs PTS, an *M*-by-2 matrix of [x y] coordinates of the boundary points. In this matrix, *M* is the number of traced boundary pixels. *M* is less than or equal to the value specified by the MaximumPixelCount property.

Construction   H = vision.BoundaryTracer returns a System object, H, that traces the boundaries of objects in a binary image. In this image nonzero pixels belong to an object and zero-valued pixels constitute the background.

H = vision.BoundaryTracer(*Name*,*Value*) returns an object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Connectivity**

How to connect pixels to each other

Specify which pixels are connected to each other as one of 4 | 8. The default is 8. Set this property to 4 to connect a pixel to the pixels on the top, bottom, left, and right. Set this property to 8 to connect a pixel to the pixels on the top, bottom, left, right, and diagonally.

InitialSearchDirection

First search direction to find next boundary pixel

Specify the first direction in which to look to find the next boundary pixel that is connected to the starting pixel.

When you set the `Connectivity` property to `8`, this property accepts a value of `North`, `Northeast`, `East`, `Southeast`, `South`, `Southwest`, `West`, or `Northwest`.

When you set the `Connectivity` property to `4`, this property accepts a value of `North`, `East`, `South`, or `West`

TraceDirection

Direction in which to trace the boundary

Specify the direction in which to trace the boundary as one of `Clockwise` | `Counterclockwise`. The default is `Clockwise`.

MaximumPixelCount

Maximum number of boundary pixels

Specify the maximum number of boundary pixels as a scalar integer greater than 1. The object uses this value to preallocate the number of rows of the output matrix, `Y`. This preallocation enables the matrix to hold all the boundary pixel location values.

The default is `500`.

NoBoundaryAction

How to fill empty spaces in output matrix

Specify how to fill the empty spaces in the output matrix, `Y` as one of `None` | `Fill with last point found` | `Fill with user-defined values`. The default is `None`. If you set this property to `None`, the object takes no action. Thus, any element that does not contain a boundary pixel location has no meaningful value. If you set this property to `Fill with last point found`, the object fills the remaining elements with the position of the last boundary pixel. If you set this property to `Fill with user-defined values`, you must specify the values in the `FillValues` property.

FillValues

Value to fill in remaining empty elements in output matrix

Set this property to a scalar value or two-element vector to fill in the remaining empty elements in the output matrix Y. This property applies when you set the NoBoundaryAction property to Fill with user-defined values.

The default is [0 0].

Methods

clone	Create boundary tracer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Trace object boundary in binary image

Examples

Trace the boundary around an image of a coin:

```
I = imread('coins.png'); % read the image
hautoth = vision.Autothresholder;
BW = step(hautoth, I);   % threshold the image
[y, x]= find(BW,1);      % select a starting point for the trace

% Determine the boundaries
  hboundtrace = vision.BoundaryTracer;
  PTS = step(hboundtrace, BW, [x y]);
```

```
% Display the results
  figure, imshow(BW);
  hold on; plot(PTS(:,1), PTS(:,2), 'r', 'Linewidth',2);
  hold on; plot(x,y,'gx','Linewidth',2); % show the starting point
```

Algorithms   This object implements the algorithm, inputs, and outputs described
on the Trace Boundary block reference page. The object properties
correspond to the block parameters.

See Also   vision.EdgeDetector | vision.ConnectedComponentLabeler |
vision.Autothresholder

Purpose   Create boundary tracer object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a BoundaryTracer System object *C*, with the
same property values as *H*. The clone method creates a new unlocked
object.

vision.BoundaryTracer.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.BoundaryTracer.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the BoundaryTracer System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
                     or hardware connections) and allows all properties and input
                     characteristics to be changed.

Note You can use the release method on a System object in code
generated from MATLAB, but once you release its resources, you cannot
use that System object again.

---

vision.BoundaryTracer.step

Purpose   Trace object boundary in binary image

Syntax   PTS = step(H,BW,STARTPT)

Description   PTS = step(H,BW,STARTPT) traces the boundary of an object in a binary image BW. The second input matrix, STARTPT, specified the starting point for tracing the boundary. STARTPT is a two-element vector of [x y] coordinates of the initial point on the object boundary. The step method outputs PTS, an *M*-by-2 matrix of [x y] coordinates of the boundary points, where *M* is the number of traced boundary pixels. *M* is less than or equal to the value specified by the MaximumPixelCount property.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Detect objects using the Viola-Jones algorithm

Description   The cascade object detector uses the Viola-Jones algorithm to detect people's faces, noses, eyes, mouth, or upper body. You can also use the `trainCascadeObjectDetector` function to train a custom classifier to use with this System object. For details on how the function works, see "Train a Cascade Object Detector".

Construction   `detector = vision.CascadeObjectDetector` creates a System object, `detector`, that detects objects using the Viola-Jones algorithm. The `ClassificationModel` property controls the type of object to detect. By default, the detector is configured to detect faces.

`detector = vision.CascadeObjectDetector(MODEL)` creates a System object, `detector`, configured to detect objects defined by the input string, `MODEL`. The `MODEL` input describes the type of object to detect. There are several valid `MODEL` strings, such as 'FrontalFaceCART', 'UpperBody', and 'ProfileFace'. See the `ClassificationModel` property description for a full list of available models.

`detector = vision.CascadeObjectDetector(XMLFILE)` creates a System object, `detector`, and configures it to use the custom classification model specified with the `XMLFILE` input. The `XMLFILE` can be created using the `trainCascadeObjectDetector` function or OpenCV (Open Source Computer Vision) training functionality. You must specify a full or relative path to the `XMLFILE`, if it is not on the MATLAB path.

`detector = vision.CascadeObjectDetector(`*Name*`,`*Value*`)` configures the cascade object detector object properties. You specify these properties as one or more name-value pair arguments. Unspecified properties have default values.

To detect a feature:

1 Define and set up your cascade object detector using the constructor.

2 Call the `step` method with the input image, I, the cascade object detector object, `detector`, points `PTS`, and any optional properties. See the syntax below for using the `step` method.

Use the step syntax with input image, I, the selected Cascade object detector object, and any optional properties to perform detection.

`BBOX = step(detector,I)` returns `BBOX`, an *M*-by-4 matrix defining *M* bounding boxes containing the detected objects. This method performs multiscale object detection on the input image, I. Each row of the output matrix, `BBOX`, contains a four-element vector, [x y width height], that specifies in pixels, the upper-left corner and size of a bounding box. The input image I, must be a grayscale or truecolor (RGB) image.

Properties

ClassificationModel

Trained cascade classification model

Specify the name of the model as a string. This value sets the classification model for the detector. You may set this string to an XML file containing a custom classification model, or to one of the valid model strings listed below.

Frontal Face (CART)

ClassificationModel String	Image Size Used to Train	Model Description
FrontalFaceCART (Default)	[20 20]	Detects faces that are upright and forward facing. This model is composed of weak classifiers, based on the classification and regression tree analysis (CART). These classifiers use Haar features to encode facial features. CART-based classifiers provide the ability to model higher-order dependencies between facial features. [1]

Frontal Face (LBP)

ClassificationModel String	Image Size Used to Train	Model Description
FrontalFaceLBP	[24 24]	Detects faces that are upright and forward facing. This model is composed of weak classifiers, based on a decision stump. These classifiers use local binary patterns (LBP) to encode facial features. LBP features can provide robustness against variation in illumination.[2]

Upper Body

ClassificationModel String	Image Size Used to Train	Model Description
UpperBody	[18 22]	Detects the upper-body region, which is defined as the head and shoulders area. This model uses Haar features to encode the details of the head and shoulder region. Because it uses more features around the head, this model is more robust against pose changes, e.g. head rotations/tilts. [3]

Eye Pair

ClassificationModel String	Image Size Used to Train	Model Description
EyePairBig EyePairSmall	[11 45] [5 22]	Detects a pair of eyes. The EyePairSmall model is trained using a smaller image. This enables the model to detect smaller eyes than the EyePairBig model can detect.[4]

Single Eye

ClassificationModel String	Image Size Used to Train	Model Description
LeftEye RightEye	[12 18]	Detects the left and right eye separately. These models are composed of weak classifiers, based on a decision stump These classifiers use Haar features to encode details.[4]

Single Eye (CART)

ClassificationModel String	Image Size Used to Train	Model Description
LeftEyeCART RightEyeCART	[20 20]	Detects the left and right eye separately. The weak classifiers that make up these models are CART-trees. Compared to decision stumps, CART-tree-based classifiers are better able to model higher-order dependencies. [5]

Profile Face

ClassificationModel String	Image Size Used to Train	Model Description
ProfileFace	[20 20]	Detects upright face profiles. This model is composed of weak classifiers, based on a decision stump. These classifiers use Haar features to encode face details.

Mouth

ClassificationModel String	Image Size Used to Train	Model Description
Mouth	[15 25]	Detects the mouth. This model is composed of weak classifiers, based on a decision stump, which use Haar features to encode mouth details.[4]

Nose

ClassificationModel String	Image Size Used to Train	Model Description
Nose	[15 18]	This model is composed of weak classifiers, based on a decision stump, which use Haar features to encode nose details.[4]

Default: FrontalFaceCART

MinSize

Size of smallest detectable object

Specify the size of the smallest object to detect. This value must be given in pixels, as a two-element vector, [height width]. It must be greater than or equal to the image size used to train the model. Use this property to reduce computation time when you know the minimum object size prior to processing the image. When you do not specify a value for this property, the detector sets it to the size of the image used to train the classification model. This property is tunable.

For details explaining the relationship between setting the size of the detectable object and the ScaleFactor property, see "Algorithms" on page 2-139 section.

Default: [ ]

MaxSize

Size of largest detectable object

Specify the size of the largest object to detect. This value must be given in pixels, as a two-element vector, [height width]. Use this property to reduce computation time when you know the maximum object size prior to processing the image. When you do

not specify a value for this property, the detector sets it to `size(I)`. This property is tunable.

For details explaining the relationship between setting the size of the detectable object and the `ScaleFactor` property, see the "Algorithms" on page 2-139 section.

Default: `[]`

ScaleFactor

Scaling for multiscale object detection

Specify the factor used to incrementally scale the search region. The detector scales the search region at increments between `MinSize` and `MaxSize` using the following relationship:

$$search\ region = \mathsf{round}((Training\ Size)*(\mathsf{ScaleFactor}^N))$$

N is the current increment, an integer greater than zero, and *Training Size* is the image size used to train the classification model. This property is tunable.

Default: `1.1`

MergeThreshold

Detection threshold

Specify a threshold value as a scalar integer. This value defines the criteria needed to declare a final detection in an area where there are multiple detections around an object. Groups of colocated detections that meet the threshold are merged to produce one bounding box around the target object. Increasing this threshold may help suppress false detections by requiring that the target object be detected multiple times during the multiscale detection phase. When you set this property to `0`, all detections are returned without performing thresholding or merging operation. This property is tunable.

Default: `4`

vision.CascadeObjectDetector

Methods

clone	Create cascade object detector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Detect objects using the Viola-Jones algorithm

Examples

Face Detection

Detect faces in an image using the frontal face classification model.

Create a detector object.

```
faceDetector = vision.CascadeObjectDetector;
```

Read input image.

```
I = imread('visionteam.jpg');
```

Detect faces.

```
bboxes = step(faceDetector, I);
```

Annotate detected faces.

```
IFaces = insertObjectAnnotation(I, 'rectangle', bboxes, 'Face');
figure, imshow(IFaces), title('Detected faces');
```

Upper Body Detection

Detect upper body in an image using the upper body classification model.

Create a detector object

```
bodyDetector = vision.CascadeObjectDetector('UpperBody');
bodyDetector.MinSize = [60 60];
bodyDetector.ScaleFactor = 1.05;
```

Read input image and detect upper body.

```
I2 = imread('visionteam.jpg');
bboxBody = step(bodyDetector, I2);
```

Annotate detected upper bodies.

```
IBody = insertObjectAnnotation(I2, 'rectangle',bboxBody,'Upper Body');
figure, imshow(IBody), title('Detected upper bodies');
```

Algorithms   Classification Model Training

Each model is trained to detect a specific type of object. The classification models are trained by extracting features from a set of known images. These extracted features are then fed into a learning algorithm to train the classification model. Computer Vision System Toolbox software uses the Viola-Jones cascade object detector. This detector includes Haar-like [6] features and cascade of classifiers trained using boosting.

The image size used to train the classifiers defines the smallest region containing the object. Training image sizes vary according to the application, type of target object, and available positive images. You must set the MinSize property to a value greater than or equal to the image size used to train the model.

Cascade of Classifiers

This object uses a cascade of classifiers to efficiently process image regions for the presence of a target object. Each stage in the cascade applies increasingly more complex binary classifiers, which allows the algorithm to rapidly reject regions that do not contain the target. If the desired object is not found at any stage in the cascade, the detector immediately rejects the region and processing is terminated. By terminating, the object avoids invoking computation-intensive classifiers further down the cascade.
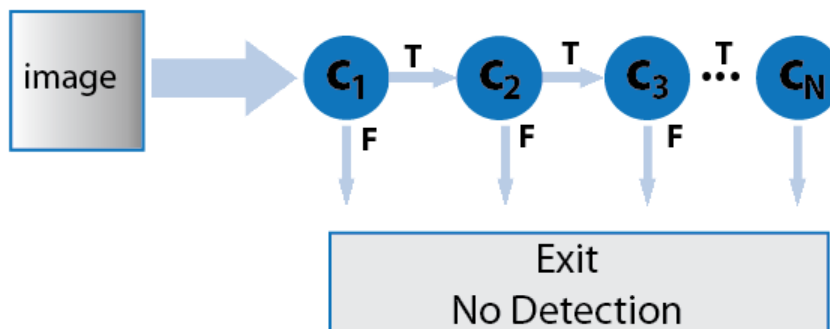
Multiscale Object Detection

The detector incrementally scales the input image to locate target objects. At each scale increment, a sliding window, whose size is the same as the training image size, scans the scaled image to locate objects. The `ScaleFactor` property determines the amount of scaling between successive increments.

The search region size is related to the `ScaleFactor` in the following way:

$$search\ region = \text{round}((Training\ Size)*(\texttt{ScaleFactor}^N))$$

N is the current increment, an integer greater than zero, and *Training Size* is the image size used to train the classification model.
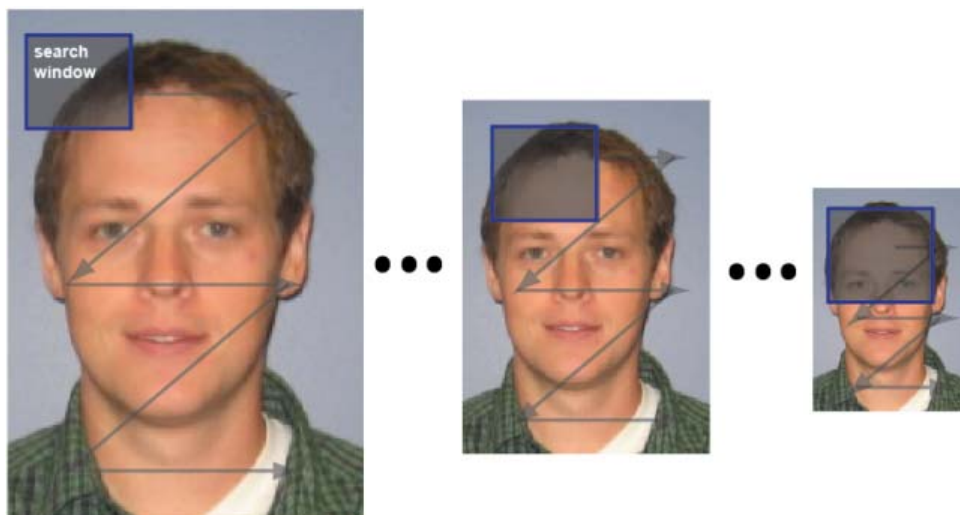
The search window traverses the image for each scaled increment.

Relationship Between MinSize, MaxSize, and ScaleFactor

Understanding the relationship between the size of the object to detect and the scale factor will help you set the properties accordingly. The MinSize and MaxSize properties limit the size range of the object to detect. Ideally, these properties are modified to reduce computation time when you know the approximate object size prior to processing the image. They are not designed to provide precise filtering of results, based on object size. The behavior of these properties is affected by the ScaleFactor. The scale factor determines the quantization of the search window sizes.

$$search\ region = \mathtt{round}((Training\ Size)*(\mathtt{ScaleFactor}^N))$$

The actual range of returned object sizes may not be exactly what you select for the MinSize and MaxSize properties. For example, a ScaleFactor value of 1.1 with a 24x24 training size, and an N value from 1 to 5, the search region calculation would be:

For a `ScaleFactor` value of `1.1` with a 24x24 training size, for `5` increments, the search region calculation would be:

```
>> search region = round(24*1.1.^(1:5))

>> 26 29 32 35 39
```

If you were to set `MaxSize` to `34`, due to the search region quantization, the actual maximum object size used by the algorithm would be `32`.

Merge Detection Threshold

For each increment in scale, the search window traverses over the image producing multiple detections around the target object. The multiple detections are merged into one bounding box per target object. You can use the `MergeThreshold` property to control the number of detections required before combining or rejecting the detections. The size of the final bounding box is an average of the sizes of the bounding boxes for the individual detections and lies between `MinSize` and `MaxSize`.
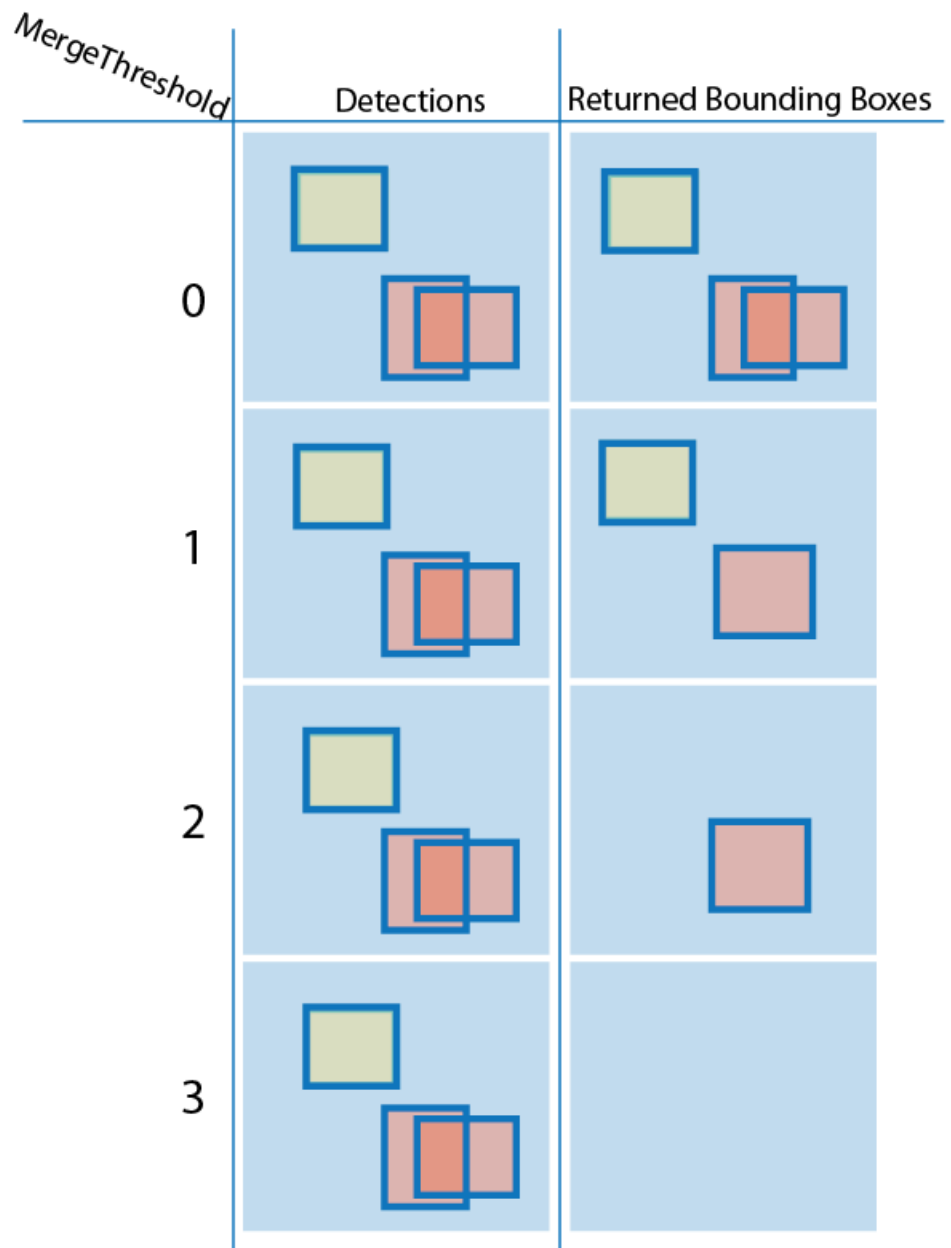
References

[1] Lienhart R., Kuranov A., and V. Pisarevsky "Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.", *Proceedings of the 25th DAGM Symposium on Pattern Recognition.* Magdeburg, Germany, 2003.

[2] Ojala Timo, Pietikäinen Matti, and Mäenpää Topi, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns" . In *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002.* Volume 24, Issue 7, pp. 971-987.

[3] Kruppa H., Castrillon-Santana M., and B. Schiele. "Fast and Robust Face Finding via Local Context". *Proceedings of the Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, 2003, pp. 157–164.

[4] Castrillón Marco, Déniz Oscar, Guerra Cayetano, and Hernández Mario, "ENCARA2: Real-time detection of multiple faces at different resolutions in video streams". In *Journal of Visual Communication and Image Representation*, 2007 (18) 2: pp. 130-140.

[5] Yu Shiqi "Eye Detection." Shiqi Yu's Homepage.http://yushiqi.cn/research/eyedetection

[6] Viola, Paul and Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. Volume: 1, pp.511–518.

[7] OpenCV (Open Source Computer Vision Library), OpenCV.

See Also   "Train a Cascade Object Detector" **|** `vision.ShapeInserter` **|** `integralImage`

Purpose   Create cascade object detector object with same property values

Syntax   `C = clone(H)`

Description   `C = clone(H)` creates an `CascadeObjectDetector` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object with uninitialized states.

vision.CascadeObjectDetector.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                 the step method.

                 The getNumInputs method returns a positive integer that is the number
                 of expected inputs (not counting the object itself) to the step method.
                 This value will change if you alter any properties that turn inputs on or
                 off. You must call the step method with the number of input arguments
                 equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.CascadeObjectDetector.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the MedianFilter System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
                  or hardware connections) and allows all properties and input
                  characteristics to be changed.

Note You cannot use the release method on System objects in
Embedded MATLAB.

---

vision.CascadeObjectDetector.step

Purpose   Detect objects using the Viola-Jones algorithm

Syntax   BBOX = step(DETECTOR,I)

Description   BBOX = step(DETECTOR,I) returns BBOX, an *M*-by-4 matrix defining *M* bounding boxes containing the detected objects. This method performs multi-scale object detection on the input image, I.

Each row of the output matrix, BBOX, contains a four-element vector, [x y width height], that specifies in pixels, the upper left corner and size of a bounding box. The input image I, must be a grayscale or truecolor (RGB) image.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Downsample or upsample chrominance components of images

Description   The `ChromaResampler` object downsamples or upsample chrominance components of images.

Construction   `H = vision.ChromaResampler` returns a chroma resampling System object, `H`, that downsamples or upsamples chroma components of a YCbCr signal to reduce the bandwidth and storage requirements.

    `H = vision.ChromaResampler(`*Name*`,`*Value*`)` returns a chroma resampling System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Resampling**

        Resampling format

        To downsample the chrominance components of images, set this property to one of the following:

```
[4:4:4 to 4:2:2]
[4:4:4 to 4:2:0 (MPEG1)]
[4:4:4 to 4:2:0 (MPEG2)]
[4:4:4 to 4:1:1]
[4:2:2 to 4:2:0 (MPEG1)]
[4:2:2 to 4:2:0 (MPEG2)]
```

        To upsample the chrominance components of images, set this property to one of the following:

```
[4:2:2 to 4:4:4]
[4:2:0 (MPEG1) to 4:4:4]
[4:2:0 (MPEG2) to 4:4:4]
[4:1:1 to 4:4:4]
[4:2:0 (MPEG1) to 4:2:2]
[4:2:0 (MPEG2) to 4:2:2]
```

The default is [4:4:4 to 4:2:2]

InterpolationFilter

Method used to approximate missing values

Specify the interpolation method used to approximate the missing chrominance values as one of Pixel replication | Linear. The default is Linear. When you set this property to Linear, the object uses linear interpolation to calculate the missing values. When you set this property to Pixel replication, the object replicates the chrominance values of the neighboring pixels to create the upsampled image. This property applies when you upsample the chrominance values.

AntialiasingFilterSource

Lowpass filter used to prevent aliasing

Specify the lowpass filter used to prevent aliasing as one of Auto | Property | None. The default is Auto. When you set this property to Auto, the object uses a built-in lowpass filter. When you set this property to Property, the coefficients of the filters are specified by the HorizontalFilterCoefficients and VerticalFilterCoefficients properties. When you set this property to None, the object does not filter the input signal. This property applies when you downsample the chrominance values.

HorizontalFilterCoefficients

Horizontal filter coefficients

Specify the filter coefficients to apply to the input signal. This property applies when you set the Resampling property to one of [4:4:4 to 4:2:2] | [4:4:4 to 4:2:0 (MPEG1)] | [4:4:4 to 4:2:0 (MPEG2)] | [4:4:4 to 4:1:1] and the AntialiasingFilterSource property to Property. The default is [0.2 0.6 0.2].

VerticalFilterCoefficients

Vertical filter coefficients

Specify the filter coefficients to apply to the input signal. This property applies when you set the `Resampling` property to one of `[4:4:4 to 4:2:0 (MPEG1)]` | `[4:4:4 to 4:2:0 (MPEG2)]` | `[4:2:2 to 4:2:0 (MPEG1)]` | `[4:2:2 to 4:2:0 (MPEG2)]` and the `AntialiasingFilterSource` property to `Property`. The default is `[0.5 0.5]`.

TransposedInput

Input is row-major format

Set this property to `true` when the input contains data elements from the first row first, then data elements from the second row second, and so on through the last row. Otherwise, the object assumes that the input data is stored in column-major format. The default is `false`.

Methods		
	clone	Create chroma resampling object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Resample input chrominance components

Examples

Resample the chrominance components of an image.

```
H = vision.ChromaResampler;
hcsc = vision.ColorSpaceConverter;
x = imread('peppers.png');
```

```
 x1 = step(hcsc, x);
 [Cb, Cr] = step(H, x1(:,:,2), x1(:,:,3));
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Chroma Resampling block reference page. The object properties correspond to the block parameters.

See Also   vision.ColorSpaceConverter

Purpose   Create chroma resampling object with same property values

Syntax   `C = clone(H)`

Description   `C = clone(H)` creates a `ChromaResampler` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object.

vision.ChromaResampler.getNumInputs

Purpose Number of expected inputs to step method

Syntax N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.ChromaResampler.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
ChromaResampler System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.ChromaResampler.step

Purpose　　Resample input chrominance components

Syntax　　`[Cb1,Cr1] = step(H,Cb,Cr)`

Description　　`[Cb1,Cr1] = step(H,Cb,Cr)` resamples the input chrominance components `Cb` and `Cr` and returns `Cb1` and `Cr1` as the resampled outputs.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose   Convert color information between color spaces

Description   The `ColorSpaceConverter` object converts color information between
                     color spaces.

Construction   `H = vision.ColorSpaceConverter` returns a System object, `H`, that
                     converts color information from RGB to YCbCr using the conversion
                     standard `Rec. 601 (SDTV)`.

                     `H = vision.ColorSpaceConverter(`*Name*`,`*Value*`)` returns a color space
                     conversion object, `H`, with each specified property set to the specified
                     value. You can specify additional name-value pair arguments in any
                     order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Conversion**

                     Color space input and output conversion

                     Specify the conversion color spaces as one of the following:

                        ```
                        [RGB to YCbCr]
                        [YCbCr to RGB]
                        [RGB to intensity]
                        [RGB to HSV]
                        [HSV to RGB]
                        [sRGB to XYZ]
                        [XYZ to sRGB]
                        [sRGB to L*a*b*]
                        [L*a*b* to sRGB]
                        ```
                     Note that the `R`, `G`, `B` and `Y` (luma) signal components in the
                     preceding color space conversions are gamma corrected. The
                     default is `[RGB to YCbCr]`.

                     WhitePoint

                     Reference white point

                     Specify the reference white point as one of `D50` | `D55` | `D65`.
                     The default is `D65`. These values are reference white standards

known as *tristimulus* values that serve to define the color "white" to correctly map color space conversions. This property applies when you set the `Conversion` property to `[sRGB to L*a*b*]` or `[L*a*b* to sRGB]`.

ConversionStandard

Standard for RGB to YCbCr conversion

Specify the standard used to convert the values between the RGB and YCbCr color spaces as one of `Rec. 601 (SDTV)` | `Rec. 709 (HDTV)`. The default is `Rec. 601 (SDTV)`. This property applies when you set the `Conversion` property to `[RGB to YCbCr]` or `[YCbCr to RGB]`.

ScanningStandard

Scanning standard for RGB-to-YCbCr conversion

Specify the scanning standard used to convert the values between the RGB and YCbCr color spaces as one of `[1125/60/2:1]` | `[1250/50/2:1]`. The default is `[1125/60/2:1]`. This property applies when you set the `ConversionStandard` property to `Rec. 709 (HDTV)`.

Methods

clone	Create color space converter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

release	Allow property value and input characteristics changes
step	Convert color space of input image

Examples   Convert an image from an RGB to an intensity color space.

```
i1 = imread('pears.png');
imshow(i1);

hcsc = vision.ColorSpaceConverter;
hcsc.Conversion = 'RGB to intensity';
i2 = step(hcsc, i1);

pause(2);
imshow(i2);
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Color Space Conversion block reference page. The object properties correspond to the block parameters, except:

The **Image signal** block parameter allows you to specify whether the block accepts the color video signal as One multidimensional signal or Separate color signals. The object does not have a property that corresponds to the **Image signal** block parameter. You must always provide the input image to the step method of the object as a single multidimensional signal.

See Also   vision.Autothresholder

vision.ColorSpaceConverter.clone

Purpose   Create color space converter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a ColorSpaceConverter System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

vision.ColorSpaceConverter.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                   ColorSpaceConverter System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

vision.ColorSpaceConverter.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Convert color space of input image

Syntax   C2 = step(H,C1)

Description   C2 = step(H,C1) converts a multidimensional input image C1 to a
                     multidimensional output image C2 . C1 and C2 are images in different
                     color spaces.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

---

vision.ConnectedComponentLabeler

Purpose   Label and count the connected regions in a binary image

Description   The `ConnectedComponentLabeler` object labels and counts the connected regions in a binary image. The System object can output a label matrix, where pixels equal to `0` represent the background, pixels equal to `1` represent the first object, pixels equal to `2` represent the second object, and so on. The object can also output a scalar that represents the number of labeled objects.

Construction   `H = vision.ConnectedComponentLabeler` returns a System object, `H`, that labels and counts connected regions in a binary image.

`H = vision.ConnectedComponentLabeler(`*Name*`,`*Value*`)` returns a label System object, `H`, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Connectivity**

Which pixels are connected to each other

Specify which pixels are connected to each other as either `4` or `8`. If a pixel should be connected to the pixels on the top, bottom, left, and right, set this property to `4`. If a pixel should be connected to the pixels on the top, bottom, left, right, and diagonally, set this property to `8`. The default is `8`.

LabelMatrixOutputPort

Enable output of label matrix

Set to `true` to output the label matrix. Both the `LabelMatrixOutputPort` and `LabelCountOutputPort` properties cannot be set to `false` at the same time. The default is `true`.

LabelCountOutputPort

Enable output of number of labels

Set to `true` to output the number of labels. Both the `LabelMatrixOutputPort` and `LabelCountOutputPort` properties cannot be set to `false` at the same time. The default is `true`.

OutputDataType

Output data type

Set the data type of the output to one of `Automatic`, `uint32`, `uint16`, `uint8`. If this property is set to `Automatic`, the System object determines the appropriate data type for the output. If it is set to `uint32`, `uint16`, or `uint8`, the data type of the output is 32-, 16-, or 8-bit unsigned integers, respectively. The default is `Automatic`.

OverflowAction

Behavior if number of found objects exceeds data type size of output

Specify the System object's behavior if the number of found objects exceeds the maximum number that can be represented by the output data type as `Use maximum value of the output data type`, or `Use zero`. If this property is set to `Use maximum value of the output data type`, the remaining regions are labeled with the maximum value of the output data type. If this property is set to `Use zero`, the remaining regions are labeled with zeroes. This property applies when you set the `OutputDataType` property to `uint16` or `uint8`. The default is `Use maximum value of the output data type`.

Methods

clone	Create connected component labeler object with same property values
getNumInputs	Number of expected inputs to step method

vision.ConnectedComponentLabeler

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Label and count connected regions in input

Examples   Label connected regions in an image.

```
img = logical([0 0 0 0 0 0 0 0 0 0 0 0 0; ...
        0 1 1 1 1 0 0 0 0 0 0 1 0; ...
      0 1 1 1 1 1 0 0 0 0 1 1 0; ...
      0 1 1 1 1 1 0 0 0 1 1 1 0; ...
      0 1 1 1 1 0 0 0 1 1 1 1 0; ...
      0 0 0 0 0 0 0 1 1 1 1 1 0; ...
      0 0 0 0 0 0 0 0 0 0 0 0 0])
hlabel = vision.ConnectedComponentLabeler;
hlabel.LabelMatrixOutputPort = true;
hlabel.LabelCountOutputPort = false;
labeled = step(hlabel, img)
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Label block reference page. The object properties correspond to the block parameters, except:

- The `LabelCountOutputPort` and `LabelMatrixOutputPort` object properties correspond to the **Output** block parameter.

See Also   `vision.AutoThresholder` | `vision.BlobAnalysis`

Purpose   Create connected component labeler object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a ConnectedComponentLabeler System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.ConnectedComponentLabeler.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.ConnectedComponentLabeler.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.ConnectedComponentLabeler.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
ConnectedComponentLabeler System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.ConnectedComponentLabeler.step

Purpose   Label and count connected regions in input

Syntax   L = step(H,BW)
                   COUNT = step(H,BW)
                   [L,COUNT] = step(H,BW)

Description   L = step(H,BW) outputs the matrix, L for input binary image BW when
                   the LabelMatrixOutputPort property is true.

                   COUNT = step(H,BW) outputs the number of distinct,
                   connected regions found in input binary image BW when you set the
                   LabelMatrixOutputPort property to false and LabelCountOutputPort
                   property to true.

                   [L,COUNT] = step(H,BW) outputs both the L matrix and number
                   of distinct, connected regions, COUNT when you set both the
                   LabelMatrixOutputPort property and LabelCountOutputPort to true.

                   Note The object performs an initialization the first time the step
                   method is executed. This initialization locks nontunable properties and
                   input specifications, such as dimensions, complexity, and data type
                   of the input data. If you change a nontunable property or an input
                   specification, the System object issues an error. To change nontunable
                   properties or inputs, you must first call the release method to unlock
                   the object.

Purpose   Adjust image contrast by linear scaling

Description   The ContrastAdjuster object adjusts image contrast by linearly scaling pixel values between upper and lower limits. Pixel values that are above or below this range are saturated to the upper or lower limit values.

Construction   H = vision.ContrastAdjuster returns a contrast adjustment object, H. This object adjusts the contrast of an image by linearly scaling the pixel values between the maximum and minimum values of the input data.

H = vision.ContrastAdjuster(*Name*,*Value*) returns a contrast adjustment object, H, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **InputRange**

How to specify lower and upper input limits

Specify how to determine the lower and upper input limits as one of Full input data range [min max] | Custom | Range determined by saturating outlier pixels. The default is Full input data range [min max].

CustomInputRange

Lower and upper input limits

Specify the lower and upper input limits as a two-element vector of real numbers. The first element corresponds to the lower input limit, and the second element corresponds to the upper input limit. This property applies only when you set the InputRange property to Custom. This property is tunable.

PixelSaturationPercentage

Percentage of pixels to consider outliers

Specify the percentage of pixels to consider outliers, as a two-element vector. This property only applies when you set

the `InputRange` property to `Range determined by saturating outlier pixels`.

The contrast adjustment object calculates the lower input limit. This calculation ensures that the maximum percentage of pixels with values smaller than the lower limit can only equal the value of the first element.

Similarly, the object calculates the upper input limit. This calculation ensures that the maximum percentage of pixels with values greater than the upper limit is can only equal the value of the second element.

The default is `[1 1]`.

HistogramNumBins

Number of histogram bins

Specify the number of histogram bins used to calculate the scaled input values.

The default is `256`.

OutputRangeSource

How to specify lower and upper output limits

Specify how to determine the lower and upper output limits as one of `Auto | Property`. The default is `Auto`. If you set the value of this property to `Auto`, the object uses the minimum value of the input data type as the lower output limit. The maximum value of the input data type indicates the upper output limit.

OutputRange

Lower and upper output limits

Specify the lower and upper output limits as a two-element vector of real numbers. The first element corresponds to the lower output limit, and the second element corresponds to the upper output limit. This property only applies when you set

the `OutputRangeSource` property to `Property`. This property is tunable.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Wrap`

ProductInputDataType

Product input word and fraction lengths

Specify the product input fixed-point data type as `Custom`.

CustomProductInputDataType

Product input word and fraction lengths

Specify the product input fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`.

The default is `numerictype([],32,16)`.

ProductHistogramDataType

Product histogram word and fraction lengths

Specify the product histogram fixed-point data type as `Custom`. This property only applies when you set the `InputRange` property to `Range determined by saturating outlier pixels`.

CustomProductHistogramDataType

Product histogram word and fraction lengths

Specify the product histogram fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when you set the `InputRange` property to `Range determined by saturating outlier pixels`.

The default is `numerictype([],32,16)`.

Methods		
	clone	Create contrast adjuster with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Adjust contrast in input image

Examples   Use contrast adjuster to enhance image quality:

```
hcontadj = vision.ContrastAdjuster;
x = imread('pout.tif');
y = step(hcontadj, x);

imshow(x); title('Original Image');
figure, imshow(y);
title('Enhanced image after contrast adjustment');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Contrast Adjustment block reference page. The object properties correspond to the block parameters.

See Also   vision.HistogramEqualizer

vision.ContrastAdjuster.clone

Purpose   Create contrast adjuster with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a ContrastAdjuster System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.ContrastAdjuster.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   `TF = isLocked(H)`

Description   `TF = isLocked(H)` returns the locked status, `TF` of the
`ContrastAdjuster` System object.

The `isLocked` method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the `step` method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the `isLocked` method returns a `true` value.

vision.ContrastAdjuster.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Adjust contrast in input image

Syntax   Y = step(H,X)

Description   Y = step(H,X) performs contrast adjustment of input *X* and returns
                    the adjusted image *Y*.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

---

vision.Convolver

Purpose	Compute 2-D discrete convolution of two input matrices
Description	The Convolver object computes 2-D discrete convolution of two input matrices.
Construction	H = vision.Convolver returns a System object, H, that performs two-dimensional convolution on two inputs.
	H = vision.Convolver(*Name,Value*) returns a 2-D convolution System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **OutputSize**

Specify dimensions of output

This property controls the size of the output scalar, vector, or matrix produced as a result of the convolution between the two inputs. You can set this property to one of Full | Same as first input | Valid. The default is Full. When you set this property to Full, the object outputs the full two-dimensional convolution with dimension, (*Ma+Mb*-1, *Na+Nb*-1). When you set this property to Same as first input, the object outputs the central part of the convolution with the same dimensions as the first input. When you set this property to Valid, the object outputs only those parts of the convolution that are computed without the zero-padded edges of any input. For this case, the object outputs dimension, (*Ma-Mb*+1, *Na-Nb*+1). (*Ma*, *Na*) is the size of the first input matrix and (*Mb*, *Na*) is the size of the second input matrix.

Normalize

Whether to normalize the output

Set to true to normalize the output. The default is false.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Wrap`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as first input` | `Custom`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as first input` | `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as first input` | `Custom`. The default is `Custom`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],32,12)`.

Methods		
	clone	Create 2-D convolver object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Compute 2-D convolution of input matrices

Examples   Compute the 2D convolution of two matrices.

```
hconv2d = vision.Convolver;
x1 = [1 2;2 1];
x2 = [1 -1;-1 1];
y = step(hconv2d, x1, x2)
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the 2-D Convolution block reference page. The object properties correspond to the block parameters.

See Also   vision.Crosscorrelator | vision.AutoCorrelator

vision.Convolver.clone

Purpose   Create 2-D convolver object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Convolver System object *C*, with the same
                     property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.Convolver.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the Convolver
                System object.

                The isLocked method returns a logical value that indicates whether
                input attributes and nontunable properties for the object are locked. The
                object performs an internal initialization the first time the step method
                is executed. This initialization locks nontunable properties and input
                specifications, such as dimensions, complexity, and data type of the
                input data. After locking, the isLocked method returns a true value.

vision.Convolver.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles
or hardware connections) and allows all properties and input
characteristics to be changed.

Note You can use the release method on a System object in code
generated from MATLAB, but once you release its resources, you cannot
use that System object again.

Purpose   Compute 2-D convolution of input matrices

Syntax   Y = step(HCONV2D,X1,X2)

Description   Y = step(HCONV2D,X1,X2) computes 2-D convolution of input matrices
                   X1 and *X2*.

> **Note** The object performs an initialization the first time the step
> method is executed. This initialization locks nontunable properties and
> input specifications, such as dimensions, complexity, and data type
> of the input data. If you change a nontunable property or an input
> specification, the System object issues an error. To change nontunable
> properties or inputs, you must first call the release method to unlock
> the object.

vision.CornerDetector

Purpose   Detect corner features

Description   The corner detector object finds corners in a grayscale image. It returns corner locations as a matrix of [x y] coordinates. The object finds corners in an image using the Harris corner detection (by Harris & Stephens), minimum eigenvalue (by Shi & Tomasi), or local intensity comparison (Features from Accelerated Segment Test, FAST by Rosten & Drummond) method.

Construction   cornerDetector = vision.CornerDetector returns a corner detector object, cornerDetector, that finds corners in an image using a Harris corner detector.

cornerDetector = vision.CornerDetector(*Name*,*Value*) configures the corner detector object properties. You specify these properties as one or more name-value pair arguments. Unspecified properties have default values.

To detect corners:

1 Define and set up your corner detector using the constructor.

2 Call the step method with the input image, I, the corner detector object, cornerDetector, and any optional properties. See the syntax below for using the step method.

LOC = step(cornerDetector,I) returns corners in the grayscale input image I. The step method outputs the location of detected corners, specified as an *M*-by-2 matrix of [x y] coordinates, where *M* is the number of detected corners. *M* can be less than or equal to the value specified by the MaximumCornerCount property.

METRIC = step(cornerDetector,I) returns a matrix, METRIC, with corner metric values. This applies when you set the MetricMatrixOutputPort property to true. The output, METRIC, represents corner strength. The size of the METRIC matrix is the same as that of the input image.

[LOC,METRIC] = step(cornerDetector,I) returns the locations of the corners in the *M*-by-2 matrix, LOC, and the corner metric matrix in METRIC. This applies when you set both the CornerLocationOutputPort and MetricMatrixOutputPort properties to true.

Properties

Method

Method to find corner values

Specify the method to find the corner values. Specify as one of Harris corner detection (Harris & Stephens) | Minimum eigenvalue (Shi & Tomasi) | Local intensity comparison (Rosten & Drummond).

Default: Harris corner detection (Harris & Stephens)

Sensitivity

Harris corner sensitivity factor

Specify the sensitivity factor, *k*, used in the Harris corner detection algorithm as a scalar numeric value such that $0 < k < 0.25$. The smaller the value of *k*, the more likely the algorithm will detect sharp corners. This property only applies when you set the Method property to Harris corner detection (Harris & Stephens). This property is tunable.

Default: 0.04

SmoothingFilterCoefficients

Smoothing filter coefficients

Specify the filter coefficients for the separable smoothing filter as a real-valued numeric vector. The vector must have an odd number of elements and a length of at least 3. For more information, see fspecial. This property applies only when you set the Method property to either Harris corner detection (Harris & Stephens) or Minimum eigenvalue (Shi & Tomasi).

Default: Output of fspecial('gaussian',[1 5],1.5)

IntensityThreshold

Intensity comparison threshold

Specify a positive scalar value. The object uses the intensity threshold to find valid bright or dark surrounding pixels. This property applies only when you set the `Method` property to `Local intensity comparison (Rosten & Drummond)`. This property is tunable.

Default: `0.1`

MaximumAngleThreshold

Maximum angle in degrees for valid corner

Specify the maximum angle in degrees to indicate a corner as one of `22.5 | 45.0 | 67.5 | 90.0 | 112.5 | 135.0 | 157.5`. This property only applies when you set the `Method` property to `Local intensity comparison (Rosten & Drummond)`. This property is tunable.

Default: `157.5`

CornerLocationOutputPort

Enable output of corner location

Set this property to `true` to output the corner location after corner detection. This property and the `MetricMatrixOutputPort` property cannot both be set to `false`.

Default: `true`

MetricMatrixOutputPort

Enable output of corner metric matrix

Set this property to `true` to output the metric matrix after corner detection. This property and the `CornerLocationOutputPort` property cannot both be set to `false`.

Default: `false`

MaximumCornerCount

Maximum number of corners to detect

Specify the maximum number of corners to detect as a positive scalar integer value. This property only applies when you set the `CornerLocationOutputPort` property to `true`.

Default: `200`

CornerThreshold

Minimum metric value to indicate a corner

Specify a positive scalar number for the minimum metric value that indicates a corner. This property applies only when you set the `CornerLocationOutputPort` property to `true`. This property is tunable.

Default: `0.0005`

NeighborhoodSize

Size of suppressed region around detected corner

Specify the size of the neighborhood around the corner metric value over which the object zeros out the values. The neighborhood size is a two element vector of positive odd integers, [*r c*]. Here, *r* indicates the number of rows in the neighborhood, and *c* indicates the number of columns. This property only applies when you set the `CornerLocationOutputPort` property to `true`.

Default: `[11 11]`

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`.

Default: `Floor`

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap | Saturate`.

Default: `Wrap`

CoefficientsDataType

Coefficients word and fraction lengths

Specify the coefficients fixed-point data type as one of `Same word length as input | Custom`. This property applies only when you do not set the `Method` property to `Local intensity comparison (Rosten & Drummond)`.

Default: `Custom`

CustomCoefficientsDataType

Coefficients word and fraction lengths

Specify the coefficients fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies only when you do not set the `Method` property to `Local intensity comparison (Rosten & Drummond)` and you set the `CoefficientsDataType` property to `Custom`.

Default: `numerictype([],16)`

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as input | Custom`. This property applies when you do not set the `Method` property to `Local intensity comparison (Rosten & Drummond)`.

Default: `Custom`

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you do not set the `Method` property to `Local intensity comparison`

(Rosten & Drummond) and you set the ProductDataType property to Custom.

Default: numerictype([],32,0)

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of Same as input | Custom.

Default: Custom

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object. This property applies when you set the AccumulatorDataType property to Custom.

Default: numerictype([],32,0)

MemoryDataType

Memory word and fraction lengths

Specify the memory fixed-point data type as one of Same as input | Custom. This property applies when you do not set the Method property to Local intensity comparison (Rosten & Drummond).

Default: Custom

CustomMemoryDataType

Memory word and fraction lengths

Specify the memory fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only you do not set the Method property to Local intensity comparison (Rosten & Drummond) and you set the MemoryDataType property to Custom.

Default: numerictype([],32,0)

MetricOutputDataType

Metric output word and fraction lengths

Specify the metric output fixed-point data type as one of `Same as accumulator` | `Same as input` | `Custom`.

Default: `Same as accumulator`

CustomMetricOutputDataType

Metric output word and fraction lengths

Specify the metric output fixed-point type as a signed, scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `MetricOutputDataType` property to `Custom`.

Default: numerictype([],32,0)

Methods

clone	Create corner detector with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Detect corners in input image

Examples **Detect Corners in an Input Image**

Detect corners in an input image using the FAST algorithm.

Read the input image.

```
I = im2single(imread('circuit.tif'));
```

Select FAST algorithm by Rosten & Drummond.

```
cornerDetector = vision.CornerDetector('Method','Local intensity compa
```

Find corners

```
pts = step(cornerDetector, I);
```

Create the markers. The color data range must match the data range of the input image. The color format for the marker is [red, green, blue].

```
color = [1 0 0];
drawMarkers = vision.MarkerInserter('Shape', 'Circle', 'BorderColor',
```

Convert the grayscale input image I to an RGB image J before inserting color markers.

```
J = repmat(I,[1 1 3]);
```

Display the image with markers

```
J = step(drawMarkers, J, pts);
imshow(J); title ('Corners detected in a grayscale image');
```

See Also   vision.LocalMaximaFinder | vision.EdgeDetector
| vision.MarkerInserter | detectHarrisFeatures |
detectFASTFeatures | detectMinEigenFeatures | insertMarker |
detectSURFFeatures | detectMSERFeatures | extractFeatures |
matchFeatures

vision.CornerDetector.clone

Purpose   Create corner detector with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a CornerDetector System object, *C*, with the
                   same property values as *H*. The clone method creates a new unlocked
                   object.

Purpose   Number of expected inputs to step method

Syntax   getNumInputs(H)

Description   getNumInputs(H) returns the number of expected inputs to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.CornerDetector.getNumOutputs

Purpose   Number of outputs from step method

Syntax   getNumOutputs(H)

Description   getNumOutputs(H) returns the number of outputs from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   isLocked(H)

Description   isLocked(H) returns the locked state of the corner detector.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

vision.CornerDetector.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Detect corners in input image

Syntax   ```
LOC = step(cornerDetector,I)
METRIC = step(cornerDetector,I)
[LOC,METRIC] = step(cornerDetector,I)
```

**Description**   LOC = step(cornerDetector,I) returns corners in the grayscale input image I. The step method outputs the location of detected corners, specified as an *M*-by-2 matrix of [x y] coordinates, where *M* is the number of detected corners. *M* can be less than or equal to the value specified by the MaximumCornerCount property.

METRIC = step(cornerDetector,I) returns a matrix, METRIC, with corner metric values. This applies when you set the MetricMatrixOutputPort property to true. The output, METRIC, represents corner strength. The size of the METRIC matrix is the same as that of the input image.

[LOC,METRIC] = step(cornerDetector,I) returns the locations of the corners in the *M*-by-2 matrix, LOC, and the corner metric matrix in METRIC. This applies when you set both the CornerLocationOutputPort and MetricMatrixOutputPort properties to true.

**Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

# vision.Crosscorrelator

**Purpose**      2-D cross-correlation of two input matrices

**Description**   The `Crosscorrelator` object computes 2-D cross-correlation of two
                 input matrices.

**Construction**  H = vision.Crosscorrelator returns a System object, H, that
                 performs two-dimensional cross-correlation between two inputs.

                 H = vision.Crosscorrelator(*Name,Value*) returns a 2-D cross
                 correlation System object, H, with each specified property set to the
                 specified value. You can specify additional name-value pair arguments
                 in any order as (*Name1, Value1,...,NameN,ValueN*).

**Properties**   **OutputSize**

                 Specify dimensions of output

                 This property controls the size of the output scalar, vector, or
                 matrix produced as a result of the cross-correlation between the
                 two inputs. This property can be set to one of `Full`, `Same as`
                 `first input`, `Valid`. If this property is set to `Full`, the output
                 is the full two-dimensional cross-correlation with dimensions
                 (*Ma+Mb*-1, *Na+Nb*-1). if this property is set to `same as first`
                 `input`, the output is the central part of the cross-correlation with
                 the same dimensions as the first input. if this property is set to
                 `valid`, the output is only those parts of the cross-correlation that
                 are computed without the zero-padded edges of any input. this
                 output has dimensions (*Ma-Mb*+1, *Na-Nb*+1). (*Ma*, *Na*) is the size
                 of the first input matrix and (*Mb*, *Nb*) is the size of the second
                 input matrix. The default is `Full`.

                 **Normalize**

                 Normalize output

                 Set this property to `true` to normalize the output. If you
                 set this property to `true`, the object divides the output by

                 $$\sqrt{\sum\left(I_{1p} \bullet I_{1p}\right) \times \sum\left(I_2 \bullet I_2\right)}\,, \text{ where } I_{1p} \text{ is the portion of the input}$$

matrix, $I_1$ that aligns with the input matrix, $I_2$. This property must be set to false for fixed-point inputs. The default is false.

**Fixed-Point Properties**

**RoundingMethod**

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero. The default is Floor.

**OverflowAction**

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. The default is Wrap.

**ProductDataType**

Product word and fraction lengths

Specify the product fixed-point data type as Same as first input, Custom. The default is Same as first input.

**CustomProductDataType**

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the ProductDataType property to Custom. The default is numerictype([],32,30).

**AccumulatorDataType**

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Same as product, Same as first input, Custom.The default is Same as product.

**CustomAccumulatorDataType**

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,30)`.

**OutputDataType**

Output word and fraction lengths

Specify the output fixed-point data type as `Same as first input`, `Custom`. The default is `Same as first input`.

**CustomOutputDataType**

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],16,15)`.

**Methods**

| | |
|---|---|
| clone | Create 2-D cross correlator object with same property values |
| getNumInputs | Number of expected inputs to step method |
| getNumOutputs | Number of outputs from step method |
| isLocked | Locked status for input attributes and nontunable properties |
| release | Allow property value and input characteristics changes |
| step | Compute 2-D correlation of input matrices |

**Examples**        Compute the 2-D correlation of two matrices.

```
hcorr2d = vision.Crosscorrelator;
x1 = [1 2;2 1];
x2 = [1 -1;-1 1];
y = step(hcorr2d,x1,x2);
```

Algorithms   This object implements the algorithm, inputs, and outputs described
                    on the 2-D Correlation block reference page. The object properties
                    correspond to the block parameters.

See Also   vision.Autocorrelator

vision.Crosscorrelator.clone

Purpose   Create 2-D cross correlator object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a Crosscorrelator System object C, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the
                    step method.

                    The getNumInputs method returns a positive integer that is the number
                    of expected inputs (not counting the object itself) to the step method.
                    This value will change if you alter any properties that turn inputs on or
                    off. You must call the step method with the number of input arguments
                    equal to the result of getNumInputs(*H*).

vision.Crosscorrelator.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
Crosscorrelator System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.Crosscorrelator.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose　　Compute 2-D correlation of input matrices

Syntax　　`Y = step(H,X1,X2)`

Description　　`Y = step(H,X1,X2)` computes the 2-D correlation of input matrices *X1* and *X2*.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.DCT

Purpose	Compute 2-D discrete cosine transform
Description	The DCT object computes a 2-D discrete cosine transform. The number of rows and columns of the input matrix must be a power of 2.
Construction	H = vision.DCT returns a discrete cosine transform System object, H, used to compute the two-dimensional discrete cosine transform (2-D DCT) of a real input signal.
	H = vision.DCT(*Name,Value*) returns a discrete cosine transform System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties

SineComputation

Specify how the System object computes sines and cosines as Trigonometric function, or Table lookup. This property must be set to Table lookup for fixed-point inputs.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero. The default is Floor. This property applies when you set the SineComputation to Table lookup.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. This property applies when you set the SineComputation to Table lookup. The default is Wrap.

SineTableDataType

Sine table word-length designation

Specify the sine table fixed-point data type as `Same word length as input`, or `Custom`. This property applies when you set the `SineComputation` to `Table lookup`. The default is `Same word length as input`.

CustomSineTableDataType

Sine table word length

Specify the sine table fixed-point type as a signed, unscaled `numerictype` object. This property applies when you set the `SineComputation` to `Table lookup` and you set the `SineTableDataType` property to `Custom`. The default is `numerictype(true,16)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Full precision`, `Same as first input`, or `Custom`. This property applies when you set the `SineComputation` to `Table lookup`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `SineComputation` to `Table lookup`, and the `ProductDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Full precision`, `Same as input`,`Same as product`, `Same as first input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`. The default is `Full precision`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled numerictype object. This property applies when you set the SineComputation to Table lookup, and AccumulatorDataType property to Custom. The default is numerictype(true,32,30).

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as Full precision, Same as first input, or Custom. This property applies when you set the SineComputation to Table lookup. The default is Custom.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a signed, scaled numerictype object. This property applies when you set the SineComputation to Table lookup, and the OutputDataType property to Custom. The default is numerictype(true,16,15).

Methods

clone	Create 2-D discrete cosine transform object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

release	Allow property value and input characteristics changes
step	Compute 2-D discrete cosine transform on input

Examples   Use 2-D discrete cosine transform to analyze the energy content in an image. Set the DCT coefficients lower than a threshold of 0, and then reconstruct the image using the 2-D inverse discrete cosine transform object.

```
hdct2d = vision.DCT;
I = double(imread('cameraman.tif'));
J = step(hdct2d, I);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar

hidct2d = vision.IDCT;
J(abs(J) < 10) = 0;
It = step(hidct2d, J);
figure, imshow(I, [0 255]), title('Original image')
figure, imshow(It,[0 255]), title('Reconstructed image')
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the 2-D DCT block reference page. The object properties correspond to the block parameters.

See Also   vision.IDCT

vision.DCT.clone

Purpose　　　Create 2-D discrete cosine transform object with same property values

Syntax　　　C = clone(H)

Description　　C = clone(H) creates a DCT System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.DCT.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the DCT System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

vision.DCT.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute 2-D discrete cosine transform on input

Syntax   Y = step(H,X)

Description Y = step(H,X) returns the 2-D discrete cosine transform *Y* of input *X*.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.Deinterlacer

Purpose   Remove motion artifacts by deinterlacing input video signal

Description   The `Deinterlacer` object removes motion artifacts by deinterlacing
input video signal.

Construction   `H = vision.Deinterlacer` returns a deinterlacing System object, `H`,
that removes motion artifacts from images composed of weaved top and
bottom fields of an interlaced signal.

`H = vision.Deinterlacer(Name,Value)` returns a deinterlacing
System object, `H`, with each specified property set to the specified value.
You can specify additional name-value pair arguments in any order as
(*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Method**

Method used to deinterlace input video

Specify how the object deinterlaces the input video as one of `Line
repetition` | `Linear interpolation` | `Vertical temporal
median filtering`. The default is `Line repetition`.

TransposedInput

Indicate if input data is in row-major order

Set this property to `true` if the input buffer contains data elements
from the first row first, then the second row second, and so on.

The default is `false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent`
| `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is
`Floor`. This property applies when you set the `Method` property to
`Linear Interpolation`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Wrap`. This property applies when you set the `Method` property to `Linear Interpolation`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as input` | `Custom`. The default is `Custom`. This property applies when you set the `Method` property to `Linear Interpolation`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property is applicable when the `AccumulatorDataType` property is `Custom`. This property applies when you set the `Method` property to `Linear Interpolation`.

The default is `numerictype([],12,3)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as input` | `Custom`. This property applies when you set the `Method` property to `Linear Interpolation`. The default is `Same as input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property is applicable when the `OutputDataType` property is `Custom`. This property applies when you set the `Method` property to `Linear Interpolation`.

vision.Deinterlacer

The default is `numerictype([],8,0)`.

Methods

clone	Create deinterlacer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Deinterlace input video signal

Examples

Use deinterlacing to remove motion artifacts from an input image:

```
hdint = vision.Deinterlacer;
x = imread('vipinterlace.png');
y = step(hdint, x);
imshow(x); title('Original Image');
figure, imshow(y); title('Image after deinterlacing');
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Deinterlacing block reference page. The object properties correspond to the block parameters.

See Also   `vision.CornerDetector`

Purpose   Create deinterlacer object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Deinterlacer System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.Deinterlacer.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

vision.Deinterlacer.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the Deinterlacer
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose	Deinterlace input video signal
Syntax	Y = step(H,X)
Description	Y = step(H,X) deinterlaces input *X* according to the algorithm set in the Method property.

> **Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose Bayer-pattern image conversion to true color

Description The DemosaicInterpolator object demosaics Bayer's format images. The object identifies the alignment as the sequence of R, G, and B pixels in the top-left four pixels of the image, in row-wise order.

Construction H = vision.DemosaicInterpolator returns a System object, H, that performs demosaic interpolation on an input image in Bayer format with the specified alignment.

H = vision.DemosaicInterpolator(*Name*,*Value*) returns a System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties **Method**

Interpolation algorithm

Specify the algorithm the object uses to calculate the missing color information as one of Bilinear | Gradient-corrected linear. The default is Gradient-corrected linear.

SensorAlignment

Alignment of the input image

Specify the sequence of R, G and B pixels that correspond to the 2-by-2 block of pixels in the top-left corner, of the image. It can be set to one of RGGB |GRBG | GBRG | BGGR. The default is RGGB. Specify the sequence in left-to-right, top-to-bottom order.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Saturate`.

ProductDataType

Product output word and fraction lengths

Specify the product output fixed-point data type as one of `Same as input` | `Custom`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Data type of the accumulator

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as input` | `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

Methods	clone	Create demosaic interpolator object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Perform demosaic operation on input

Examples

Demosaic a Bayer pattern encoded-image photographed by a camera with a sensor alignment of BGGR.

```
x = imread('mandi.tif');
hdemosaic = ...
    vision.DemosaicInterpolator('SensorAlignment', 'BGGR');
y = step(hdemosaic, x);

imshow(x,'InitialMagnification',20);
title('Original Image');
figure, imshow(y,'InitialMagnification',20);
title('RGB image after demosaic');
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Demosaic block reference page. The object properties correspond to the block parameters, except:

The **Output image signal** block parameter allows you to specify whether the block outputs the image as One multidimensional signal or Separate color signals. The object does not have a property that corresponds to the **Output image signal** block

parameter. The object always outputs the image as an M-by-N-by-P color video signal.

See Also   vision.GammaCorrector

Purpose   Create demosaic interpolator object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a `DemosaicInterpolator` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object.

vision.DemosaicInterpolator.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose Number of outputs from step method

Syntax N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

vision.DemosaicInterpolator.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
DemosaicInterpolator System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.DemosaicInterpolator.step

Purpose   Perform demosaic operation on input

Syntax   Y = step(H,X)

Description Y = step(H,X) performs the demosaic operation on the input *X*. The step method outputs an *M*-by-*N*-by-*P* color video signal, where *P* is the number of color planes.

> **Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Display video

Description   The `DeployableVideoPlayer` object displays video frames. This video player object supports C code generation on Windows.

Construction   `depVideoPlayer = vision.DeployableVideoPlayer` returns a video player System object, `depVideoPlayer`, for displaying video frames. Each call to the `step` method, displays the next video frame. This object only works on Windows platforms. Unlike the `vision.VideoPlayer` object, it can also generate C code.

`depVideoPlayer = vision.DeployableVideoPlayer(FRAMERATE)` returns a deployable video player System object, `depVideoPlayer`, with the display frame rate set to `FRAMERATE`. When you invoke the `step` method at a rate higher than `FRAMERATE`, the object slows down execution in order to maintain the requested display rate. This can happen when other computations in the processing loop execute at a rate faster than the display rate.

`depVideoPlayer =`
`vision.DeployableVideoPlayer(...,Name,Value)` configures the video player properties, specified as one or more `Name`,`Value` pair arguments. Unspecified properties have default values.

To display video:

1 Define and set up your deployable video player object using the constructor.

2 Call the `step` method with the deployable video player object, `depVideoPlayer`, and any optional properties. See the syntax below for using the `step` method.

`step(depVideoPlayer,I)` displays one grayscale or truecolor RGB video frame, `I`, in the video player.

`step(depVideoPlayer,Y,Cb,Cr)` displays one frame of YCbCr 4:2:2 video in the color components Y, Cb, and Cr when you set the `InputColorFormat` property to `YCbCr 4:2:2`. The number of columns

in the Cb and Cr components must be half the number of columns in the Y component.

Properties

Location

Location of bottom left corner of video window

Specify the location for the bottom left corner of the video player window as a two-element vector. The first and second elements are specified in pixels and represent the horizontal and vertical coordinates respectively. The coordinates [0 0] represent the bottom left corner of the screen.

Default: Dependent on the screen resolution, and will result in a window positioned in the center of the screen.

Name

Video window title bar caption

Specify the caption to display on the video player window title bar as any string.

Default: 'Deployable Video Player'

Size

Size of video display window

Specify the video player window size as Full-screen, True size (1:1) or Custom. When this property is set to Full-screen, use the Esc key to exit out of full-screen mode.

Default: True size (1:1)

CustomSize

Custom size for video player window

Specify the custom size of the video player window as a two-element vector. The first and second elements are specified in pixels and represent the horizontal and vertical components respectively. The video data will be resized to fit the window. This property applies when you set the Size property to Custom.

Default: [300 410]

FrameRate

Frame rate of video data

Specify the rate of video data in frames per second as a positive integer-valued scalar.

Default: 30

InputColorFormat

Color format of input signal

Specify the color format of the input data as one of RGB or YCbCr 4:2:2. The number of columns in the Cb and Cr components must be half the number of columns in Y.

Default: RGB

Methods

clone	Create deployable video player object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Send multidimensional video to computer screen

Examples

Play a Video from a File

```
videoFReader  = vision.VideoFileReader;
```

```
depVideoPlayer = vision.DeployableVideoPlayer;

while ~isDone(videoFReader)
  frame = step(videoFReader);
  step(depVideoPlayer, frame);
end
release(videoFReader);
release(depVideoPlayer);
```

See Also   vision.VideoFileWriter | vision.VideoFileReader |
                  vision.VideoPlayer

Purpose   Create deployable video player object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a `DeployableVideoPlayer` System object *C*,
                   with the same property values as *H*. The clone method creates a new
                   unlocked object.

vision.DeployableVideoPlayer.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                     the step method.

                     The getNumInputs method returns a positive integer that is the number
                     of expected inputs (not counting the object itself) to the step method.
                     This value will change if you alter any properties that turn inputs on or
                     off. You must call the step method with the number of input arguments
                     equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

vision.DeployableVideoPlayer.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
DeployableVideoPlayer System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.DeployableVideoPlayer.step

Purpose   Send multidimensional video to computer screen

Syntax   step(depVideoPlayer,I)
step(depVideoPlayer,Y,Cb,Cr)

Description   step(depVideoPlayer,I) displays one grayscale or truecolor RGB
video frame, I, in the video player.

step(depVideoPlayer,Y,Cb,Cr) displays one frame of YCbCr
4:2:2 video in the color components Y, Cb, and Cr when you set the
InputColorFormat property to YCbCr 4:2:2. The number of columns
in the Cb and Cr components must be half the number of columns in
the Y component.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

Purpose   Find object edge

Description   The `EdgeDetector` object finds edges of objects in images. For Sobel, Prewitt and Roberts algorithms, the object finds edges in an input image by approximating the gradient magnitude of the image. The object obtains the gradient as a result of convolving the image with the Sobel, Prewitt or Roberts kernel. For Canny algorithm, the object finds edges by looking for the local maxima of the gradient of the input image. The calculation derives the gradient using a Gaussian filter. This algorithm is more robust to noise and more likely to detect true weak edges.

Construction   `H = vision.EdgeDetector` returns an edge detection System object, `H`, that finds edges in an input image using Sobel, Prewitt, Roberts, or Canny algorithm.

`H = vision.EdgeDetector(`*Name*`,`*Value*`)` returns an edge detection object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Method**

Edge detection algorithm

Specify the edge detection algorithm as one of `Sobel` | `Prewitt` | `Roberts` | `Canny`. The default is `Sobel`.

BinaryImageOutputPort

Output the binary image

Set this property to `true` to output the binary image after edge detection. When you set this property to `true`, the object will output a logical matrix. The nonzero elements of this matrix correspond to the edge pixels and the zero elements correspond to the background pixels. This property applies when you set the `Method` property to `Sobel`, `Prewitt` or `Roberts`.

The default is `true`.

GradientComponentOutputPorts

Output the gradient components

Set this property to `true` to output the gradient components after edge detection. When you set this property to `true`, and the `Method` property to `Sobel` or `Prewitt`, this System object outputs the gradient components that correspond to the horizontal and vertical edge responses. When you set the `Method` property to `Roberts`, the System object outputs the gradient components that correspond to the `45` and `135` degree edge responses. Both `BinaryImageOutputPort` and `GradientComponentOutputPorts` properties cannot be `false` at the same time.

The default is `false`.

ThresholdSource

Source of threshold value

Specify how to determine threshold as one of `Auto` | `Property` | `Input port`. The default is `Auto`. This property applies when you set the `Method` property to `Canny`. This property also applies when you set the `Method` property to `Sobel`, `Prewitt` or `Roberts` and the `BinaryImageOutputPort` property to `true`.

Threshold

Threshold value

Specify the threshold value as a scalar of a MATLAB built-in numeric data type. When you set the you set the `Method` property to `Sobel`, `Prewitt` or `Roberts`, you must a value within the range of the input data. When you set the `Method` property to `Canny`, specify the threshold as a two-element vector of low and high values that define the weak and strong edges. This property is accessible when the `ThresholdSource` property is `Property`. This property is tunable.

The default is`[0.25 0.6]` when you set the `Method` property to `Canny`. Otherwise, The default is`20`.

ThresholdScaleFactor

Multiplier to adjust value of automatic threshold

Specify multiplier that is used to adjust calculation of automatic threshold as a scalar MATLAB built-in numeric data type. This property applies when you set the `Method` property to `Sobel`, `Prewitt` or `Roberts` and the `ThresholdSource` property to `Auto`. This property is tunable.

The default is `4`.

EdgeThinning

Enable performing edge thinning

Indicate whether to perform edge thinning. Choosing to perform edge thinning requires additional processing time and resources. This property applies when you set the `Method` property to `Sobel`, `Prewitt` or `Roberts` and the `BinaryImageOutputPort` property to `true`.

The default is `false`.

NonEdgePixelsPercentage

Approximate percentage of weak and nonedge pixels

Specify the approximate percentage of weak edge and nonedge image pixels as a scalar between `0` and `100`. This property applies when you set the `Method` property to `Canny` and the `ThresholdSource` to `Auto`. This property is tunable.

The default is `70`.

GaussianFilterStandardDeviation

Standard deviation of Gaussian

filter Specify the standard deviation of the Gaussian filter whose derivative is convolved with the input image. You can set this property to any positive scalar. This property applies when you set the `Method` property to `Canny`.

The default is 1.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero. The default is Custom. This property applies when you do not set the Method property to Canny.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. This property applies when you do not set the Method property to Canny.

The default is Wrap.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Same as first input | Custom. The default is Custom. This property applies when you do not set the Method property to Canny.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property iapplies when you do not set the Method property to Canny. This property applies when you set the ProductDataType property to Custom.

The default is numerictype([],32,8).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as first input` | `Same as product` | `Custom`. The default is `Same as product`. This property applies when you do not set the `Method` property to `Canny`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you do not set the `Method` property to `Canny`. This property applies when you set the `AccumulatorDataType` property to `Custom`.

The default is `numerictype([],32,8)`.

GradientDataType

Gradient word and fraction lengths

Specify the gradient components fixed-point data type as one of `Same as accumulator` | `Same as first input` | `Same as product` | `Custom`. The default is `Same as first input`. This property applies when you do not set the `Method` property to `Canny` and you set the `GradientComponentPorts` property to `true`.

CustomGradientDataType

Gradient word and fraction lengths

Specify the gradient components fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property is accessible when the `Method` property is not `Canny`. This property is applicable when the `GradientDataType` property is `Custom`.

The default is `numerictype([],16,4)`.

vision.EdgeDetector

Methods	clone	Create edge detector object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Find edges in input image

Examples

Find the edges in an image:

```
hedge = vision.EdgeDetector;
hcsc = vision.ColorSpaceConverter(...
        'Conversion', 'RGB to intensity');
hidtypeconv = ...
  vision.ImageDataTypeConverter('OutputDataType','single');
img = step(hcsc, imread('peppers.png'));
img1 = step(hidtypeconv, img);
edges = step(hedge, img1);
imshow(edges);
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Edge Detection block reference page. The object properties correspond to the block parameters.

See Also

vision.TemplateMatcher

Purpose   Create edge detector object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a EdgeDetector System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.EdgeDetector.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
                 method.

                 The getNumOutputs method returns a positive integer that is the
                 number of outputs from the step method. This value will change if you
                 alter any properties that turn outputs on or off.

vision.EdgeDetector.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the EdgeDetector
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.EdgeDetector.step

Purpose   Find edges in input image

Syntax   EDGES = step(H,IMG)
                [GV,GH] = step(H,IMG)
                [EDGES,GV,GH] = step(H,IMG)

Description   EDGES = step(H,IMG) finds the edges, EDGES , in input IMG using the
                specified algorithm when the BinaryImageOutputPort property is
                true. EDGES is a boolean matrix with non-zero elements representing
                edge pixels and zero elements representing background pixels.

                [GV,GH] = step(H,IMG) finds the two gradient components, GV and GH
                , of the input IMG, when you set the Method property to Sobel, Prewitt
                or Roberts, and you set the GradientComponentOutputPorts property
                to true and the BinaryImageOutputPort property to false. If you set
                the Method property to Sobel or Prewitt, then GV is a matrix of gradient
                values in the vertical direction and GH is a matrix of gradient values in
                the horizontal direction. If you set the Method property to Roberts, then
                GV represents the gradient component at 45 degree edge response, and
                GH represents the gradient component at 135 degree edge response.

                [EDGES,GV,GH] = step(H,IMG) finds the edges, EDGES , and the
                two gradient components, GV and GH , of the input IMG when you
                set the Method property to Sobel, Prewitt or Roberts and both
                the BinaryImageOutputPort and GradientComponentOutputPorts
                properties are true.

                Note The object performs an initialization the first time the step
                method is executed. This initialization locks nontunable properties and
                input specifications, such as dimensions, complexity, and data type
                of the input data. If you change a nontunable property or an input
                specification, the System object issues an error. To change nontunable
                properties or inputs, you must first call the release method to unlock
                the object.

Purpose   Two-dimensional discrete Fourier transform

Description   The `vision.FFT` object computes the 2D discrete Fourier transform (DFT) of a two-dimensional input matrix.

Construction   `fftObj = vision.FFT` returns a 2D FFT object, `fftObj`, that computes the fast Fourier transform of a two-dimensional input.

`fftObj = vision.FFT(Name,Value)` configures the System object properties, specified as one or more name-value pair arguments. Unspecified properties have default values.

To compute FFT:

1 Define and set up your FFT object using the constructor.

2 Call the `step` method with the input image, `I` and the FFT object, `fftObj`. See the syntax below for using the `step` method.

`J = step(fftObj,I)` computes the 2-D FFT, `J`, of an *M*-by-*N* input matrix `I`, where *M* and *N* specify the dimensions of the input. The dimensions *M* and *N* must be positive integer powers of two when any of the following are true:

   The input is a fixed-point data type
   You set the `BitReversedOutput` property to `true`.
   You set the `FFTImplementation` property to `Radix-2`.

Properties   **FFTImplementation**

   FFT implementation

   Specify the implementation used for the FFT as one of `Auto` | `Radix-2` | `FFTW`. When you set this property to `Radix-2`, the FFT length must be a power of two.

   Default: `Auto`

   BitReversedOutput

Output in bit-reversed order relative to input

Designates the order of output channel elements relative to the order of input elements. Set this property to `true` to output the frequency indices in bit-reversed order.

Default: `false`

Normalize

Divide butterfly outputs by two

Set this property to `true` if the output of the FFT should be divided by the FFT length. This option is useful when you want the output of the FFT to stay in the same amplitude range as its input. This is particularly useful when working with fixed-point data types.

Default: `false` with no scaling

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`.

Default: `Floor`

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`.

Default: `Wrap`

SineTableDataType

Sine table word and fraction lengths

Specify the sine table data type as `Same word length as input`, or `Custom`.

Default: `Same word length as input`

CustomSineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SineTableDataType` property to `Custom`.

Default: `numerictype([],16)`

ProductDataType

Product word and fraction lengths

Specify the product data type as `Full precision`, `Same as input`, or `Custom`.

Default: `Full precision`

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`.

Default: `numerictype([],32,30)`

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator data type as `Full precision`, `Same as input`, `Same as product`, or `Custom`.

Default: `Full precision`

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`.

Default: `numerictype([],32,30)`

OutputDataType

Output word and fraction lengths

Specify the output data type as `Full precision`, `Same as input`, or `Custom`.

Default: `Full precision`

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`.

Default: `numerictype([],16,15)`

Methods		
	clone	Create FFT object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Compute 2D discrete Fourier transform of input

Examples   **Use 2-D FFT to View the Frequency Components of an Image**

Create the FFT object.

```
fftObj = vision.FFT;
```

Read the image.

```
I = im2single(imread('pout.tif'));
```

Compute the FFT.

```
J = step(fftObj, I);
```

Shift zero-frequency components to the center of spectrum.

```
J_shifted = fftshift(J);
```

Display original image and visualize its FFT magnitude response.

```
figure; imshow(I); title('Input image, I');
figure; imshow(log(max(abs(J_shifted), 1e-6)),[]), colormap(jet(64));
title('Magnitude of the FFT of I');
```

References   [1] FFTW (http://www.fftw.org)

[2] Frigo, M. and S. G. Johnson, "FFTW: An Adaptive Software
Architecture for the FFT,"*Proceedings of the International Conference on
Acoustics, Speech, and Signal Processing*, Vol. 3, 1998, pp. 1381-1384.

See Also   vision.IFFT | vision.DCT | vision.IDCT | fft2

Purpose   Create FFT object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an FFT System object C, with the same property
                   values as H. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

vision.FFT.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the FFT System
object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.FFT.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute 2D discrete Fourier transform of input

Syntax   Y = step(H,X)

Description Y = step(H,X) computes the 2D discrete Fourier transform (DFT), *Y* , of an *M*-by-*N* input matrix *X* , where the values of *M* and *N* are integer powers of two.

vision.ForegroundDetector

Purpose   Detects foreground using Gaussian mixture models

Description   The `ForegroundDetector` object compares a color or grayscale video frame to a background model to determine whether individual pixels are part of the background or the foreground. It then computes a foreground mask.

Construction   `detector = vision.ForegroundDetector` returns a foreground detector System object, `H`, that computes a foreground mask. Given a series of either grayscale or color video frames, the object computes the foreground mask using Gaussian mixture models (GMM).

`detector = vision.ForegroundDetector(Name,Value)` returns a foreground detector System object, `detector`, with each specified property name set to the specified value. `Name` can also be a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name-value pair arguments in any order as `Name1,Value1, ,NameN,ValueN`.

Properties   **AdaptLearningRate**

Adapt learning rate

Enables the object to adapt the learning rate during the period specified by the `NumTrainingFrames` property. When you set this property to `true`, the object sets the `LearningRate` property to 1/(current frame number). When you set this property to `false`, the `LearningRate` property must be set at each time step.

Default: `true`

NumTrainingFrames

Number of initial video frames for training background model

Set this property to the number of training frames at the start of the video sequence.

When you set the `AdaptLearningRate` to `false`, this property will not be available.

Default: 150

LearningRate

Learning rate for parameter updates

Specify the learning rate to adapt model parameters. This property controls how quickly the model adapts to changing conditions. Set this property appropriately to ensure algorithm stability.

When you set AdaptLearningRate to true, the LearningRate property takes effect only after the training period specified by NumTrainingFrames is over.

When you set the AdaptLearningRate to false, this property will not be available. This property is tunable.

Default: 0.005

MinimumBackgroundRatio

Threshold to determine background model

Set this property to represent the minimum of the apriori probabilities for pixels to be considered background values. Multimodal backgrounds can not be handled, if this value is too small.

Default: 0.7

NumGaussians

Number of Gaussian modes in the mixture model

Specify the number of Gaussian modes in the mixture model. Set this property to a positive integer. Typically this value is 3, 4 or 5. Set this value to 3 or greater to be able to model multiple background modes.

Default: 5

InitialVariance

Variance when initializing a new Gaussian mode

Initial variance to initialize all distributions that compose the foreground-background mixture model. For a `uint8` input a typical value is 30^2. For a floating point input this typical value is scaled to $\left(30\big/255\right)^2$. This property applies to all color channels for color inputs.

Default: `(30/255)^2;`

Methods		
	clone	Create foreground detector with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	reset	Reset the GMM model to its initial state
	step	Detect foreground using Gaussian mixture models

Examples

Track Cars:

```
videoSource = vision.VideoFileReader('viptraffic.avi','ImageColorSpace','
detector = vision.ForegroundDetector(...
      'NumTrainingFrames', 5, ... % 5 because of short video
      'InitialVariance', 30*30); % initial standard deviation of 30
blob = vision.BlobAnalysis(...
      'CentroidOutputPort', false, 'AreaOutputPort', false, ...
```

```
        'BoundingBoxOutputPort', true, ...
        'MinimumBlobAreaSource', 'Property', 'MinimumBlobArea', 250);
shapeInserter = vision.ShapeInserter('BorderColor','White');

videoPlayer = vision.VideoPlayer();
while ~isDone(videoSource)
    frame  = step(videoSource);
    fgMask = step(detector, frame);
    bbox   = step(blob, fgMask);
    out    = step(shapeInserter, frame, bbox); % draw bounding boxes
    step(videoPlayer, out); % view results in the video player
end
release(videoPlayer);
release(videoSource);
```

References

[1] P. Kaewtrakulpong, R. Bowden, *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection*, In Proc. *2nd European Workshop on Advanced Video Based Surveillance Systems*, AVBS01, VIDEO BASED SURVEILLANCE SYSTEMS: Computer Vision and Distributed Processing (September 2001)

[2] Stauffer, C. and Grimson, W.E.L,*Adaptive Background Mixture Models for Real-Time Tracking*, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, Vol. 2 (06 August 1999), pp. 2246-252 Vol. 2.

vision.ForegroundDetector.clone

Purpose   Create foreground detector with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a ForegroundDetector System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                  the step method.

                  The getNumInputs method returns a positive integer that is the number
                  of expected inputs (not counting the object itself) to the step method.
                  This value will change if you alter any properties that turn inputs on or
                  off. You must call the step method with the number of input arguments
                  equal to the result of getNumInputs(*H*).

vision.ForegroundDetector.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
ForegroundDetector System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.ForegroundDetector.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
                   or hardware connections) and allows all properties and input
                   characteristics to be changed.

> **Note** You cannot use the release method on System objects in code
> generated from MATLAB.

Purpose   Reset the GMM model to its initial state

Syntax   reset(H)

Description reset(H) resets the GMM model to its initial state for the
                 ForegroundDetector object H.

vision.ForegroundDetector.step

Purpose   Detect foreground using Gaussian mixture models

Syntax   foregroundMask = step(H,I)
                     foregroundMask = step(H,I,learningRate)

Description   foregroundMask = step(H,I) computes the foreground mask for
                     input image I, and returns a logical mask. When the object returns
                     foregroundMask set to 1, it represents foreground pixels. Image I can
                     be grayscale or color. When you set the AdaptLearningRate property to
                     true (default), the object permits this form of the step function call.

                     foregroundMask = step(H,I,learningRate) computes the foreground
                     mask for input image I using the LearningRate you provide. When
                     you set the AdaptLearningRate property to false, the object permits
                     this form of the step function call.

                     Note The object performs an initialization the first time the step
                     method is executed. This initialization locks nontunable properties and
                     input specifications, such as dimensions, complexity, and data type
                     of the input data. If you change a nontunable property or an input
                     specification, the System object issues an error. To change nontunable
                     properties or inputs, you must first call the release method to unlock
                     the object.

Purpose	Apply or remove gamma correction from images or video streams
Description	The GammaCorrector object applies gamma correction to input images or video streams.
Construction	H = vision.GammaCorrector returns a System object, HGAMMACORR. This object applies or removes gamma correction from images or video streams.

H = vision.GammaCorrector(*Name*,*Value*) returns a gamma corrector object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

H = vision.GammaCorrector(GAMMA,*Name*,*Value*) returns a gamma corrector object, H, with the Gamma property set to GAMMA and other specified properties set to the specified values.

Properties

Correction

Specify gamma correction or linearization

Specify the object's operation as one of Gamma | De-gamma. The default is Gamma.

Gamma

Gamma value of output or input

When you set the Correction property to Gamma, this property gives the desired gamma value of the output video stream. When you set the Correction property to De-gamma, this property indicates the gamma value of the input video stream. You must set this property to a numeric scalar value greater than or equal to 1.

The default is 2.2.

LinearSegment

Enable gamma curve to have linear portion near origin

vision.GammaCorrector

Set this property to `true` to make the gamma curve have a linear portion near the origin.

The default is `true`.

BreakPoint

I-axis value of the end of gamma correction linear segment

Specify the I-axis value of the end of the gamma correction linear segment as a scalar numeric value between `0` and `1`. This property applies when you set the `LinearSegment` property to `true`.

The default is `0.018`.

Methods

clone	Create gamma corrector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Apply or remove gamma correction from input

Examples

Improve image contrast.

```
hgamma = ...
  vision.GammaCorrector(2.0,'Correction','De-gamma');
x = imread('pears.png');
y = step(hgamma, x);

imshow(x); title('Original Image');
```

```
figure, imshow(y);
title('Enhanced Image after De-gamma Correction');
```

Algorithms   This object implements the algorithm, inputs, and outputs described
on the Gamma Correction block reference page. The object properties
correspond to the block parameters.

See Also   vision.HistogramEqualizer | vision.ContrastAdjuster

Purpose   Create gamma corrector object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a GammaCorrector System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.GammaCorrector.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the GammaCorrector
                  System object.

                  The isLocked method returns a logical value that indicates whether
                  input attributes and nontunable properties for the object are locked. The
                  object performs an internal initialization the first time the step method
                  is executed. This initialization locks nontunable properties and input
                  specifications, such as dimensions, complexity, and data type of the
                  input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Apply or remove gamma correction from input

Syntax   Y = step(H,X)

Description   Y = step(H,X) applies or removes gamma correction from input *X* and
                   returns the gamma corrected or linearized output *Y*.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

---

vision.GeometricRotator

Purpose	Rotate image by specified angle

Description   The GeometricRotator object rotates an image by a specified angle.

Construction   H = vision.GeometricRotator returns a geometric rotator System

object, H, that rotates an image by $\dfrac{\pi}{6}$ radians.

H = vision.GeometricRotator(*Name*,*Value*) returns a geometric rotator object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **OutputSize**

Output size as full or same as input image size

Specify the size of output image as one of Expanded to fit rotated input image | Same as input image. The default is Expanded to fit rotated input image. When you set this property to Expanded to fit rotated input image, the object outputs a matrix that contains all the rotated image values. When you set this property to Same as input image, the object outputs a matrix that contains the middle part of the rotated image.

AngleSource

Source of angle

Specify how to specify the rotation angle as one of Property | Input port. The default is Property.

Angle

Rotation angle value (radians)

Set this property to a real, scalar value for the rotation angle (radians). The default is pi/6. This property applies when you set the AngleSource property to Property.

MaximumAngle

Maximum angle by which to rotate image

Specify the maximum angle by which to rotate the input image as a numeric scalar value greater than `0`. The default is `pi`. This property applies when you set the `AngleSource` property to `Input port`.

RotatedImageLocation

How the image is rotated

Specify how the image is rotated as one of `Top-left corner` | `Center`. The default is `Center`. When you set this property to `Center`, the object rotates the image about its center point. When you set this property to `Top-left corner`, the object rotates the image so that two corners of the input image are always in contact with the top and left side of the output image. This property applies when you set the `OutputSize` property to `Expanded to fit rotated input image`, and, the `AngleSource` property to `Input port`.

SineComputation

How to calculate the rotation

Specify how to calculate the rotation as one of `Trigonometric function` | `Table lookup`. The default is `Table lookup`. When you set this property to `Trigonometric function`, the object computes the sine and cosine values it needs to calculate the rotation of the input image. When you set this property to `Table lookup`, the object computes the trigonometric values it needs to calculate the rotation of the input image, and stores them in a table. The object uses the table, each time it calls the `step` method. In this case, the object requires extra memory.

BackgroundFillValue

Value of pixels outside image

Specify the value of pixels that are outside the image as a numeric scalar value or a numeric vector of same length as the third

dimension of input image. The default is 0. This property is tunable.

InterpolationMethod

Interpolation method used to rotate image

Specify the interpolation method used to rotate the image as one of `Nearest neighbor` | `Bilinear` | `Bicubic`. The default is `Bilinear`. When you set this property to `Nearest neighbor`, the object uses the value of one nearby pixel for the new pixel value. When you set this property to `Bilinear`, the object sets the new pixel value as the weighted average of the four nearest pixel values. When you set this property to `Bicubic`, the object sets the new pixel value as the weighted average of the sixteen nearest pixel values.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. he default is `Nearest`. This property applies when you set the `SineComputation` property to `Table lookup`. T

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Saturate`. This property applies when you set the `SineComputation` property to `Table lookup`.

AngleDataType

Angle word and fraction lengths

Specify the angle fixed-point data type as one of `Same word length as input` | `Custom`. The default is `Same word length as`

input. This property applies when you set the `SineComputation` property to `Table lookup`, and the `AngleSource` property to `Property`.

CustomAngleDataType

Angle word and fraction lengths

Specify the angle fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, the `AngleSource` property is `Property` and the `AngleDataType` property to `Custom`. The default is `numerictype([],32,10)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as first input` | `Custom`. The default is `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as first input` | `Custom`. The default is `Same as product`. This property applies when you set the `SineComputation` property to `Table lookup`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as first input | Custom`. The default is `Same as first input`. This property applies when you set the `SineComputation` property to `Table lookup`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `OutputDataType` property to `Custom`. The default is `numerictype([],32,10)`.

Methods

clone	Create geometric rotator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Return rotated image

Examples   Rotate an image 30 degrees ($\pi/6$ radians)

```
img = im2double(imread('peppers.png'));
hrotate = vision.GeometricRotator;
hrotate.Angle = pi / 6;

% Rotate img by pi/6
rotimg = step(hrotate,img);
imshow(rotimg);
```

Rotate an image by multiple angles, specifying each rotation angle as an input. The angle can be changed during simulation.

```
hrotate = vision.GeometricRotator;
hrotate.AngleSource = 'Input port';
img = im2double(imread('onion.png'));
figure;
% Rotate img by multiple angles
for i = 1: 4
  rotimg = step(hrotate,img,i*pi/16);
  subplot(1,4,i);
  imshow(rotimg);
end
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Rotate block reference page. The object properties correspond to the block parameters.

See Also   vision.GeometricTranslator | vision.GeometricScaler

vision.GeometricRotator.clone

Purpose   Create geometric rotator object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a GeometricRotator System object C, with the
                   same property values as H. The clone method creates a new unlocked
                   object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.GeometricRotator.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                     GeometricRotator System object.

                     The isLocked method returns a logical value that indicates whether
                     input attributes and nontunable properties for the object are locked. The
                     object performs an internal initialization the first time the step method
                     is executed. This initialization locks nontunable properties and input
                     specifications, such as dimensions, complexity, and data type of the
                     input data. After locking, the isLocked method returns a true value.

vision.GeometricRotator.release

Purpose Allow property value and input characteristics changes

Syntax release(H)

Description release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Return rotated image

Syntax   Y = step(H,IMG)
                Y = step(H,IMG,ANGLE)

Description Y = step(H,IMG) returns a rotated image *Y* , with the rotation angle
                specified by the Angle property.

                Y = step(H,IMG,ANGLE) uses input *ANGLE* as the angle to rotate the
                input *IMG* when you set the AngleSource property to Input port.

                Note The object performs an initialization the first time the step
                method is executed. This initialization locks nontunable properties and
                input specifications, such as dimensions, complexity, and data type
                of the input data. If you change a nontunable property or an input
                specification, the System object issues an error. To change nontunable
                properties or inputs, you must first call the release method to unlock
                the object.

                ---

vision.GeometricScaler

Purpose   Enlarge or shrink image size

Description   The `GeometricScaler` object enlarges or shrinks image sizes.

`J= step(H,I)` returns a resized image, `J`, of input image `I`.

`J= step(H,I,ROI)` resizes a particular region of the image `I` defined by the *ROI* specified as [x y width height], where [x y] represents the upper left corner of the ROI. This option applies when you set the `SizeMethod` property to `Number of output rows and columns`, the `Antialiasing` property to `false`, the `InterpolationMethod` property to `Bilinear`, `Bicubic` or `Nearest neighbor`, and the `ROIProcessing` property to `true`.

`[J,FLAG] = step(H,I,ROI)` also returns *FLAG* which indicates whether the given region of interest is within the image bounds. This applies when you set the `SizeMethod` property to `Number of output rows and columns`, the `Antialiasing` property to `false`, the `InterpolationMethod` property to `Bilinear`, `Bicubic` or `Nearest neighbor` and, the `ROIProcessing` and the `ROIValidityOutputPort` properties to `true`.

Construction   `H = vision.GeometricScaler` returns a System object, `H`, that changes the size of an image or a region of interest within an image.

`H = vision.GeometricScaler(`*Name*`,`*Value*`)` returns a geometric scaler object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **SizeMethod**

Aspects of image to resize

Specify which aspects of the input image to resize as one of `Output size as a percentage of input size` | `Number of output columns and preserve aspect ratio` | `Number of output rows and preserve aspect ratio` | `Number of output rows`

and columns. The default is `Output size as a percentage of input size`.

ResizeFactor

Percentage by which to resize rows and columns

Set this property to a scalar percentage or a two-element vector. The default is `[200 150]`. When you set this property to a scalar percentage, the object applies this value to both rows and columns. When you set this property to a two-element vector, the object uses the first element as the percentage to resize the rows, and the second element as the percentage to resize the columns. This property applies when you set the `SizeMethod` property to `Output size as a percentage of input size`.

NumOutputColumns

Number of columns in output image

Specify the number of columns of the output image as a positive, integer scalar value. The default is `25`. This property applies when you set the `SizeMethod` property to `Number of output columns and preserve aspect ratio`.

NumOutputRows

Number of rows in output image

Specify the number of rows of the output image as a positive integer scalar value. The default is `25`. This property applies when you set the `SizeMethod` property to `Number of output rows and preserve aspect ratio`.

Size

Dimensions of output image

Set this property to a two-element vector. The default is `[25 35]`. The object uses the first element for the number of rows in the output image, and the second element for the number of columns. This property applies when you set the `SizeMethod` property to `Number of output rows and columns`.

vision.GeometricScaler

InterpolationMethod

Interpolation method used to resize the image

Specify the interpolation method to resize the image as one of Nearest neighbor | Bilinear | Bicubic | Lanczos2 | Lanczos3. The default is Bilinear. When you set this property to Nearest neighbor, the object uses one nearby pixel to interpolate the pixel value. When you set this property to Bilinear, the object uses four nearby pixels to interpolate the pixel value. When you set this property to Bicubic or Lanczos2, the object uses sixteen nearby pixels to interpolate the pixel value. When you set this property to Lanczos3, the object uses thirty six surrounding pixels to interpolate the pixel value.

Antialiasing

Enable low-pass filtering when shrinking image

Set this property to true to perform low-pass filtering on the input image before shrinking it. This prevents aliasing when the ResizeFactor property value is between 0 and 100 percent.

ROIProcessing

Enable region-of-interest processing

Indicate whether to resize a particular region of each input image. The default isfalse. This property applies when you set the SizeMethod property to Number of output rows and columns, the InterpolationMethod parameter to Nearest neighbor, Bilinear, or Bicubic, and the Antialiasing property to false.

ROIValidityOutputPort

Enable indication that ROI is outside input image

Indicate whether to return the validity of the specified ROI being completely inside image. The default isfalse. This property applies when you set the ROIProcessing property to true.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Nearest`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Saturate`.

InterpolationWeightsDataType

Interpolation weights word and fraction lengths

Specify the interpolation weights fixed-point data type as one of `Same word length as input` | `Custom`. The default is `Same word length as input`. This property applies when you set:

- The `InterpolationMethod` property to `Bicubic`, `Lanczos2`, or `Lanczos3`.

- The `SizeMethod` property to any value other than `Output size as a percentage of input size`.

- When you set the combination of:

    The `SizeMethod` property to `Output size as a percentage of input size`.
    The `InterpolationMethod` property to `Bilinear` or `Nearest neighbor`.
    Any of the elements of the `ResizeFactor` less than `100`, implying shrinking of the input image.
    The `Antialiasing` property to `true`.

CustomInterpolationWeightsDataType

Interpolation weights word and

fraction lengths Specify the interpolation weights fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `InterpolationWeightsDataType` property to `Custom`. The default is `numerictype([],32)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as input` | `Custom`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as input` | `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as input` | `Custom`. The default is `Same as input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],32,10)`.

Methods

clone	Create geometric scaler object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Resize image

Examples   Enlarge an image. Display the original and the enlarged images.

```
I=imread('cameraman.tif');
hgs=vision.GeometricScaler;
hgs.SizeMethod = ...
    'Output size as a percentage of input size';
hgs.InterpolationMethod='Bilinear';

J = step(hgs,I);
imshow(I); title('Original Image');
figure,imshow(J);title('Resized Image');
```

vision.GeometricScaler

Algorithms   This object implements the algorithm, inputs, and outputs described on the Resize block reference page. The object properties correspond to the block parameters.

See Also   `vision.Pyramid` | `vision.GeometricRotator` | `vision.GeometricTranslator`

Purpose   Create geometric scaler object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a `GeometricScaler` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object.

vision.GeometricScaler.getNumInputs

Purpose	Number of expected inputs to step method
Syntax	N = getNumInputs(H)
Description	N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.
	The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   `N = getNumOutputs(H)`

Description   `N = getNumOutputs(H)` returns the number of outputs, *N* from the `step` method.

The `getNumOutputs` method returns a positive integer that is the number of outputs from the `step` method. This value will change if you alter any properties that turn outputs on or off.

vision.GeometricScaler.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
GeometricScaler System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.GeometricScaler.step

Purpose   Resize image

Syntax   J = step(H,I)
                 J = step(H,X,ROI)
                 [J,FLAG] = step(H,I,ROI)

Description J = step(H,I) returns a resized image, *J*, of input image *I*.

J = step(H,X,ROI) resizes a particular region of the image *I* defined by the *ROI* input. This applies when you set the SizeMethod property to Number of output rows and columns, the Antialiasing property to false, the InterpolationMethod property to Bilinear, Bicubic or Nearest neighbor, and the ROIProcessing property to true.

[J,FLAG] = step(H,I,ROI) also returns *FLAG* which indicates whether the given region of interest is within the image bounds. This applies when you set the SizeMethod property to Number of output rows and columns, the Antialiasing property to false, the InterpolationMethod property to Bilinear, Bicubic or Nearest neighbor and, the ROIProcessing and the ROIValidityOutputPort properties to true.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Shift rows or columns of image by linearly varying offset

Description   The `GeometricShearer` System object shifts the rows or columns of an image by a gradually increasing distance left or right or up or down.

Construction `H = vision.GeometricShearer` creates a System object, *H*, that shifts the rows or columns of an image by gradually increasing distance left or right or up or down.

`H = vision.GeometricShearer(`*Name*`,`*Value*`, ...` `'`*NameN*`',`*ValueN*`)` creates a geometric shear object, *H*, with each specified property set to the specified value.

Properties   **Direction**

Direction to apply offset

Specify the direction of linearly increasing the offset as one of `Horizontal` | `Vertical`. The default is `Horizontal`. Set this property to `Horizontal` to linearly increase the offset of the rows, or `Vertical` to linearly increase the offset of the columns.

OutputSize

Output size

Specify the size of the output image as one of `Full` | `Same as input image`. The default is `Full`. If you set this property to `Full`, the object outputs a matrix that contains the sheared image values. If you set this property to `Same as input image`, the object outputs a matrix that is the same size as the input image and contains a portion of the sheared image.

ValuesSource

Source of shear values

Specify the source of shear values as one of `Property` | `Input port`. The default is `Property`.

Values

Shear values in pixels

Specify the shear values as a two-element vector that represents the number of pixels by which to shift the first and last rows or columns of the input. The default is [0 3]. This property applies when you set the ValuesSource property to Property.

MaximumValue

Maximum number of pixels by which to shear image

Specify the maximum number of pixels by which to shear the image as a real, numeric scalar. The default is 20. This property applies when you set the ValuesSource property to Input port.

BackgroundFillValue

Value of pixels outside image

Specify the value of pixels that are outside the image as a numeric scalar or a numeric vector that has the same length as the third dimension of the input image. The default is 0. This property is tunable.

InterpolationMethod

Interpolation method used to shear image

Specify the interpolation method used to shear the image as on of Nearest neighbor | Bilinear | Bicubic. The default is Bilinear. When you set this property to Nearest neighbor, the object uses the value of one nearby pixel for the new pixel value. When you set this property to Bilinear, the object sets the new pixel value as the weighted average of the two nearest pixel values. When you set this property to Bicubic, the object sets the new pixel value as the weighted average of the four nearest pixel values.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Nearest`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Saturate`.

ValuesDataType

Shear values word and fraction lengths

Specify the shear values fixed-point data type as one of `Same word length as input` | `Custom`. The default is `Same word length as input`. This property applies when you set the `ValuesSource` property to `Property`.

CustomValuesDataType

Shear values word and fraction lengths

Specify the shear values fixed-point type as an auto-signed `numerictype` object. This property applies when you set the `ValuesSource` property to `Property` and the `ValuesDataType` property to `Custom`. The default is `numerictype([],32,10)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as first input` | `Custom`. The default is `Custom`. This property applies when you set the `InterpolationMethod` property to either `Bilinear` or `Bicubic`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as an auto-signed scaled `numerictype` object. This property applies when you set the `InterpolationMethod` property to either `Bilinear` or `Bicubic`, and the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as first input` | `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as an auto-signed scaled `numerictype` object. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Same as first input` or `Custom`. The default is `Same as first input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as an auto-signed scaled `numerictype` object. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],32,10)`.

Methods	clone	Create a geometric shear object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	step	Shift rows or columns of image by linearly varying offset

Examples   Apply a horizontal shear to an image.

```
hshear = vision.GeometricShearer('Values',[O 20]);
img = im2single(checkerboard);
outimg = step(hshear,img);
subplot(2,1,1), imshow(img);
title('Original image');
subplot(2,1,2), imshow(outimg);
title('Output image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Shear block reference page. The object properties correspond to the block parameters.

See Also   vision.GeometricTransformer | vision.GeometricScaler

vision.GeometricShearer.clone

Purpose   Create a geometric shear object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a GeometricShearer object C, with the same
                       property values as H. The clone method creates a new unlocked object
                       with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(*H*)

Description   N = getNumInputs(*H*) returns the number of expected inputs, *N*, to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.GeometricShearer.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N*, from the
step method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, *TF* of the
                   GeometricShearer System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

vision.GeometricShearer.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Shift rows or columns of image by linearly varying offset

Syntax   Y = step(H,IMG)
                     Y = step(H,IMG,S)

Description   Y = step(H,IMG) shifts the input, *IMG*, and returns the shifted image,
                    Y, with the shear values specified by the Values property.

                    Y = step(H,IMG,S) uses the two-element vector, *S*, as the number of
                    pixels by which to shift the first and last rows or columns of *IMG*. This
                    applies when you set the ValuesSource property to Input port.

                    Note The object performs an initialization the first time the step
                    method is executed. This initialization locks nontunable properties and
                    input specifications, such as dimensions, complexity, and data type
                    of the input data. If you change a nontunable property or an input
                    specification, the System object issues an error. To change nontunable
                    properties or inputs, you must first call the release method to unlock
                    the object.

                    ---

vision.GeometricTransformer

Purpose   Apply projective or affine transformation to image

Description   The `GeometricTransformer` object applies a projective or affine transformation to an image.

Construction   `H = vision.GeometricTransformer` returns a geometric transformation System object, `H`. This object applies a projective or affine transformation to an image.

`H = vision.GeometricTransformer(`*Name*`,`*Value*`)` returns a geometric transformation object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **TransformMatrixSource**

Method to specify transformation matrix

Specify as one of `Property` | `Input port`.

Default: `Input port`.

TransformMatrix

Transformation matrix

Specify the applied transformation matrix as a 3-by-2 or *Q*-by-6 affine transformation matrix, or a 3-by-3 or a *Q*-by-9 projective transformation matrix. *Q* is the number of transformations. This property applies when you set the `TransformMatrixSource` property to `Property`.

Default: `[1 0 0; 0 1 0; 0 0 1]`

InterpolationMethod

Interpolation method

Specify as one of `Nearest neighbor` | `Bilinear` | `Bicubic` for calculating the output pixel value.

Default: `Bilinear`

BackgroundFillValue

Background fill value

Specify the value of the pixels that are outside of the input image. The value can be either scalar or a *P*-element vector, where *P* is the number of color planes.

Default: `0`

OutputImagePositionSource

Method to specify output image location and size

Specify the value of this property as one of `Auto` | `Property`. If this property is set to `Auto`, the output image location and size are the same values as the input image.

Default: `Auto`

OutputImagePosition

Output image position vector

Specify the location and size of output image, as a four-element double vector in pixels, of the form, `[x y width height]`. This property applies when you set the `OutputImagePositionSource` property to `Property`.

Default: `[1 1 512 512]`

ROIInputPort

Enable the region of interest input port

Set this property to `true` to enable the input of the region of interest. When set to `false`, the whole input image is processed.

Default: `false`

ROIShape

Region of interest shape

Specify ROIShape as one of `Rectangle ROI` | `Polygon ROI`. This property applies when you set the `ROIInputPort` property to `true`.

Default:`Rectangle ROI`

ROIValidityOutputPort

Enable output of ROI flag

Set this property to `true` to enable the output of an ROI flag indicating when any part of the ROI is outside the input image. This property applies when you set the `ROIInputPort` property to `true`.

Default:`false`

ProjectiveTransformMethod

Projective transformation method

Method to compute the projective transformation. Specify as one of `Compute exact values` | `Use quadratic approximation`.

Default: `Compute exact values`

ErrorTolerance

Error tolerance (in pixels)

Specify the maximum error tolerance in pixels for the projective transformation. This property applies when you set the `ProjectiveTransformMethod` property to `Use quadratic approximation`.

Default: `1`

ClippingStatusOutputPort

Enable clipping status flag output

Set this property to `true` to enable the output of a flag indicating if any part of the output image is outside the input image.

Default: `false`

Methods	clone	Create geometric transformer object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Apply geometric transform to input image

Examples   Apply a horizontal shear to an intensity image.

```
htrans1 = vision.GeometricTransformer(...
          'TransformMatrixSource', 'Property', ...
          'TransformMatrix',[1 0 0; .5 1 0; 0 0 1],...
          'OutputImagePositionSource', 'Property',...
          'OutputImagePosition', [0 0 400 750]);
img1 = im2single(rgb2gray(imread('peppers.png')));
transimg1 = step(htrans1,img1);
imshow(transimg1);
```

Apply a transform with multiple polygon ROI's.

```
htrans2 = vision.GeometricTransformer;
img2 = checker_board(20,10);

tformMat = [     1        0       30        0        1       -30; ...
            1.0204  -0.4082       70        0   0.4082        30; ...
            0.4082        0  89.1836  -0.4082        1   10.8164];
```

```
polyROI = [   1   101    99   101    99   199     1   199; ...
              1     1    99     1    99    99     1    99; ...
            101   101   199   101   199   199   101   199];

htrans2.BackgroundFillValue = [0.5 0.5 0.75];
htrans2.ROIInputPort = true;
htrans2.ROIShape = 'Polygon ROI';
transimg2 = step(htrans2,img2,tformMat,polyROI);
figure; imshow(img2);
figure; imshow(transimg2);
```

Algorithms   This object implements the algorithm, inputs, and outputs described on
the Apply Geometric Transformation block reference page. The object
properties correspond to the block parameters.

See Also   vision.GeometricTransformEstimator

Purpose   Create geometric transformer object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a GeometricTransformer System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.GeometricTransformer.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
                the step method.

                The getNumInputs method returns a positive integer that is the number
                of expected inputs (not counting the object itself) to the step method.
                This value will change if you alter any properties that turn inputs on or
                off. You must call the step method with the number of input arguments
                equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.GeometricTransformer.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                   GeometricTransformer System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

Purpose　　　　Allow property value and input characteristics changes

Syntax　　　　`release(H)`

Description　　`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note　You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.GeometricTransformer.step

Purpose   Apply geometric transform to input image

Syntax   Y = step(H,X,TFORM)
                 Y = step(H,X)
                 Y = step(H,X,ROI)
                 [Y, ROIFLAG]=(H,X,...)
                 [Y,CLIPFLAG]=(H,X,...)
                 [Y,ROIFLAG,CLIPFLAG] = step(H,X,TFORM,ROI)

Description Y = step(H,X,TFORM) outputs the transformed image, *Y*, of the input
                image, *X* . *X* is either an *M*-by-*N* or an *M*-by-*N*-by-*P* matrix, where
                M is the number of rows, *N* is the number of columns, and *P* is the
                number of color planes in the image.*TFORM* is the applied transformation
                matrix. *TFORM* can be a 2-by-3 or 6-by-*Q* affine transformation matrix,
                or a 3-by-3 or 9-by-*Q* projective transformation matrix, where *Q* is the
                number of transformations.

                Y = step(H,X) outputs the transformed image, *Y* , of the input image,
                X. This applies when you set the `TransformMatrixSource` property
                to `Property`.

                Y = step(H,X,ROI) outputs the transformed image of the input image
                within the region of interest, *ROI* . When specifying a rectangular region
                of interest, *ROI* must be a 4-element vector or a 4-by-*R* matrix. When
                specifying a polygonal region of interest, *ROI* must be a 2*L*-element
                vector or a 2*L*-by-*R* matrix. *R* is the number of regions of interest, and *L*
                is the number of vertices in a polygon region of interest.

                [Y, ROIFLAG]=(H,X,...) returns a boolean flag, *ROIFLAG*, indicating
                if any part of the region of interest is outside the input image. This
                applies when you set the `ROIValidityOutputPort` property to `true`.

                [Y,CLIPFLAG]=(H,X,...) returns a boolean flag, *CLIPFLAG*, indicating
                if any transformed pixels were clipped. This applies when you set the
                `ClippingStatusOutputPort` property to `true`.

                [Y,ROIFLAG,CLIPFLAG] = step(H,X,TFORM,ROI) outputs the
                transformed image, *Y*, of the input image, *X* , within the region of
                interest, *ROI*, using the transformation matrix, *TFORM*. *ROIFLAG*,

indicates if any part of the region of interest is outside the input image, and *CLIPFLAG*, indicates if any transformed pixels were clipped. This provides all operations simultaneously with all possible inputs. Properties must be set appropriately.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.GeometricTransformEstimator

Purpose   Estimate geometric transformation from matching point pairs

Description The `GeometricTransformEstimator` object estimates geometric transformation from matching point pairs and returns the transform in a matrix. Use this object to compute projective, affine, or nonreflective similarity transformations with robust statistical methods, such as, RANSAC and Least Median of Squares.

Use the step syntax below with input points, `MATCHED_POINTS1` and `MATCHED_POINTS2`, the object, `H`, and any optional properties.

`TFORM = step(H,MATCHED_POINTS1, MATCHED_POINTS2)` calculates the transformation matrix, `TFORM`. The input arrays, `MATCHED_POINTS1` and `MATCHED_POINTS2` specify the locations of matching points in two images. The points in the arrays are stored as a set of $[x_1 \, y_1; \, x_2 \, y_2; \, ...; \, x_N \, y_N]$ coordinates, where N is the number of points. When you set the `Transform` property to `Projective`, the `step` method outputs `TFORM` as a 3-by-3 matrix. Otherwise, the `step` method outputs `TFORM` as a 3-by-2 matrix.

`[TFORM,INLIER_INDEX] = step(H, ...)` additionally outputs the logical vector, `INLIER_INDEX`, indicating which points are the inliers.

Construction `H = vision.GeometricTransformEstimator` returns a geometric transform estimation System object, `H`. This object finds the transformation matrix that maps the largest number of points between two images.

`H = vision.GeometricTransformEstimator(Name,Value)` returns a geometric transform estimation object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Transform**

Transformation type

Specify transformation type as one of `Nonreflective similarity` | `Affine` | `Projective`.

Default: `Affine`

ExcludeOutliers

Whether to exclude outliers from input points

Set this property to `true` to find and exclude outliers from the input points and use only the inlier points to calculate the transformation matrix. When you set this property to `false`, all input points are used to calculate the transformation matrix.

Default: `true`

Method

Method to find outliers

Specify the method to find outliers as one of `Random Sample Consensus (RANSAC)` | `Least Median of Squares`.

Default: `Random Sample Consensus (RANSAC)`.

AlgebraicDistanceThreshold

Algebraic distance threshold for determining inliers

Specify a scalar threshold value for determining inliers as a positive, scalar value. The threshold controls the upper limit used to find the algebraic distance for the RANSAC method. This property applies when you set the `Transform` property to `Projective` and the `Method` property to `Random Sample Consensus (RANSAC)`. This property is tunable.

Default: `2.5`.

PixelDistanceThreshold

Distance threshold for determining inliers in pixels

Specify the upper limit of the algebraic distance that a point can differ from the projection location of its associating point. Set this property as a positive, scalar value. This property applies when you set the `Transform` property to `Nonreflective similarity` or

to `Affine`, and the `Method` property to `Random Sample Consensus (RANSAC)`. This property is tunable.

Default: `2.5`.

NumRandomSamplingsMethod

How to specify number of random samplings

Indicate how to specify number of random samplings as one of `Specified value | Desired confidence`. Set this property to `Desired confidence` to specify the number of random samplings as a percentage and a maximum number. This property applies when you set the `ExcludeOutliers` property to true and the `Method` property to `Random Sample Consensus (RANSAC)`.

Default: `Specified value`.

NumRandomSamplings

Number of random samplings

Specify the number of random samplings as a positive, integer value. This property applies when you set the `NumRandomSamplingsMethod` property to `Specified value`. This property is tunable.

Default: `500`.

DesiredConfidence

Probability to find largest group of points

Specify as a percentage, the probability to find the largest group of points that can be mapped by a transformation matrix. This property applies when you set the `NumRandomSamplingsMethod` property to `Desired confidence`. This property is tunable.

Default: `99`.

MaximumRandomSamples

Maximum number of random samplings

Specify the maximum number of random samplings as a positive, integer value. This property applies when you set the `NumRandomSamplingsMethod` property to `Desired confidence`. This property is tunable.

Default: `1000`.

InlierPercentageSource

Source of inlier percentage

Indicate how to specify the threshold to stop random sampling when a percentage of input point pairs have been found as inliers. You can set this property to one of `Auto | Property`. If set to `Auto` then inlier threshold is disabled. This property applies when you set the `Method` property to `Random Sample Consensus (RANSAC)`.

Default: `Auto`.

InlierPercentage

Percentage of point pairs to be found to stop random sampling

Specify the percentage of point pairs that needs to be determined as inliers, to stop random sampling. This property applies when you set the `InlierPercentageSource` property to `Property`. This property is tunable.

Default: `75`.

RefineTransformMatrix

Whether to refine transformation matrix

Set this property to `true` to perform additional iterative refinement on the transformation matrix. This property applies when you set the `ExcludeOutliers` property to `true`.

Default: `false`.

TransformMatrixDataType

Data type of the transformation matrix

Specify transformation matrix data type as one of `single` | `double` when the input points are built-in integers. This property is not used when the data type of points is single or double.

Default: `single`.

Methods

clone	Create geometric transform estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Calculate transformation matrix mapping largest number of valid points from input arrays

Examples

Recover Transformed Image Using SURF Feature Points

Detect, extract, and match SURF features from two images.

1

Read and transform input images

```
Iin  = imread('cameraman.tif'); imshow(Iin); title('Base image');
Iout = imresize(Iin, 0.7); Iout = imrotate(Iout, 31);
figure; imshow(Iout); title('Transformed image');
```

2

Detect and extract features from both images.

```
ptsIn  = detectSURFFeatures(Iin);
ptsOut = detectSURFFeatures(Iout);
[featuresIn   validPtsIn] = extractFeatures(Iin,  ptsIn);
[featuresOut validPtsOut]  = extractFeatures(Iout, ptsOut);
```

3

Match feature vectors.

```
index_pairs = matchFeatures(featuresIn, featuresOut);
```

4

Get matching points.

```
matchedPtsIn  = validPtsIn(index_pairs(:,1));
matchedPtsOut = validPtsOut(index_pairs(:,2));
figure; showMatchedFeatures(Iin,Iout,matchedPtsIn,matchedPtsOut);
title('Matched SURF points, including outliers');
```

5

Compute the transformation matrix using RANSAC.

```
gte = vision.GeometricTransformEstimator;
gte.Transform = 'Nonreflective similarity';
[tform inlierIdx] = step(gte, matchedPtsOut.Location, matchedPtsIn.Loc
figure; showMatchedFeatures(Iin,Iout,matchedPtsIn(inlierIdx),matchedPt
title('Matching inliers'); legend('inliersIn', 'inliersOut');
```

6

Recover the original image.

```
agt = vision.GeometricTransformer;
Ir = step(agt, im2single(Iout), tform);
figure; imshow(Ir); title('Recovered image');
```

Transform an Image According to Specified Points

Create and transform a checkboard image to specified points.

1

Create a checkerboard input image.

```
input = checkerboard;
[h, w] = size(input);
inImageCorners = [1 1; w 1; w h; 1 h];
outImageCorners = [4 21; 21 121; 79 51; 26 6];
hgte1 = vision.GeometricTransformEstimator('ExcludeOutliers', false);
tform = step(hgte1, inImageCorners, outImageCorners);
```

2

Use tform to transform the image.

```
hgt = vision.GeometricTransformer;
output = step(hgt, input, tform);
figure; imshow(input); title('Original image');
figure; imshow(output); title('Transformed image');
```

Troubleshooting The success of estimating the correct geometric transformation depends heavily on the quality of the input point pairs. If you chose the RANSAC or LMS algorithm, the block will randomly select point pairs to compute the transformation matrix and will use the transformation that best fits the input points. There is a chance that all of the randomly selected point pairs may contain outliers despite repeated samplings. In this case, the output transformation matrix, TForm, is invalid, indicated by a matrix of zeros.

To improve your results, try the following:

Increase the percentage of inliers in the input points.
Increase the number for random samplings.
For the RANSAC method, increase the desired confidence.

For the LMS method, make sure the input points have 50% or more inliers.

Use features appropriate for the image contents

Be aware that repeated patterns, for example, windows in office building, will cause false matches when you match the features. This increases the number of outliers.

Do not use this function if the images have significant parallax. You can use the estimateFundamentalMatrix function instead.

Choose the minimum transformation for your problem.

If a projective transformation produces the error message, "A portion of the input image was transformed to the location at infinity. Only transformation matrices that do not transform any part of the image to infinity are supported.", it is usually caused by a transformation matrix and an image that would result in an output distortion that does not fit physical reality. If the matrix was an output of the GeometricTransformationEstimator object, then most likely it could not find enough inliers.

Algorithms   This object implements the algorithm, inputs, and outputs described on the Estimate Geometric Transformation block reference page. The object properties correspond to the block parameters.

See Also   vision.GeometricTransformer | detectSURFFeatures | vision.CornerDetector | extractFeatures | estimateFundamentalMatrix

vision.GeometricTransformEstimator.clone

Purpose   Create geometric transform estimator object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a GeometricTransformEstimator System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.GeometricTransformEstimator.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   `TF = isLocked(H)`

Description   `TF = isLocked(H)` returns the locked status, `TF` of the
                   `GeometricTransformEstimator` System object.

                   The `isLocked` method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the `step` method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the `isLocked` method returns a `true` value.

vision.GeometricTransformEstimator.release

Purpose　　　　Allow property value and input characteristics changes

Syntax　　　　`release(H)`

Description　　`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note　You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Calculate transformation matrix mapping largest number of valid points from input arrays

Syntax   TFORM = step(H,MATCHED_POINTS1, MATCHED_POINTS2)
                  [TFORM,INLIER_INDEX] = step(H, ...)

Description   TFORM = step(H,MATCHED_POINTS1, MATCHED_POINTS2) calculates the transformation matrix, TFORM. The input arrays, MATCHED_POINTS1 and MATCHED_POINTS2 specify the locations of matching points in two images. The points in the arrays are stored as a set of $[x_1\ y_1;\ x_2\ y_2;\ ...;\ x_N\ y_N]$ coordinates, where N is the number of points. When you set the Transform property to Projective, the step method outputs TFORM as a 3-by-3 matrix. Otherwise, the step method outputs TFORM as a 3-by-2 matrix.

[TFORM,INLIER_INDEX] = step(H, ...) additionally outputs the logical vector, INLIER_INDEX, indicating which points are the inliers.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

vision.GeometricTranslator

Purpose   Translate image in two-dimensional plane using displacement vector

Description   The `GeometricTranslator` object translates images in two-dimensional plane using displacement vector.

Construction   `H = vision.GeometricTranslator` returns a System object, `H`. This object moves an image up or down, and left or right.

`H = vision.GeometricTranslator(`*Name*`,`*Value*`)` returns a geometric translator System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **OutputSize**

Output size as full or same as input image size

Specify the size of output image after translation as one of `Full` | `Same as input image`. The default is `Full`. When you set this property to `Full`, the object outputs a matrix that contains the translated image values. When you set this property to `Same as input image`, the object outputs a matrix that is the same size as the input image and contains a portion of the translated image.

OffsetSource

Source of specifying offset values

Specify how the translation parameters are provided as one of `Input port` | `Property`. The default is `Property`. When you set the `OffsetSource` property to `Input port`, a two-element offset vector must be provided to the object's `step` method.

Offset

Translation values

Specify the number of pixels to translate the image as a two-element offset vector. The default is `[1.5 2.3]`. The first element of the vector represents a shift in the vertical direction and a positive value moves the image downward. The second

element of the vector represents a shift in the horizontal direction and a positive value moves the image to the right. This property applies when you set the `OffsetSource` property to `Property`.

MaximumOffset

Maximum number of pixels by which to translate image

Specify the maximum number of pixels by which to translate the input image as a two-element real vector with elements greater than `0`. The default is `[8 10]`. This property must have the same data type as the `Offset` input. This property applies when you set the `OutputSize` property to `Full` and `OffsetSource` property to `Input port`. The system object uses this property to determine the size of the output matrix. If the `Offset` input is greater than this property value, the object saturates to the maximum value.

BackgroundFillValue

Value of pixels outside image

Specify the value of pixels that are outside the image as a numeric scalar value or a numeric vector of same length as the third dimension of input image. The default is `0`.

InterpolationMethod

Interpolation method used to translate image

Specify the interpolation method used to translate the image as one of `Nearest neighbor` | `Bilinear` | `Bicubic`. The default is `Bilinear`. When you set this property to `Nearest neighbor`, the object uses the value of the nearest pixel for the new pixel value. When you set this property to `Bilinear`, the object sets the new pixel value as the weighted average of the four nearest pixel values. When you set this property to `Bicubic`, the object sets the new pixel value as the weighted average of the sixteen nearest pixel values.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero. The default is Nearest.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of Wrap |Saturate. The default is Saturate.

OffsetValuesDataType

Offset word and fraction lengths

Specify the offset fixed-point data type as one of Same word length as input | Custom. The default is Same word length as input.

CustomOffsetValuesDataType

Offset word and fraction lengths

Specify the offset fixed-point type as a signed numerictype object with a Signedness of Auto. This property applies when you set the OffsetValuesDataType property to Custom. The default is numerictype([],16,6).

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Same as first input | Custom. The default is Custom. This property is applies when you set the InterpolationMethod property to Bilinear or Bicubic.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. This property is applies when you set the `InterpolationMethod` property to `Bilinear` or `Bicubic`, and the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Same as first input` | `Custom`. The default is `Same as product`. This property is applies when you set the `InterpolationMethod` property to `Bilinear` or `Bicubic`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`. This property is applies when you set the `InterpolationMethod` property to `Bilinear` or `Bicubic`, and the `AccumulatorDataType` property to `Custom`

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as first input` | `Custom`. The default is `Same as first input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when

vision.GeometricTranslator

you set the `OutputDataType` property to `Custom`. The default is
`numerictype([],32,10)`.

Methods

clone	Create geometric translator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Return a translated image

Examples

Translate an image

```
htranslate=vision.GeometricTranslator;
htranslate.OutputSize='Same as input image';
htranslate.Offset=[30 30];

I=im2single(imread('cameraman.tif'));
Y = step(htranslate,I);
imshow(Y);
```

Algorithms

This object implements the algorithm, inputs, and outputs described on
the Translate block reference page. The object properties correspond
to the block parameters.

See Also

vision.GeometricRotator

Purpose　　　Create geometric translator object with same property values

Syntax　　　　C = clone(H)

Description　　C = clone(H) creates an GeometricTranslator System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.GeometricTranslator.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.GeometricTranslator.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
                GeometricTranslator System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.GeometricTranslator.step

Purpose   Return a translated image

Syntax   `Y = step(H,I)`
       `Y = step(H,I,OFFSET)`

Description `Y = step(H,I)` translates the input image *I*, and returns a translated image *Y*, with the offset specified by the `Offset` property.

`Y = step(H,I,OFFSET)` uses input *OFFSET* as the offset to translate the image *I*. This applies when you set the `OffsetSource` property to `Input port`. *OFFSET* is a two-element offset vector that represents the number of pixels to translate the image. The first element of the vector represents a shift in the vertical direction and a positive value moves the image downward. The second element of the vector represents a shift in the horizontal direction and a positive value moves the image to the right.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Purpose　　　Generate histogram of each input matrix

Description　　The `Histogram` object generates histogram of each input matrix.

Construction　`H = vision.Histogram` returns a histogram System object, H, that computes the frequency distribution of the elements in each input matrix.

`H = vision.Histogram(`*Name*`,`*Value*`,)` returns a histogram object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

`H = vision.Histogram(MIN,MAX,NUMBINS,`*Name*`,`*Value*`,...)` returns a histogram System object, H, with the `LowerLimit` property set to MIN, `UpperLimit` property set to MAX, `NumBins` property set to NUMBINS and other specified properties set to the specified values.

Properties　　**LowerLimit**

　　　　　Lower boundary

　　　　　Specify the lower boundary of the lowest-valued bin as a real-valued scalar value. `NaN` and `Inf` are not valid values for this property. The default is `0`. This property is tunable.

　　　　UpperLimit

　　　　　Upper boundary

　　　　　Specify the upper boundary of the highest-valued bin as a real-valued scalar value. `NaN` and `Inf` are not valid values for this property. The default is `1`. This property is tunable.

　　　　NumBins

　　　　　Number of bins in the histogram

　　　　　Specify the number of bins in the histogram. The default is `256`.

　　　　Normalize

Enable output vector normalization

Specify whether the output vector, v, is normalized such that sum(v) = 1. Use of this property is not supported for fixed-point signals. The default is `false`.

RunningHistogram

Enable calculation over successive calls to step method

Set this property to `true` to enable computing the histogram of the input elements over successive calls to the `step` method. Set this property to `false` to enable basic histogram operation. The default is `false`.

ResetInputPort

Enable resetting in running histogram mode

Set this property to `true` to enable resetting the running histogram. When the property is set to `true`, a reset input must be specified to the `step` method to reset the running histogram. This property applies when you set the `RunningHistogram` property to `true`. When you set this property to `false`, the object does not reset. The default is `false`.

ResetCondition

Condition for running histogram mode

Specify event to reset the running histogram as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`. The default is `Non-zero`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

ProductDataType

Data type of product

Specify the product data type as `Full precision`, `Same as input`, or `Custom`. The default is `Full precision`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a signed, scaled `numerictype` object. This property applies only when you set the `ProductDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

AccumulatorDataType

Data type of the accumulator

Specify the accumulator data type as, `Same as input`, `Same as product`, or `Custom`. The default is `Same as input`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

vision.Histogram

Methods

clone	Create 2-D histogram object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset histogram bin values to zero
step	Return histogram for input data

Examples

Compute histogram of a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hhist2d = vision.Histogram;
y = step(hhist2d,img);
bar((0:255)/256, y);
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the 2D-Histogram block reference page. The object properties correspond to the block parameters, except:

- **Reset port** block parameter corresponds to both the ResetCondition and the ResetInputPort object properties.

- The **Find Dimensions Over** block parameter with Entire input or Each column options, does not have a corresponding property. The object finds dimensions over the entire input.

See Also   dsp.Histogram

Purpose   Create 2-D histogram object with same property values

Syntax   `C = clone(H)`

Description   `C = clone(H)` creates an `Histogram` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

vision.Histogram.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.Histogram.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the Histogram
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.Histogram.reset

Purpose   Reset histogram bin values to zero

Syntax   reset(H)

Description reset(H) sets the Histogram object bin values to zero when the
RunningHistogram property is true.

Purpose   Return histogram for input data

Syntax   Y = step(H,X)
                   Y = step(H,X,R)

Description   Y = step(H,X) returns a histogram *y* for the input data *X*. When you set
                   the RunningHistogram property to true, *Y* corresponds to the histogram
                   of the input elements over successive calls to the step method.

                   Y = step(H,X,R) computes the histogram of the input X elements over
                   successive calls to the step method, and optionally resets the object's
                   state based on the value of R and the object's ResetCondition property.
                   This applies when you set the RunningHistogram and ResetInputPort
                   properties to true.

                   Note The object performs an initialization the first time the step
                   method is executed. This initialization locks nontunable properties and
                   input specifications, such as dimensions, complexity, and data type
                   of the input data. If you change a nontunable property or an input
                   specification, the System object issues an error. To change nontunable
                   properties or inputs, you must first call the release method to unlock
                   the object.

Purpose   Histogram-based object tracking

Description   The histogram-based tracker incorporates the continuously adaptive
mean shift (CAMShift) algorithm for object tracking. It uses the
histogram of pixel values to identify the tracked object.

Use the `step` method with input image, `I`, the histogram-based tracker
object `H`, and any optional properties to identify the bounding box of
the tracked object.

`BBOX = step(H,I)` returns the bounding box, `BBOX`, of the tracked
object. The bounding box output is in the format [x y width height].
Before calling the `step` method, you must identify the object to track
and set the initial search window. Use the `initializeObject` method
to do this.

`[BBOX,ORIENTATION] = step(H,I)` additionally returns the angle
between the x-axis and the major axis of the ellipse that has the same
second-order moments as the object. The range of the returned angle
can be from –pi/2 to pi/2.

`[BBOX,ORIENTATION, SCORE] = step(H,I) )` additionally returns the
confidence score indicating whether the returned bounding box, `BBOX`,
contains the tracked object. `SCORE` is between `0` and `1`, with the greatest
confidence equal to `1`.

Use the `initializeObject` method before calling the `step` method in
order to set the object to track and to set the initial search window.

`initializeObject(H,I,R)` sets the object to track by extracting it from
the [x y width height] region `R` located in the 2-D input image, `I`. The
input image, `I`, can be any 2-D feature map that distinguishes the object
from the background. For example, the image can be a hue channel of
the HSV color space. Typically, `I` will be the first frame in which the
object appears. The region, `R`, is also used for the initial search window,
in the next call to the `step` method. For best results, the object must
occupy the majority of the region, `R`.

`initializeObject(H,I,R,N)` additionally, lets you specify `N`, the
number of histogram bins. By default, `N` is set to `16`. Increasing `N`

enhances the ability of the tracker to discriminate the object. However, this approach also narrows the range of changes to the object's visual characteristics that the tracker can accommodate. Consequently, this narrow range increases the likelihood of losing track.

Tip You can improve the computational speed of the HistogramBasedTracker object by setting the class of the input image to uint8.

Construction   H = vision.HistogramBasedTracker returns a System object, H, that tracks an object by using the CAMShift algorithm. It uses the histogram of pixel values to identify the tracked object. To initialize the tracking process, you must use the initializeObject method to specify an exemplar image of the object. After you specify the image of the object, use the step method to track the object in consecutive video frames.

BBOX= vision.HistogramBasedTracker(*Name,Value*) returns a tracker System object with one or more name-value pair arguments. Unspecified properties have default values.

Properties   **ObjectHistogram**

Normalized pixel value histogram

An *N*-element vector. This vector specifies the normalized histogram of the object's pixel values. Histogram values must be normalized to a value between 0 and 1. You can use the initializeObject method to set the property. This property is tunable.

Default: []

Methods   clone   Create histogram-based tracker object with same property values

initializeObject   Set object to track

initializeSearchWindow	Initialize search window
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Histogram-based object tracking

Examples   **Track a Face**

Track and display a face in each frame of an input video.

1

Create System objects for reading and displaying video and for drawing a bounding box of the object.

```
videoFileReader = vision.VideoFileReader('vipcolorsegmentation.avi');
videoPlayer = vision.VideoPlayer();
shapeInserter = vision.ShapeInserter('BorderColor','Custom','CustomBorder
```

2

Read the first video frame, which contains the object. Convert the image to HSV color space. Then define and display the object region.

```
objectFrame = step(videoFileReader);
objectHSV = rgb2hsv(objectFrame);
objectRegion = [40, 45, 25, 25];
objectImage = step(shapeInserter, objectFrame, objectRegion);
figure; imshow(objectImage); title('Red box shows object region');
```

Optionally, you can select the object region using your mouse. The object must occupy the majority of the region. Use the following command.
```
figure; imshow(objectFrame);
objectRegion=round(getPosition(imrect))
```

3

Set the object, based on the hue channel of the first video frame.

```
tracker = vision.HistogramBasedTracker;
initializeObject(tracker, objectHSV(:,:,1) , objectRegion);
```

4

Track and display the object in each video frame. The while loop reads each image frame, converts the image to HSV color space, then tracks the object in the hue channel where it is distinct from the background. Finally, the example draws a box around the object and displays the results.

```
while ~isDone(videoFileReader)
  frame = step(videoFileReader);
  hsv = rgb2hsv(frame);
  bbox = step(tracker, hsv(:,:,1));

  out = step(shapeInserter, frame, bbox);
  step(videoPlayer, out);
end
```

5

Release the video reader and player.

```
release(videoPlayer);
release(videoFileReader);
```

References Bradski, G. and A. Kaehler, *Learning OpenCV :Computer Vision with the OpenCV Library*, O'Reilly Media Inc.: Sebastopol, CA, 2008.

See Also size | imrect | rgb2hsvvision.ShapeInserter |

vision.HistogramBasedTracker.clone

Purpose   Create histogram-based tracker object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a `HistogramBasedTracker` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object.

vision.HistogramBasedTracker.initializeObject

Purpose   Set object to track

Syntax   initializeObject(H,I,R)

Description Use the `initializeObject` method to set the object to track, and to set the initial search window. Use this method before calling the `step` method.

`initializeObject(H,I,R)` sets the object to track by extracting it from the [x y width height] region R located in the 2-D input image, I. The input image, I, can be any 2-D feature map that distinguishes the object from the background. For example, the image can be a hue channel of the HSV color space. Typically, I will be the first frame in which the object appears. The region, R, is also used for the initial search window, in the next call to the `step` method. For best results, the object must occupy the majority of the region, R.

`initializeObject(H,I,R,N)` additionally, lets you specify N, the number of histogram bins. By default, N is set to 16. Increasing N enhances the ability of the tracker to discriminate the object. However, this approach also narrows the range of changes to the object's visual characteristics that the tracker can accommodate. Consequently, this narrow range increases the likelihood of losing track.

vision.HistogramBasedTracker.initializeSearchWindow

Purpose   Initialize search window

Syntax   initializeSearchWindow(H,R)

Description initializeSearchWindow(H,R) sets the initial search window, R, specified in the format, [x y width height]. The next call to the step method will use R as the initial window to search for the object. Use this method when you lose track of the object. You can also use it to re-initialize the object's initial location and size.

Purpose   Locked status for input attributes and nontunable properties

Syntax   isLocked(H)

Description   isLocked(H) returns the locked state of the histogram-based detector.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

vision.HistogramBasedTracker.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
or hardware connections) and allows all properties and input
characteristics to be changed.

Note You cannot use the release method on System objects in
Embedded MATLAB.

Purpose   Histogram-based object tracking

Syntax   BBOX = step(H,I)
                  [BBOX,ORIENTATION] = step(H,I)
                  [BBOX,ORIENTATION, SCORE] = step(H,I)

Description   BBOX = step(H,I) returns the bounding box, BBOX, of the tracked
                  object. The bounding box output is in the format [x y width height].
                  Before calling the step method, you must identify the object to track,
                  and set the initial search window. Use the initializeObject method
                  to do this.

                  [BBOX,ORIENTATION] = step(H,I) additionally returns the angle
                  between the x-axis and the major axis of the ellipse that has the same
                  second-order moments as the object. The returned angle is between
                  –pi/2 and pi/2.

                  [BBOX,ORIENTATION, SCORE] = step(H,I) additionally returns the
                  confidence score for the returned bounding box, BBOX, that contains the
                  tracked object. SCORE is between 0 and 1, with the greatest confidence
                  equal to 1.

                  Before calling the step method, you must identify the object to track,
                  and set the initial search window. Use the initializeObject method
                  to do this.

                  Note The object performs an initialization the first time the step
                  method is executed. This initialization locks nontunable properties and
                  input specifications, such as dimensions, complexity, and data type
                  of the input data. If you change a nontunable property or an input
                  specification, the System object issues an error. To change nontunable
                  properties or inputs, you must first call the release method to unlock
                  the object.

vision.HistogramEqualizer

Purpose	Enhance contrast of images using histogram equalization

Description   The `HistogramEqualizer` object enhances contrast of images using histogram equalization.

Construction   `H = vision.HistogramEqualizer` returns a System object, `H`. This object enhances the contrast of input image using histogram equalization.

`H = vision.HistogramEqualizer(`*Name*,*Value*`)` returns a histogram equalization object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Histogram**

How to specify histogram

Specify the desired histogram of the output image as one of `Uniform` | `Input port` | `Custom`. The default is `Uniform`.

CustomHistogram

Desired histogram of output image

Specify the desired histogram of output image as a numeric vector. This property applies when you set the `Histogram` property to `Custom`.

The default is `ones(1,64)`.

BinCount

Number of bins for uniform histogram

Specify the number of equally spaced bins the uniform histogram has as an integer scalar value greater than 1. This property applies when you set the `Histogram` property to `Uniform`. The default is `64`.

Methods

clone	Create histogram equalizer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform histogram equalization on input image

Examples   Enhance quality of an image:

```
hhisteq = vision.HistogramEqualizer;
x = imread('tire.tif');
y = step(hhisteq, x);
imshow(x); title('Original Image');
figure, imshow(y);
title('Enhanced Image after histogram equalization');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Histogram Equalization block reference page. The object properties correspond to the block parameters, except:

The Histogram property for the object, corresponds to both the **Target Histogram** and the **Histogram Source** parameters for the block.

See Also   vision.Histogram

vision.HistogramEqualizer.clone

Purpose Create histogram equalizer object with same property values

Syntax C = clone(H)

Description C = clone(H) creates an HistogramEqualizer System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

vision.HistogramEqualizer.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                  HistogramEqualizer System object.

                  The isLocked method returns a logical value that indicates whether
                  input attributes and nontunable properties for the object are locked. The
                  object performs an internal initialization the first time the step method
                  is executed. This initialization locks nontunable properties and input
                  specifications, such as dimensions, complexity, and data type of the
                  input data. After locking, the isLocked method returns a true value.

vision.HistogramEqualizer.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose　　　Perform histogram equalization on input image

Syntax　　　Y = step(H,X)
　　　　　　　　Y = step(H,X,HIST)

Description　　Y = step(H,X) performs histogram equalization on input image, *X*, and returns the enhanced image, *Y*.

Y = step(H,X,HIST) performs histogram equalization on input image, *X* using input histogram, *HIST*, and returns the enhanced image, *Y* when you set the Histogram property to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.HoughLines

Purpose   Find Cartesian coordinates of lines that are described by rho and theta pairs

Description   The `HoughLines` object finds Cartesian coordinates of lines that are described by rho and theta pairs. The object inputs are the theta and rho values of lines and a reference image. The object outputs the zero-based row and column positions of the intersections between the lines and two of the reference image boundary lines. The boundary lines are the left and right vertical boundaries and the top and bottom horizontal boundaries of the reference image.

Construction   `H = vision.HoughLines` returns a Hough lines System object, `H`, that finds Cartesian coordinates of lines that are described by rho and theta pairs.

`H = vision.HoughLines(Name,Value)` returns a Hough lines object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **SineComputation**

Method to calculate sine values used to find intersections of lines

Specify how to calculate sine values which are used to find intersection of lines as `Trigonometric function`, or `Table lookup`. If this property is set to `Trigonometric function`, the object computes sine and cosine values it needs to calculate the intersections of the lines. If it is set to `Table lookup`, the object computes and stores the trigonometric values it needs to calculate the intersections of the lines in a table and uses the table for each step call. In this case, the object requires extra memory. For floating-point inputs, this property must be set to `Trigonometric function`. For fixed-point inputs, the property must be set to `Table lookup`. The default is `Table lookup`.

ThetaResolution

Spacing of the theta-axis

Specify the spacing of the theta-axis. This property applies when you set the `SineComputation` property to `Table lookup`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`. This property applies when you set the `SineComputation` property to `Table lookup`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the `SineComputation` property to `Table lookup`. The default is `Wrap`.

SineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point data type as a constant property always set to `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

CustomSineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `SineTableDataType` property to `Custom`. The default is `numerictype([],16)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Same as first input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `ProductDataType` property to `Custom`. The default is `numerictype([],32,16)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup`, and the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,16)`.

Methods	clone	Create hough lines object with same property values
	getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Output intersection coordinates of a line described by a theta and rho pair and reference image boundary lines

Examples   Use Hough lines to detect the longest line in an image. Use the edge detection object to find edges in the intensity image. The step method outputs a binary image required by the HoughTransform object and improves the efficiency of the HoughLines object.

```
I = imread('circuit.tif');
hedge = vision.EdgeDetector;
hhoughtrans = vision.HoughTransform(pi/360,'ThetaRhoOutputPort', true
hfindmax = vision.LocalMaximaFinder(1, 'HoughMatrixInput', true);
hhoughlines = vision.HoughLines('SineComputation','Trigonometric func

% Find the edges in the intensity image
  BW = step(hedge, I);

% Run the edge output through the transform
  [ht, theta, rho] = step(hhoughtrans, BW);

% Find the location of the max value in the Hough matrix.
   idx = step(hfindmax, ht);

% Find the longest line.
   linepts = step(hhoughlines, theta(idx(1)-1), rho(idx(2)-1), I);

% View the image superimposed with the longest line.
```

vision.HoughLines

```
imshow(I); hold on;
line(linepts([1 3])-1, linepts([2 4])-1,'color',[1 1 0]);
```

Algorithms This object implements the algorithm, inputs, and outputs described on the Hough Lines block reference page. The object properties correspond to the block parameters.

See Also vision.HoughTransform | vision.LocalMaximaFinder | vision.EdgeDetector

Purpose   Create hough lines object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a HoughLines System object *C*, with the same
property values as *H*. The clone method creates a new unlocked object.

vision.HoughLines.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.HoughLines.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the HoughLines System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

vision.HoughLines.step

Purpose   Output intersection coordinates of a line described by a theta and rho pair and reference image boundary lines

Syntax   PTS = step(H,THETA,RHO,REFIMG)

Description   PTS = step(H,THETA,RHO,REFIMG) outputs PTS as the zero-based row and column positions of the intersections between the lines described by THETA and RHO and two of the reference image boundary lines.

> **Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Find lines in images via Hough transform

Description   The HoughTransform object finds lines in images via Hough transform. The Hough transform maps points in the Cartesian image space to curves in the Hough parameter space using the following equation:

$$rho = x * \cos(theta) + y * \sin(theta)$$

Here, *rho* denotes the distance from the origin to the line along a vector perpendicular to the line, and *theta* denotes the angle between the *x*-axis and this vector. This object computes the parameter space matrix, whose rows and columns correspond to the rho and theta values respectively. Peak values in this matrix represent potential straight lines in the input image.

Construction   H = vision.HoughTransform returns a Hough transform System object, H, that implements the Hough transform to detect lines in images.

H = vision.HoughTransform(*Name*,*Value*) returns a Hough transform object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

H = vision.HoughTransform(THETARES, RHORES,'*Name*',*Value*,...) returns a Hough transform object, H, with the ThetaResolution property set to THETARES, the RhoResolution property set to RHORES, and other specified properties set to the specified values.

Properties   **ThetaResolution**

Theta resolution in radians

Specify the spacing of the Hough transform bins along the theta-axis in radians, as a scalar numeric value between 0 and pi/2. The default is pi/180.

RhoResolution

Rho resolution

Specify the spacing of the Hough transform bins along the rho-axis as a scalar numeric value greater than 0. The default is 1.

ThetaRhoOutputPort

Enable theta and rho outputs

Set this property to `true` for the object to output theta and rho values. The default is `false`.

OutputDataType

Data type of output

Specify the data type of the output signal as `double`, `single`, or `Fixed point`. The default is `double`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.This property applies when you set the `OutputDataType` property to `Fixed point`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the `OutputDataType` property to `Fixed point`. The default is `Saturate`.

SineTableDataType

Sine table word and fraction lengths

This property is constant and is set to Custom. This property applies when you set the OutputDataType property to Fixed point.

CustomSineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Fixed point. The default is numerictype([],16,14).

RhoDataType

Rho word and fraction lengths

This property is constant and is set to Custom. This property applies when you set the OutputDataType property to Fixed point.

CustomRhoDataType

Rho word and fraction lengths

Specify the rho fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Fixed point. The default is numerictype([],32,16).

ProductDataType

Product word and fraction lengths

This property is constant and is set to Custom. This property applies when you set the OutputDataType property to Fixed point.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you

set the `OutputDataType` property to `Fixed point`. The default is `numerictype([],32,20)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Custom`. This property applies when you set the `OutputDataType` property to `Fixed point`. The default is `Custom`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Fixed point`. The default is `numerictype([],32,20)`.

HoughOutputDataType

Hough output word and fraction lengths

This property is constant and is set to `Custom`. This property applies when you set the `OutputDataType` property to `Fixed point`.

CustomHoughOutputDataType

Hough output word and fraction lengths

Specify the hough output fixed-point data type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Fixed point`. The default is `numerictype(false,16)`.

ThetaOutputDataType

Theta output word and fraction lengths

This property is constant and is set to `Custom`. This property applies when you set the `OutputDataType` property to `Fixed point`.

CustomThetaOutputDataType

Theta output word and fraction lengths

Specify the theta output fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Fixed point. The default is numerictype([],32,16).

Methods

clone	Create Hough transform object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Output parameter space matrix for binary input image matrix

Examples

Compute the hough transform of an image. Use the hough transform to detect the longest line in the image.

```
 I = imread('circuit.tif');

% Use the EdgeDetection System object to find edges in the
% intensity image. This step outputs a binary image
% required by the HoughTransform System object and
% improves the efficiency of the HoughLines System object.
 hedge = vision.EdgeDetector;
 hhoughtrans = vision.HoughTransform(pi/360,'ThetaRhoOutputPort', tr
 hfindmax = vision.LocalMaximaFinder(1,'HoughMatrixInput', true);
```

```
hhoughlines = vision.HoughLines('SineComputation','Trigonometric functi

% Find the edges in the intensity image
  BW = step(hedge, I);

% Run the edge output through the transform
  [ht, theta, rho] = step(hhoughtrans, BW);

% Find the location of the max value in the Hough matrix.
  idx = step(hfindmax, ht);

% Find the longest line.
  linepts = step(hhoughlines, theta(idx(1)), rho(idx(2)), I);

% View the image superimposed with the longest line.
   imshow(I); hold on;
   line(linepts([1 3]), linepts([2 4]));
```

Algorithms   This object implements the algorithm, inputs, and outputs described
                  on the Hough Transform block reference page. The object properties
                  correspond to the block parameters.

See Also   vision.DCT | vision.HoughLines | vision.LocalMaximaFinder |
                  vision.EdgeDetector

Purpose   Create Hough transform object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a HoughTransform System object C, with the
same property values as *H*. The clone method creates a new unlocked
object.

vision.HoughTransform.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   `N = getNumInputs(H)`

Description   `N = getNumInputs(H)` returns the number of expected inputs, *N* to the `step` method.

The `getNumInputs` method returns a positive integer that is the number of expected inputs (not counting the object itself) to the `step` method. This value will change if you alter any properties that turn inputs on or off. You must call the `step` method with the number of input arguments equal to the result of `getNumInputs(`*H*`)`.

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.HoughTransform.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the HoughTransform
                     System object.

                     The isLocked method returns a logical value that indicates whether
                     input attributes and nontunable properties for the object are locked. The
                     object performs an internal initialization the first time the step method
                     is executed. This initialization locks nontunable properties and input
                     specifications, such as dimensions, complexity, and data type of the
                     input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.HoughTransform.step

Purpose   Output parameter space matrix for binary input image matrix

Syntax   HT = step(H,BW)
[HT,THETA,RHO] = step(H,BW)

Description   HT = step(H,BW) outputs the parameter space matrix, HT , for the binary input image matrix BW.

[HT,THETA,RHO] = step(H,BW) also returns the theta and rho values, in vectors THETA and RHO respectively, when you set the ThetaRhoOutputPort property to true. RHO denotes the distance from the origin to the line along a vector perpendicular to the line, and THETA denotes the angle between the x-axis and this vector. This object computes the parameter space matrix, whose rows and columns correspond to the rho and theta values respectively. Peak values in this matrix represent potential straight lines in the input image.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Compute 2-D inverse discrete cosine transform

Description The IDCT object computes 2-D inverse discrete cosine transform of the input signal. The number of rows and columns of the input matrix must be a power of 2.

Construction `H = vision.IDCT` returns a System object, `H`, used to compute the two-dimensional inverse discrete cosine transform (2-D IDCT) of a real input signal.

`H = vision.IDCT(`*Name,Value*`)` returns a 2-D inverse discrete cosine transform System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties **SineComputation**

Specify how the System object computes sines and cosines as `Trigonometric function`, or `Table lookup`. This property must be set to `Table lookup` for fixed-point inputs.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`. This property applies when you set the `SineComputation` to `Table lookup`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the `SineComputation` to `Table lookup`. The default is `Wrap`.

SineTableDataType

Sine table word-length designation

Specify the sine table fixed-point data type as `Same word length as input`, or `Custom`. This property applies when you set the `SineComputation` to `Table lookup`. The default is `Same word length as input`.

CustomSineTableDataType

Sine table word length

Specify the sine table fixed-point type as a signed, unscaled `numerictype` object. This property applies when you set the `SineComputation` to `Table lookup` and you set the `SineTableDataType` property to `Custom`. The default is `numerictype(true,16)`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Full precision`, `Same as first input`, or `Custom`. This property applies when you set the `SineComputation` to `Table lookup`. The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `SineComputation` to `Table lookup`, and the `ProductDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Full precision`, `Same as input`,`Same as product`, `Same as first input`,

Custom. This property applies when you set the SineComputation property to Table lookup. The default is Full precision.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled numerictype object. This property applies when you set the SineComputation to Table lookup, and AccumulatorDataType property to Custom. The default is numerictype(true,32,30).

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as Full precision, Same as first input, or Custom. This property applies when you set the SineComputation to Table lookup. The default is Custom.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a signed, scaled numerictype object. This property applies when you set the SineComputation to Table lookup, and the OutputDataType property to Custom. The default is numerictype(true,16,15).

Methods		
	clone	Create 2-D inverse discrete cosine transform object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties

vision.IDCT

release	Allow property value and input characteristics changes
step	Compute 2-D inverse discrete cosine transform of the input

Examples

Use 2-D discrete cosine transform (DCT) to analyze the energy content in an image. Set the DCT coefficients lower than a threshold of 0. Reconstruct the image using 2-D inverse discrete cosine transform (IDCT).

```
hdct2d = vision.DCT;
I = double(imread('cameraman.tif'));
J = step(hdct2d, I);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar

hidct2d = vision.IDCT;
J(abs(J) < 10) = 0;
It = step(hidct2d, J);
figure, imshow(I, [0 255]), title('Original image')
figure, imshow(It,[0 255]), title('Reconstructed image')
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the 2-D IDCT block reference page. The object properties correspond to the block parameters.

See Also   vision.DCT

Purpose   Create 2-D inverse discrete cosine transform object with same property
                   values

Syntax   C = clone(H)

Description   C = clone(H) creates a IDCT System object *C*, with the same property
                   values as *H*. The clone method creates a new unlocked object.

vision.IDCT.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.IDCT.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the IDCT System
object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note   You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.IDCT.step

Purpose   Compute 2-D inverse discrete cosine transform of the input

Syntax   Y = step(H,X)

Description Y = step(H,X) computes the 2-D inverse discrete cosine transform, *Y*, of input *X*.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Two–dimensional inverse discrete Fourier transform

Description   The vision.IFFT object computes the inverse 2D discrete Fourier transform (IDFT) of a two-dimensional input matrix. The object uses one or more of the following fast Fourier transform (FFT) algorithms depending on the complexity of the input and whether the output is in linear or bit-reversed order:

- Double-signal algorithm

- Half-length algorithm

- Radix-2 decimation-in-time (DIT) algorithm

- Radix-2 decimation-in-frequency (DIF) algorithm

- An algorithm chosen by FFTW [1], [2]

Construction   H = vision.IFFT returns a 2D IFFT object, H, with the default property and value pair settings.

H = vision.IFFT(*Name*,*Value*) returns a 2D IFFT object, H, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **FFTImplemenation**

FFT implementation

Specify the implementation used for the FFT as one of Auto | Radix-2 | FFTW. When you set this property to Radix-2, the FFT length must be a power of two.

BitReversedInput

Indicates whether input is in bit-reversed order

Set this property to true if the order of 2D FFT transformed input elements are in bit-reversed order. The default is false, which denotes linear ordering.

ConjugateSymmetricInput

Indicates whether input is conjugate symmetric

Set this property to `true` if the input is conjugate symmetric. The 2D DFT of a real valued signal is conjugate symmetric and setting this property to `true` optimizes the 2D IFFT computation method. Setting this property to `false` for conjugate symmetric inputs results in complex output values with nonzero imaginary parts. Setting this property to `true` for non conjugate symmetric inputs results in invalid outputs. This property must be `false` for fixed-point inputs. The default is `true`.

Normalize

Divide output by FFT length

Specify if the 2D IFFT output should be divided by the FFT length. The value of this property defaults to `true` and divides each element of the output by the product of the row and column dimensions of the input matrix.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. This property applies when you set the `ConjugateSymmetricInput` property to `false`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the `ConjugateSymmetricInput` property to `false`. The default is `Wrap`.

SineTableDataType

Sine table word and fraction lengths

Specify the sine table data type as `Same word length as input`, `Custom`. This property applies when you set the `ConjugateSymmetricInput` property to `false`. The default is `Same word length as input`.

CustomSineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `ConjugateSymmetricInput` property to `false` and the `SineTableDataType` property is `Custom`. The default is `numerictype([],16)`.

ProductDataType

Product word and fraction lengths

Specify the product data type as `Full precision`, `Same as input`, or `Custom`. This property applies when you set the `ConjugateSymmetricInput` property to `false`. The default is `Full precision`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ConjugateSymmetricInput` property to `false` and the `ProductDataType` property to `Custom`. The default is `numerictype([],32,30)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator data type as `Full precision`, `Same as input`, `Same as product`, or `Custom`. This property applies when

you set the `ConjugateSymmetricInput` property to `false`. The default is `Full precision`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ConjugateSymmetricInput` property to `false` and the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,30)`.

OutputDataType

Output word and fraction lengths

Specify the output data type as `Full precision`, `Same as input`, or `Custom`. This property applies when you set the `ConjugateSymmetricInput` property to `false`. The default is `Full precision`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ConjugateSymmetricInput` property to `false` and the `OutputDataType` property is `Custom`. The default is `numerictype([],16,15)`.

Methods

clone	Create IFFT object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Compute 2D inverse discrete Fourier transform

Examples   Use the 2D IFFT object to convert an intensity image.

```
hfft2d = vision.FFT;
hifft2d = vision.IFFT;

% Read in the image
  xorig = single(imread('cameraman.tif'));

% Convert the image from the spatial
% to frequency domain and back
  Y = step(hfft2d, xorig);
  xtran = step(hifft2d, Y);

% Display the newly generated intensity image
  imshow(abs(xtran), []);
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the 2-D IFFT block reference page. The object properties correspond to the Simulink block parameters.

References   [1] FFTW (http://www.fftw.org)

[2] Frigo, M. and S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT,"*Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, 1998, pp. 1381-1384.

See Also   vision.FFT | vision.DCT | vision.IDCT

vision.IFFT.clone

Purpose   Create IFFT object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an IFFT System object *C*, with the same property
                     values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   getNumInputs(H)

Description   getNumInputs(H) returns the number of expected inputs to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.IFFT.getNumOutputs

Purpose   Number of outputs from step method

Syntax   getNumOutputs(H)

Description   getNumOutputs(H) returns the number of output arguments from the
step method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   isLocked(H)

Description   isLocked(H) returns the locked state of the IFFT object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

vision.IFFT.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute 2D inverse discrete Fourier transform

Syntax   Y = step(H,X)

Description Y = step(H,X) computes the 2D inverse discrete Fourier transform
(IDFT), *Y*, of an *M*-by-*N* input matrix *X*, where the values of *M* and *N*
are integer powers of two.

vision.ImageComplementer

Purpose   Complement of pixel values in binary or intensity image

Description   The `ImageComplementer` object computes the complement of pixel values in binary or intensity image. For binary images, the object replaces pixel values equal to `0` with `1`, and pixel values equal to `1` with `0`. For an intensity image, the object subtracts each pixel value from the maximum value the data input type can represent and then outputs the difference.

Construction   `H = vision.ImageComplementer` returns an image complement System object, `H`. The object computes the complement of a binary, intensity, or `RGB` image.

Methods

clone	Create image complementer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Compute complement of input image

Examples   Compute the complement of an input image.

```
himgcomp = vision.ImageComplementer;
hautoth = vision.Autothresholder;

% Read in image
I = imread('coins.png');
```

```
% Convert the image to binary
 bw = step(hautoth, I);

% Take the image complement
 Ic = step(himgcomp, bw);

% Display the results
  figure;
 subplot(2,1,1), imshow(bw), title('Original Binary image')
  subplot(2,1,2), imshow(Ic), title('Complemented image')
```

Algorithms   This object implements the algorithm, inputs, and outputs described
on the Image Complement block reference page. The object properties
correspond to the block parameters.

See Also   vision.Autothresholder

vision.ImageComplementer.clone

Purpose   Create image complementer object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an ImageComplementer System object C, with the same property values as H. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.ImageComplementer.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                   ImageComplementer System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

vision.ImageComplementer.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note   You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute complement of input image

Syntax   Y = step(H,X)

Description   Y = step(H,X) computes the complement of an input image *X*.

> **Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

vision.ImageDataTypeConverter

Purpose　　Convert and scale input image to specified output data type

Description　　The `ImageDataTypeConverter` object converts and scales an input image to a specified output data type. When converting between floating-point data types, the object casts the input into the output data type and clips values outside the range, to `0` or `1`. When converting between all other data types, the object casts the input into the output data type. The object then scales the data type values into the dynamic range of the output data type. For double- and single-precision floating-point data types, the object sets the dynamic range between `0` and `1`. For fixed-point data types, the object sets the dynamic range between the minimum and maximum values that the data type can represent.

Construction　　`H = vision.ImageDataTypeConverter` returns a System object, `H`, that converts the input image to a single precision data type.

`H = vision.ImageDataTypeConverter(`*Name*`,`*Value*`)` returns an image data type conversion object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties　　**OutputDataType**

Data type of output

Specify the data type of the output signal as one of `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `boolean` | `Custom`. The default is `single`.

Fixed-Point Properties

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a signed or unsigned, scaled `numerictype` object. This property applies when you

set the `OutputDataType` property to `Custom`. The default is `numerictype([],16,0)`.

Methods

clone	Create image data type converter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Convert data type of input image

Examples

Convert the image datatype from uint8 to single.

```
x = imread('pout.tif');
hidtypeconv = vision.ImageDataTypeConverter;
y = step(hidtypeconv, x);
imshow(y);
whos y % Image has been converted from uint8 to single.
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Image Data Type Conversion block reference page. The object properties correspond to the block parameters.

See Also

vision.ColorSpaceConverter

vision.ImageDataTypeConverter.clone

Purpose   Create image data type converter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a ImageDataTypeConverter System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.ImageDataTypeConverter.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
ImageDataTypeConverter System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.ImageDataTypeConverter.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Convert data type of input image

Syntax   Y = step(H,X)

Description   Y = step(H,X) converts the data type of the input image *X*. You can set the data type of the converted output image *Y* with the OutputDataType property.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.ImageFilter

Purpose   Perform 2-D FIR filtering of input matrix

Description   The `ImageFilter` object performs 2-D FIR filtering of input matrix.

Construction `H = vision.ImageFilter` returns a System object, `H`. This object performs two-dimensional `FIR` filtering of an input matrix using the specified filter coefficient matrix.

`H = vision.ImageFilter(`*Name*`,`*Value*`)` returns an image filter System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **SeparableCoefficients**

Set to `true` if filter coefficients are separable

Using separable filter coefficients reduces the amount of calculations the object must perform to compute the output. The function `isfilterseparable` can be used to check filter separability. The default is `false`.

CoefficientsSource

Source of filter coefficients

Indicate how to specify the filter coefficients as one of `Property` | `Input port`. The default is `Property`.

Coefficients

Filter coefficients

Specify the filter coefficients as a real or complex-valued matrix. This property applies when you set the `SeparableCoefficients` property to `false` and the `CoefficientsSource` property to `Property`. The default is `[1 0; 0 -1]`.

VerticalCoefficients

Vertical filter coefficients for the separable filter

Specify the vertical filter coefficients for the separable filter as a vector. This property applies when you set the SeparableCoefficients property to true and the CoefficientsSource property to Property. The default is [4 0].

HorizontalCoefficients

Horizontal filter coefficients for the separable filter

Specify the horizontal filter coefficients for the separable filter as a vector. This property applies when you set the SeparableCoefficients property to true and the CoefficientsSource property to Property. The default is [4 0].

OutputSize

Output size as full, valid or same as input image size

Specify how to control the size of the output as one of Full | Same as first input | Valid. The default is Full. When you set this property to Full, the object outputs the image dimensions in the following way:

output rows = input rows + filter coefficient rows + 1
output columns = input columns + filter coefficient columns + 1

When you set this property to Same as first input, the object outputs the same dimensions as the input image.

When you set this property to Valid, the object filters the input image only where the coefficient matrix fits entirely within it, and no padding is required. In this case, the dimensions of the output image are as follows:

output rows = input rows - filter coefficient rows + 1
output columns = input columns - filter coefficient columns + 1

PaddingMethod

How to pad boundary of input matrix

Specify how to pad the boundary of input matrix as one of
`Constant | Replicate | Symmetric, | Circular`. The default is
`Constant`. Set this property to one of the following:

- `Constant` to pad the input matrix with a constant value

- `Replicate` to pad the input matrix by repeating its border
  values

- `Symmetric` to pad the input matrix with its mirror image

- `Circular` to pad the input matrix using a circular repetition
  of its elements

This property applies when you set the `OutputSize` property to
`Full` or to `Same as first input`.

PaddingValueSource

Source of padding value

Specify how to define the constant boundary value as one of
`Property | Input port`. This property applies when you set the
`PaddingMethod` property to `Constant`. The default is `Property`.

PaddingValue

Constant value with which to pad matrix

Specify a constant value with which to pad the input matrix. This
property applies when you set the `PaddingMethod` property to
`Constant` and the `PaddingValueSource` property to `Property`.
The default is `0`. This property is tunable.

Method

Method for filtering input matrix

Specify the method by which the object filters the input matrix as
one of `Convolution | Correlation`. The default is `Convolution`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling| Convergent| Floor| Nearest| Round| Simplest| Zero. The default is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of Wrap| Saturate. The default is Wrap.

CoefficientsDataType

Coefficients word and fraction lengths

Specify the coefficients fixed-point data type as one of Same word length as input| Custom. This property applies when you set the CoefficientsSource property to Property. The default is Custom.

CustomCoefficientsDataType

Coefficients word and fraction lengths

Specify the coefficients fixed-point type as a signed or unsigned numerictype object. This property applies when you set the CoefficientsSource property to Property and the CoefficientsDataType property to Custom. The default is numerictype([],16).

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Same as input| Custom. The default is Custom

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as an auto-signed, scaled `numerictype` object. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype([],32,10)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product`| `Same as input`| `Custom`. The default is `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as an auto-signed, scaled `numerictype` object. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype([],32,10)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as input`| `Custom`. The default is `Same as input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype`. This property applies when you set the `OutputDataType` property to `Custom`. The default is `numerictype([],32,12)`.

Methods

clone	Create image filter object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Filter input image

Examples   Filter an image to enhance the edges of 45 degree

```
img = im2single(rgb2gray(imread('peppers.png')));
hfir2d = vision.ImageFilter;
hfir2d.Coefficients = [1 0; 0 -.5];
fImg = step(hfir2d, img);
subplot(2,1,1);imshow(img);title('Original image')
subplot(2,1,2);imshow(fImg);title('Filtered image')
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the 2-D FIR Filter block reference page. The object properties correspond to the block parameters.

See Also   vision.MedianFilter

vision.ImageFilter.clone

Purpose   Create image filter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an ImageFilter System object *C*, with the same property values as *H*. The clone method creates a new unlocked object with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.ImageFilter.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the ImageFilter
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.ImageFilter.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Filter input image

Syntax
```
Y = step(H,I)
Y = step(H,I,COEFFS)
Y = step(H,I,HV,HH)
Y = step(H,...,PVAL)
```

Description   Y = step(H,I) filters the input image *I* and returns the filtered image *Y*.

Y = step(H,I,COEFFS) uses filter coefficients, *COEFFS*, to filter the input image. This applies when you set the CoefficientsSource property to Input port and the SeparableCoefficients property to false.

Y = step(H,I,HV,HH) uses vertical filter coefficients, *HV*, and horizontal coefficients, *HH*, to filter the input image. This applies when you set the CoefficientsSource property to Input port and the SeparableCoefficients property to true.

Y = step(H,...,PVAL) uses the input pad value *PVAL* to filter the input image *I*. This applies when you set the OutputSize property to either Full or Same as first input, the PaddingMethod property to Constant, and the PaddingValueSource property to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

integralKernel

Purpose   Integral image filter

Description   This object describes box filters to use with integral images.

Construction   H = integralKernel(BBOX,WEIGHTS) defines a box filter, H, from an *M*-by-4 matrix of bounding boxes, BBOX, and an *M*-length vector, WEIGHTS, of corresponding weights. Each row of BBOX corresponds to a different region, which you can use to sum the pixels. The bounding boxes can overlap. The format of each row, [x y width height], contains the coordinates of the upper-left corner and size of each region. See "Define a Vertical Derivative Filter" on page 2-481 for an example of how to specify a box filter.

Input Arguments

BBOX

M-by-4 matrix of bounding boxes. Each of the *M* number of 4-element row vectors [x y width height] define a bounding box. The first two elements represent the coordinates of the upper-left corner of the bounding box. The second two elements represent the width and height of the bounding box.

WEIGHTS

M length vector of weights corresponding to the bounding boxes.

Properties   The following properties are read-only

BoundingBoxes

Bounding boxes which define the filter. The format is [x y width height].

Weights

Vector containing a weight for each bounding box.

Coefficients

Conventional filter coefficients.

Center

Center coordinates of the filter.

Size

Filter size.

Methods   transpose   Transpose filter

Examples   ### Define a Vertical Derivative Filter

Specify a conventional filter with coefficients.

Specify a conventional filter with the following coefficients.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

Use the format [x y width height] for the two bounding boxes.

First region   x = 1, y = 1, width = 4, height = 2, weight = 1

Second region   x = 1, y = 3, width = 4, height = 2, weight = –1

Enter the specification as follows.

```
boxH = integralKernel([1 1 4 2; 1 3 4 2], [1, -1]);
```

Define an Average Filter

Define an 11-by-11 average filter.

```
avgH = integralKernel([1 1 11 11], 1/11^2);
```

Define a Filter Approximating a Second-Order Partial Derivative

Define filter which approximates second order partial derivative in XY direction. This filter can be used by the SURF feature detector.

```
% The filter is zero-padded, using weight of zero, so that it is centered
xydH = integralKernel([1,1,9,9; 2,2,3,3; 6,2,3,3; 2 6 3 3; 6 6 3 3], [0,
```

```
% Visualize the filter
  imshow(xydH.Coefficients, [], 'InitialMagnification', 'fit');
```

```
% Mark the filter center
  hold on; plot(xydH.Center(1), xydH.Center(2), 'o');
  impixelregion;
```

Define a Filter to Approximate a Gaussian Second-order Partial Derivative

Define a filter to approximate a Gaussian second order partial derivative in Y direction.

```
ydH = integralKernel([1,1,5,9;1,4,5,3], [1, -3]);
```

```
% Note that this same filter could have been defined as:
%   integralKernel([1,1,5,3;1,4,5,3;1,7,5,3], [1, -2, 1]);
% but it would be less efficient since it requires 3 bounding boxes.
```

```
ydH.Coefficients % visualize the filter
```

References   Viola, Paul and Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. Volume: 1, pp. 511–518.

More About

Computing an Integral Image and Using it for Filtering with Box Filters

The `integralImage` function together with the `integralKernel` object and `integralFilter` function complete the workflow for box filtering based on integral images. You can use this workflow for filtering with box filters.

- Use the `integralImage` function to compute the integral images
- Use the `integralFilter` function for filtering
- Use the `integralKernel` object to define box filters

The `integralKernel` object allows you to transpose the filter. You can use this to aim a directional filter. For example, you can turn a horizontal edge detector into vertical edge detector.

See Also

`detectMSERFeatures` | `integralImage` | `integralFilter` | `detectSURFFeaturesSURFPoints` |

Related Examples

integralKernel.transpose

Purpose   Transpose filter

Syntax   intKernel = intKernel.transpose

Description   intKernel = intKernel.transpose transposes the integral kernel.
You can use this operation to change the direction of an oriented filter.

Example   **Construct Harr-like Wavelet Filters**

Construct Harr-like wavelet filters using the dot and transpose notation.

```
horiH = integralKernel([1 1 4 3; 1 4 4 3], [-1, 1]); % horizontal filter
vertH = horiH.'; % vertical filter; note use of the dot before '
```

Purpose   Pad or crop input image along its rows, columns, or both

Description   The `ImagePadder` object pads or crop input image along its rows, columns, or both.

Construction   `HIMPAD = vision.ImagePadder` returns an image padder System object, `HIMPAD`, that performs two-dimensional padding and/or cropping of an input image.

`HIMPAD = vision.ImagePadder(`*Name*`,`*Value*`)` returns an image padder object, `HIMPAD`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **Method**

How to pad input image

Specify how to pad the input image as `Constant` | `Replicate` | `Symmetric` | `Circular`. The default is `Constant`.

PaddingValueSource

How to specify pad value

Indicate how to specify the pad value as either `Property` | `Input port`. This property applies when you set the `Method` property to `Constant`. The default is `Property`.

PaddingValue

Pad value

Specify the constant scalar value with which to pad the image. This property applies when you set the `Method` property to `Constant` and the `PaddingValueSource` property to `Property`. The default is `0`. This property is tunable.

SizeMethod

How to specify output image size

Indicate how to pad the input image to obtain the output image by specifying `Pad size | Output size`. When this property is `Pad size`, the size of the padding in the vertical and horizontal directions are specified. When this property is `Output size`, the total number of output rows and output columns are specified. The default is `Pad size`.

RowPaddingLocation

Location at which to add rows

Specify the direction in which to add rows to as `Top | Bottom | Both top and bottom | None`. Set this property to `Top` to add additional rows to the top of the image, `Bottom` to add additional rows to the bottom of the image, `Both top and bottom` to add additional rows to the top and bottom of the image, and `None` to maintain the row size of the input image. The default is `Both top and bottom`.

NumPaddingRows

Number of rows to add

Specify the number of rows to be added to the top, bottom, or both sides of the input image as a scalar value. When the `RowPaddingLocation` property is `Both top and bottom`, this property can also be set to a two element vector, where the first element controls the number of rows the System object adds to the top of the image and the second element controls the number of rows the System object adds to the bottom of the image. This property applies when you set the `SizeMethod` property to `Pad size` and the `RowPaddingLocation` property is not set to `None`. The default is `[2 3]`.

NumOutputRowsSource

How to specify number of output rows

Indicate how to specify the number of output rows as `Property | Next power of two`. If this property is `Next power of two`, the System object adds rows to the input image until the number

of rows is equal to a power of two. This property applies when you set the SizeMethod property to Output size. The default is Property.

NumOutputRows

Total number of rows in output

Specify the total number of rows in the output as a scalar integer. If the specified number is smaller than the number of rows of the input image, then image is cropped. This property applies when you set the SizeMethod property to Output size and the NumOutputRowsSource property to Property. The default is 12.

ColumnPaddingLocation

Location at which to add columns

Specify the direction in which to add columns one of Left | Right | Both left and right | None. Set this property to Left to add additional columns on the left side of the image, Right to add additional columns on the right side of the image, Both left and right to add additional columns on the left and right side of the image, and None to maintain the column length of the input image. The default is Both left and right.

NumPaddingColumns

Number of columns to add

Specify the number of columns to be added to the left, right, or both sides of the input image as a scalar value. When the ColumnPaddingLocation property is Both left and right, this property can also be set to a two element vector, where the first element controls the number of columns the System object adds to the left side of the image and the second element controls the number of columns the System object adds to the right side of the image. This property applies when you set the SizeMethod property to Pad size and the NumPaddingColumns property is not set to None. The default is 2.

NumOutputColumnsSource

How to specify number of output columns

Indicate how to specify the number of output columns as `Property | Next power of two`. If you set this property to `Next power of two`, the System object adds columns to the input until the number of columns is equal to a power of two. This property applies when you set the `SizeMethod` property to `Output size`. The default is `Property`.

NumOutputColumns

Total number of columns in output

Specify the total number of columns in the output as a scalar integer. If the specified number is smaller than the number of columns of the input image, then image is cropped. This property applies when you set the `SizeMethod` property to `Output size` and the `NumOutputColumnsSource` property to `Property`. The default is `10`.

Methods		
	clone	Create image padder object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Perform two-dimensional padding or cropping of input

Examples   Pad two rows to the bottom, and three columns to the right of an image. Use the value of the last array element as the padding value.

```
himpad = vision.ImagePadder('Method', 'Replicate', ...
         'RowPaddingLocation', 'Bottom', ...
         'NumPaddingRows', 2, ...
         'ColumnPaddingLocation', 'Right', ...
         'NumPaddingColumns', 3);
x = [1 2;3 4];
y = step(himpad,x);
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Image Pad block reference page. The object properties correspond to the block parameters.

See Also   vision.GeometricScaler

vision.ImagePadder.clone

Purpose   Create image padder object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a ImagePadder System object *C*, with the same
                    property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.ImagePadder.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the ImagePadder
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.ImagePadder.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Perform two-dimensional padding or cropping of input

Syntax   Y = step(H,X)
               Y = step(H,X,PAD)

Description   Y = step(H,X) performs two-dimensional padding or cropping of input, *X*.

Y = step(H,X,PAD) performs two-dimensional padding and/or cropping of input, *X*, using the input pad value *PAD*. This applies when you set the Method property to Constant and the PaddingValueSource property to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.KalmanFilter

Purpose   Kalman filter for object tracking

Description   The Kalman filter object is designed for tracking. You can use it to predict a physical object's future location, to reduce noise in the detected location, or to help associate multiple physical objects with their corresponding tracks. A Kalman filter object can be configured for each physical object for multiple object tracking. To use the Kalman filter, the object must be moving at constant velocity or constant acceleration.

The Kalman filter algorithm involves two steps, prediction and correction (also known as the update step). The first step uses previous states to predict the current state. The second step uses the current measurement, such as object location, to correct the state. The Kalman filter implements a discrete time, linear State-Space System.

Note To make configuring a Kalman filter easier, you can use the `configureKalmanFilter` object to configure a Kalman filter. It sets up the filter for tracking a physical object in a Cartesian coordinate system, moving with constant velocity or constant acceleration. The statistics are the same along all dimensions. If you need to configure a Kalman filter with different assumptions, do not use the function, use this object directly.

Construction   `obj = vision.KalmanFilter` returns a Kalman filter object for a discrete time, constant velocity system. In this "State-Space System" on page 2-498, the state transition model, *A*, and the measurement model, *H*, are set as follows:

Variable	Value
A	[1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1]
H	[1 0 0 0; 0 1 0 0]

```
obj =
vision.KalmanFilter(StateTransitionModel,MeasurementModel)
```

configures the state transition model, *A*, and the measurement
model, *H*.

```
obj =
vision.KalmanFilter(StateTransitionModel,MeasurementModel,ControlModel
```
additionally configures the control model, *B*.

```
obj =
vision.KalmanFilter(StateTransitionModel,MeasurementModel,ControlModel
```
) configures the Kalman filter object properties, specified as one or
more `Name,Value` pair arguments. Unspecified properties
have default values.

To Track Objects:

Use the `predict` and `correct` methods based on detection results.

- When the tracked object is detected, use the `predict` and
  `correct` methods with the Kalman filter object and the detection
  measurement. Call the methods in the following order:

  ```
  [...] = predict(obj);
  [...] = correct(obj,measurement);
  ```

- When the tracked object is not detected, call the `predict` method,
  but not the `correct` method. When the tracked object is missing or
  occluded, no measurement is available. Set the methods up with
  the following logic:

  ```
  [...] = predict(obj);
  If measurement exists
   [...] = correct(obj,measurement);
  end
  ```

- If the tracked object becomes available after missing for t-1
  consecutive time steps, you can call the `predict` method with the
  time input syntax. This syntax is particularly useful when processing
  asynchronous video. For example,

  ```
  [...] = predict(obj,t);
  ```

```
[...] = correct(obj,measurement);
If t=1, predict(obj,t) is the same as predict(obj).
```

Use the distance method to find the best matches. The computed distance values describe how a set of measurements matches the Kalman filter. You can thus select a measurement that best fits the filter. This strategy can be used for matching object detections against object tracks in a multiobject tracking problem. This distance computation takes into account the covariance of the predicted state and the process noise. The distance method can only be called after the predict method.

Distance Equation

$$d(z) = (z - Hx)^T \sum{}^{-1} (z - Hx) + \ln|\Sigma|$$

Where $\Sigma = HPH^T + R$ and $|\Sigma|$ is the determinant of Σ. You can then find the best matches by examining the returned distance values.

d = distance(obj, z_matrix) computes a distance between the location of a detected object and the predicted location by the Kalman filter object. Each row of the *N*-column z_matrix input matrix contains a measurement vector. The distance method returns a row vector where each distance element corresponds to the measurement input.

State-Space System

This object implements a discrete time, linear state-space system, described by the following equations.

State equation:	$x(k) = Ax(k-1) + Bu(k-1) + w(k-1)$
Measurement equation:	$z(k) = Hx(k) + v(k)$

Variable Definition

Variable	Description	Dimension
k	Time.	Scalar
x	State. Gaussian vector with covariance P. $[x \sim N(\bar{x}, P)]$	M-element vector
P	State estimation error covariance.	M-by-M matrix
A	State transition model.	M-by-M matrix
B	Control model.	M-by-L matrix
u	Control input.	L-element vector
w	Process noise; Guassian vector with zero mean and covariance Q. $[w \sim N(0, Q)]$	M-element vector
Q	Process noise covariance.	M-by-M matrix
z	Measurement. For example, location of detected object.	N-element vector
H	Measurement model.	N-by-M matrix
v	Measurement noise; Gaussian vector with zero mean and covariance R. $[v \sim N(0, R)]$	N-element vector
R	Measurement noise covariance.	N-by-N matrix

Properties　　**StateTransitionModel**

Model describing state transition between time steps (A)

Specify the transition of state between times as an M-by-M matrix. After the object is constructed, this property cannot be changed. This property relates to the A variable in the "State-Space System" on page 2-498.

Default: [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1]

MeasurementModel

Model describing state to measurement transformation (*H*)

Specify the transition from state to measurement as an *N*-by-*M* matrix. After the object is constructed, this property cannot be changed. This property relates to the *H* variable in the "State-Space System" on page 2-498.

Default: [1 0 0 0; 0 1 0 0]

ControlModel

Model describing control input to state transformation (*B*)

Specify the transition from control input to state as an *M*-by-*L* matrix. After the object is constructed, this property cannot be changed. This property relates to the *B* variable in the "State-Space System" on page 2-498.

Default: []

State

State (*x*)

Specify the state as a scalar or an *M*-element vector. If you specify it as a scalar it will be extended to an *M*-element vector. This property relates to the *x* variable in the "State-Space System" on page 2-498.

Default: [0]

StateCovariance

State estimation error covariance (*P*)

Specify the covariance of the state estimation error as a scalar or an *M*-by-*M* matrix. If you specify it as a scalar it will be extended

to an *M*-by-*M* diagonal matrix. This property relates to the *P* variable in the "State-Space System" on page 2-498.

Default: [1]

ProcessNoise

Process noise covariance (*Q*)

Specify the covariance of process noise as a scalar or an *M*-by-*M* matrix. If you specify it as a scalar it will be extended to an *M*-by-*M* diagonal matrix. This property relates to the *Q* variable in the "State-Space System" on page 2-498.

Default: [1]

MeasurementNoise

Measurement noise covariance (*R*)

Specify the covariance of measurement noise as a scalar or an *N*-by-*N* matrix. If you specify it as a scalar it will be extended to an *N*-by-*N* diagonal matrix. This property relates to the *R* variable in the "State-Space System" on page 2-498.

Default: [1]

Methods

clone	Create Kalman filter object with same property values
correct	Correction of measurement, state, and state estimation error covariance
distance	Confidence value of measurement
predict	Prediction of measurement

Examples	**Track Location of an Object**

Track the location of a physical object moving in one direction.

Generate synthetic data which mimics the 1-D location of a physical object moving at a constant speed.

```
detectedLocations = num2cell(2*randn(1,40) + (1:40));
```

Simulate missing detections by setting some elements to empty.

```
detectedLocations{1} = [];
for idx = 16: 25
    detectedLocations{idx} = [];
end
```

Create a figure to show the location of detections and the results of using the Kalman filter for tracking.

```
figure; hold on;
ylabel('Location');
ylim([0,50]);
xlabel('Time');
xlim([0,length(detectedLocation)]);
```

Create a 1-D, constant speed Kalman filter when the physical object is first detected. Predict the location of the object based on previous states. If the object is detected at the current time step, use its location to correct the states.

```
kalman = [];
for idx = 1: length(detectedLocations)
   location = detectedLocations{idx};
   if isempty(kalman)
     if ~isempty(location)

       stateModel = [1 1; 0 1];
       measurementModel = [1 0];
       kalman = vision.KalmanFilter(stateModel, measurementModel, ...
```

```
        'ProcessNoise', 1e-4, 'MeasurementNoise', 4);
      kalman.State = [location, 0];
    end
  else
    trackedLocation = predict(kalman);
    if ~isempty(location)
      plot(idx, location, 'k+');
     d = distance(kalman, location);
      title(sprintf('Distance: %f', d));
      trackedLocation = correct(kalman, location);
    else
      title('Missing detection');
    end
    pause(0.2);
    plot(idx, trackedLocation, 'ro');
  end
 end
 legend('Detected locations', 'Predicted/corrected locations');
```

Remove Noise From a Signal

Use Kalman filter to remove noise from a random signal corrupted by a zero-mean Gaussian noise.

1

Synthesize a random signal that has value of 1 and is corrupted by a zero-mean Gaussian noise with standard deviation of 0.1.

```
x = 1;
len = 100;
z = x + 0.1 * randn(1, len);
```

2

Remove noise from the signal by using a Kalman filter. The state is expected to be constant, and the measurement is the same as state.

```
stateTransitionModel = 1;
measurementModel = 1;
obj = vision.KalmanFilter(stateTransitionModel,measurementModel,'StateCov

z_corr = zeros(1, len);
for idx = 1: len
  predict(obj);
  z_corr(idx) = correct(obj, z(idx));
end
```

3

Plot results

```
figure, plot(x * ones(1, len), 'g-'); hold on;
plot(1:len, z, 'b+', 1:len, z_corr, 'r-');
legend('Original signal', 'Noisy signal', 'Filtered signal');
```

References   Welch, Greg, and Gary Bishop, *An Introduction to the Kalman Filter*,
TR 95–041. University of North Carolina at Chapel Hill, Department of
Computer Science.

See Also   configureKalmanFilter | assignDetectionsToTracks

Purpose   Create Kalman filter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a KalmanFilter object *C*, with the same property
                 values as *H*. The clone method creates a new unlocked object.

vision.KalmanFilter.distance

Purpose   Confidence value of measurement

Syntax   d = distance(obj, z_matrix)

Description   d = distance(obj, z_matrix) computes a distance between the
location of a detected object and the predicted location by the Kalman
filter object. Each row of the *N*-column z_matrix input matrix contains
a measurement vector. The distance method returns a row vector where
each distance element corresponds to the measurement input.

This distance computation takes into account the covariance of the
predicted state and the process noise. The distance method can only be
called after the predict method.

Purpose   Correction of measurement, state, and state estimation error covariance

Syntax   [z_corr,x_corr,P_corr] = correct(obj,z)

Description [z_corr,x_corr,P_corr] = correct(obj,z) returns the correction
of measurement, state, and state estimation error covariance. The
correction is based on the current measurement *z*, an *N*-element vector.
The object overwrites the internal state and covariance of the Kalman
filter with corrected values.

vision.KalmanFilter.predict

Purpose   Prediction of measurement

Syntax
```
[z_pred, x_pred, P_pred] = predict(obj)
[z_pred, x_pred, P_pred] = predict(obj,t)
[z_pred, x_pred, P_pred] = predict(obj,t,u)
```

Description   `[z_pred, x_pred, P_pred] = predict(obj)` returns the prediction of measurement, state, and state estimation error covariance at the next time step (e.g., the next video frame). The object overwrites the internal state and covariance of the Kalman filter with the prediction results.

`[z_pred, x_pred, P_pred] = predict(obj,t)` additionally, lets you specify the time t, a positive integer. The returned values are the measurement, state, and state estimation error covariance predicted for t time steps (e.g., t video frames) later.

`[z_pred, x_pred, P_pred] = predict(obj,t,u)` additionally lets you specify the control input, u, an L-element vector. This syntax applies when you set the control model, B.

Purpose   Find local maxima in matrices

Description   The `LocalMaximaFinder` object finds local maxima in matrices.

After constructing the object and optionally setting properties, use the `step` method to find the coordinates of the local maxima in the input image. Use the `step` syntax below with input matrix, `I`, `LocalMaximaFinder` object, `H`, and any optional properties.

`IDX = step(H,I)` returns [x y] coordinates of the local maxima in an *M*-by-2 matrix, `IDX`. *M* represents the number of local maximas found. The maximum value of *M* may not exceed the value set in the `MaximumNumLocalMaxima` property.

`[...] = step(H,I,THRESH)` finds the local maxima in input image `I`, using the threshold value `THRESH`, when you set the `ThresholdSource` property to `Input port`.

Construction `H = vision.LocalMaximaFinder` returns a local maxima finder System object, `H`, that finds local maxima in input matrices.

`H = vision.LocalMaximaFinder(Name,Value)` returns a local maxima finder object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

`H = vision.LocalMaximaFinder(MAXNUM,NEIGHBORSIZE,Name,Value)` returns a local maxima finder object, `H`, with the `MaximumNumLocalMaxima` property set to MAXNUM, `NeighborhoodSize` property set to NEIGHBORSIZE, and other specified properties set to the specified values.

Properties   **MaximumNumLocalMaxima**

Maximum number of maxima to find

Specify the maximum number of maxima to find as a positive scalar integer value. The default is 2.

NeighborhoodSize

Neighborhood size for zero-ing out values

Specify the size of the neighborhood around the maxima, over which the System object zeros out values, as a 2-element vector of positive odd integers. The default is [5 7].

ThresholdSource

Source of threshold

Specify how to enter the threshold value as `Property`, or `Input port`. The default is `Property`.

Threshold

Value that all maxima should match or exceed

Specify the threshold value as a scalar of MATLAB built-in numeric data type. This property applies when you set the `ThresholdSource` property to `Property`. The default is `10`. This property is tunable.

HoughMatrixInput

Indicator of Hough Transform matrix input

Set this property to `true` if the input is antisymmetric about the

rho axis and the theta value ranges from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ radians, which correspond to a Hough matrix.

When you set this property to `true`, the object assumes a Hough matrix input . The block applies additional processing, specific to Hough transform on the right and left boundaries of the input matrix.

The default is `false`.

IndexDataType

Data type of index values

Specify the data type of index values as `double`, `single`, `uint8`, `uint16`, or `uint32`. The default is `uint32`.

Methods

clone	Create local maxima finder object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Find local maxima in input image

Examples   Find a local maxima in an input.

```
I = [0 0 0 0 0 0 0 0 0 0 0 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 3 5 7 5 3 1 0 0; ... % local max at x=7, y=4
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 0 0 0 0 0 0 0 0 0];

hLocalMax = vision.LocalMaximaFinder;
hLocalMax.MaximumNumLocalMaxima = 1;
hLocalMax.NeighborhoodSize = [3 3];
hLocalMax.Threshold = 1;

location = step(hLocalMax, I)
```

vision.LocalMaximaFinder

Algorithms   This object implements the algorithm, inputs, and outputs described on the Find Local Maxima block reference page. The object properties correspond to the block parameters.

See Also   `vision.HoughTransform` | `vision.HoughLines` | `vision.Maximum`

Purpose   Create local maxima finder object with same property values

Syntax   `C = clone(H)`

Description   `C = clone(H)` creates an `LocalMaximaFinder` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

vision.LocalMaximaFinder.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, *N* from the `step` method.

The `getNumOutputs` method returns a positive integer that is the number of outputs from the `step` method. This value will change if you alter any properties that turn outputs on or off.

vision.LocalMaximaFinder.isLocked

Purpose　　　　　Locked status for input attributes and nontunable properties

Syntax　　　　　　TF = isLocked(H)

Description　　　TF = isLocked(H) returns the locked status, TF of the
LocalMaximaFinder System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.LocalMaximaFinder.step

Purpose   Find local maxima in input image

Syntax   IDX = step(H,I)
                   [...] = step(H,I,THRESH)

Description   IDX = step(H,I) returns [x y] coordinates of the local maxima in an *M*-by-2 matrix, IDX, where *M* represents the number of local maximas found. The maximum value of *M* may not exceed the value set in the MaximumNumLocalMaxima property.

[...] = step(H,I,THRESH) finds the local maxima in input image I, using the threshold value THRESH, when you set the ThresholdSource property to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Draw markers on output image

Description   The MarkerInserter object can draw circle, x-mark, plus sign, star, or
                   rectangle markers in a 2-D grayscale or truecolor RGB image. The
                   output image can then be displayed or saved to a file.

Construction   markerInserter = vision.MarkerInserter returns a marker inserter
                   System object, markerInserter, that draws a circle in an image.

                   markerInserter = vision.MarkerInserter(*Name*,*Value*) returns
                   a marker inserter object, H, with each specified property set to the
                   specified value. You can specify additional name-value pair arguments
                   in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

To insert a marker:

1 Define and set up your marker inserter using the constructor.

2 Call the step method with the input image, I, the marker inserter
      object, markerInserter, points PTS, and any optional properties. See
      the syntax below for using the step method.

J = step(markerInserter,I,PTS) draws the marker specified by the
Shape property on input image *I*. The input PTS specify the coordinates
for the location of the marker. You can specify the PTS input as an
M-by-2 [*x y*] matrix of *M* number of markers. Each [*x y*] pair defines
the center location of the marker. The markers are embedded on the
output image J.

J = step(markerInserter,I,PTS,ROI) draws the specified marker in
a rectangular area defined by the ROI input. This applies when you set
the ROIInputPort property to true. The ROI input defines a rectangular
area as [*x y width height*], where [*x y*] determine the upper-left corner
location of the rectangle, and *width* and *height* specify the size.

J = step(markerInserter,I,PTS,...,CLR) draws the marker with
the border or fill color specified by the CLR input. This applies when you
set the BorderColorSource property or the FillColorSource property
to 'Input port'.

Properties   **Shape**

Shape of marker

Specify the type of marker to draw as `Circle` | `X-mark` | `Plus`,
`Star` | `Square`.

Default: `Circle`

Size

Size of marker

Specify the size of the marker, in pixels, as a scalar value greater
than or equal to `1`. This property is tunable.

Default: `3`

Fill

Enable marker fill

Set this property to true to fill the marker with an intensity value
or a color. This property applies when you set the `Shape` property
to `Circle` or `Square`.

Default: `false`

BorderColorSource

Border color source

Specify how the marker's border color is provided as `Input port`,
`Property`. This property applies either when you set the `Shape`
property to `X-mark`, `Plus`, or `Star`, or when you set the `Shape`
property to `Circle` or `Square`, and the `Fill` property to `false`.
When you set `BorderColorSource` to `Input port`, a border color
vector must be provided as an input to the System object's `step`
method.

Default: `Property`

BorderColor

Border color of marker

Specify the border color of the marker as `Black` | `White` | `Custom`. If this property is set to `Custom`, the `CustomBorderColor` property is used to specify the value. This property applies when the `BorderColorSource` property is enabled and set to `Property`.

Default:`Black`

CustomBorderColor

Intensity or color value for marker's border

Specify an intensity or color value for the marker's border. If the input is an intensity image, this property can be set to a scalar intensity value for one marker or R-element vector where R is the number of markers. If the input is a color image, this property can be set to a P-element vector where P is the number of color planes or a P-by-R matrix where P is the number of color planes and R is the number of markers. This property applies when you set the `BorderColor` property to `Custom`. This property is tunable when the `Antialiasing` property is `false`.

Default: `[200 120 50]`

FillColorSource

Source of fill color

Specify how the marker's fill color is provided as `Input port` | `Property`. This property applies when you set the `Shape` property to `Circle` or `Square`, and the `Fill` property to `true`. When this property is set to `Input port`, a fill color vector must be provided as an input to the System object's step method.

Default: `Property`

FillColor

Fill color of marker

Specify the color to fill the marker as `Black` | `White` | `Custom`. If this property is set to `Custom`, the `CustomFillColor` property is used to specify the value. This property applies when the `FillColorSource` property is enabled and set to `Property`.

Default: `Black`

CustomFillColor

Intensity or color value for marker's interior

Specify an intensity or color value to fill the marker. If the input is an intensity image, this property can be set to a scalar intensity value for one marker or R-element vector where R is the number of markers. If the input is a color image, this property can be set to a P-element vector where P is the number of color planes or a P-by-R matrix where P is the number of color planes and R is the number of markers. This property applies when you set the `FillColor` property to `Custom`. This property is tunable when the `Antialiasing` property is `false`.

Default: `[200 120 50]`

Opacity

Opacity of shading inside marker

Specify the opacity of the shading inside the marker by a scalar value between `0` and `1`, where `0` is transparent and `1` is opaque. This property applies when you set the `Fill` property to `true`. This property is tunable.

Default: `0.6`

ROIInputPort

Enable defining a region of interest to draw marker

Set this property to `true` to specify a region of interest (ROI) on the input image through an input to the `step` method. If the property is set to `false`, the object uses the entire image.

Default: `false`

Antialiasing

Enable performing smoothing algorithm on marker

Set this property to `true` to perform a smoothing algorithm on the marker. This property applies when you do not set the `Shape` property to `Square` or `Plus`.

Default: `false`

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`.

Default: `Floor`

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap | Saturate`. This property applies when you set the `Fill` property to true and/or the `Antialiasing` property to `true`.

Default: `Wrap`

OpacityDataType

Opacity word length

Specify the opacity fixed-point data type as `Same word length as input` or `Custom`. This property applies when you set the `Fill` property to `true`.

Default: `Custom`

CustomOpacityDataType

Opacity word length

Specify the opacity fixed-point type as an unscaled numerictype object with a Signedness of Auto. This property applies when you set the Fill property to true and the OpacityDataType property to Custom.

Default: numerictype([],16)

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as Same as first input or Custom. This property applies when you set the Fill property to true and/or the Antialiasing property to true.

Default: Custom

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the Fill property to true and/or the Antialiasing property to true, and the ProductDataType property to Custom.

Default: numerictype([],32,14)

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Same as product | Same as first input | Custom. This property applies when you set the Fill property to true and/or the Antialiasing property to true.

Default: Same as product

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you

set the `Fill` property to `true` and/or the `Antialiasing` property to `true`, and the `AccumulatorDataType` property to `Custom`.

Default: `numerictype([],32,14)`

Methods

clone	Create marker inserter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Draw specified marker on input image

Examples

Draw White Plus Signs in a Grayscale Image

```
markerInserter = vision.MarkerInserter('Shape','Plus','BorderColor','w
I = imread('cameraman.tif');
Pts = int32([10 10; 20 20; 30 30]);
J = step(markerInserter, I, Pts);
imshow(J);
```

Draw Red Circles in a Grayscale Image

```
red = uint8([255 0 0]);  % [R G B]; class of red must match class of ]
markerInserter = vision.MarkerInserter('Shape','Circle','BorderColor',
I = imread('cameraman.tif');
RGB = repmat(I,[1 1 3]); % convert the image to RGB
Pts = int32([60 60; 80 80; 100 100]);
J = step(markerInserter, RGB, Pts);
```

```
imshow(J);
```

Draw Blue X-marks in a Color Image

```
markerInserter = vision.MarkerInserter('Shape','X-mark','BorderColor','Cu
RGB = imread('autumn.tif');
Pts = int32([20 20; 40 40; 60 60]);
J = step(markerInserter, RGB, Pts);
imshow(J);
```

See Also   vision.ShapeInserter | vision.TextInserter

Purpose   Create marker inserter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an MarkerInserter System object *C*, with the
                   same property values as *H*. The clone method creates a new unlocked
                   object.

vision.MarkerInserter.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.MarkerInserter.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the MarkerInserter
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.MarkerInserter.step

Purpose   Draw specified marker on input image

Syntax   J = step(markerInserter,I,PTS)
                 J = step(markerInserter,I,PTS,ROI)
                 J = step(markerInserter,I,PTS,...,CLR)

Description J = step(markerInserter,I,PTS) draws the marker specified by the
                 Shape property on input image *I*. The input PTS specify the coordinates
                 for the location of the marker. You can specify the PTS input as an
                 M-by-2 [*x y*] matrix of *M* number of markers. Each [*x y*] pair defines
                 the center location of the marker. The markers are embedded on the
                 output image J.

                 J = step(markerInserter,I,PTS,ROI) draws the specified marker in
                 a rectangular area defined by the ROI input. This applies when you set
                 the ROIInputPort property to true. The ROI input defines a rectangular
                 area as [*x y width height*], where [*x y*] determine the upper-left corner
                 location of the rectangle, and *width* and *height* specify the size.

                 J = step(markerInserter,I,PTS,...,CLR) draws the marker with
                 the border or fill color specified by the CLR input. This applies when you
                 set the BorderColorSource property or the FillColorSource property
                 to 'Input port'.

                 Note The object performs an initialization the first time the step
                 method is executed. This initialization locks nontunable properties and
                 input specifications, such as dimensions, complexity, and data type
                 of the input data. If you change a nontunable property or an input
                 specification, the System object issues an error. To change nontunable
                 properties or inputs, you must first call the release method to unlock
                 the object.

                 ### Input Arguments

                 H

                 Shape inserter object with shape and properties specified.

I

Input image.

PTS

Input *M*-by-2 matrix of *M* number of [x y] coordinates describing the location for markers. This property must be an integer value. If you enter non-integer value, the object rounds it to the nearest integer.

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_M & y_M \end{bmatrix}$$

Each [x y] pair defines the center of a marker.

The table below shows the data types required for the inputs, I and PTS.

Input image I	Input matrix PTS
built-in integer	built-in or fixed-point integer
fixed-point integer	built-in or fixed-point integer
double	double, single, or build-in integer
single	double, single, or build-in integer

RGB

Scalar, vector, or matrix describing one plane of the RGB input video stream.

ROI

Input *4*-element vector of integers [x y width height], that define a rectangular area in which to draw the shapes. The first two elements represent the one-based coordinates of the upper-left

corner of the area. The second two elements represent the width and height of the area.

- Double-precision floating point

- Single-precision floating point

- 8-, 16-, and 32-bit signed integer

- 8-, 16-, and 32-bit unsigned integer

CLR

This port can be used to dynamically specify shape color.

P-element vector or an *M*-by-*P* matrix, where *M* is the number of shapes, and *P*, the number of color planes. You can specify a color (RGB), for each shape, or specify one color for all shapes. The data type for the CLR input must be the same as the input image.

Output Arguments

J

Output image. The markers are embedded on the output image.

Purpose   Find maximum values in input or sequence of inputs

Description   The Maximum object finds maximum values in an input or sequence of inputs.

Construction   H = vision.Maximum returns an object, H, that computes the value and index of the maximum elements in an input or a sequence of inputs.

H = vision.Maximum(*Name*,*Value*) returns a maximum-finding object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **ValueOutputPort**

Output maximum value

Set this property to true to output the maximum value of the input. This property applies when you set the RunningMaximum property to false.

Default:true.

RunningMaximum

Calculate over single input or multiple inputs

When you set this property to true, the object computes the maximum value over a sequence of inputs. When you set this property to false, the object computes the maximum value over the current input.

Default: false.

IndexOutputPort

Output the index of the maximum value

Set this property to true to output the index of the maximum value of the input. This property applies only when you set the RunningMaximum property to false.

Default:`true`.

ResetInputPort

Additional input to enable resetting of running maximum

Set this property to `true` to enable resetting of the running maximum. When you set this property to `true`, a reset input must be specified to the `step` method to reset the running maximum. This property applies only when you set the `RunningMaximum` property to `true`.

Default: `false`.

ResetCondition

Condition that triggers resetting of running maximum

Specify the event that resets the running maximum as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`.

Default: `Non-zero`.

IndexBase

Numbering base for index of maximum value

Specify the numbering used when computing the index of the maximum value as starting from either `One` or `Zero`. This property applies only when you set the `IndexOutputPort` property to true.

Default: `One`.

Dimension

Dimension to operate along

Specify how the maximum calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. This property applies only when you set the `RunningMaximum` property to `false`.

Default: `Column`.

CustomDimension

Numerical dimension to calculate over

Specify the integer dimension of the input signal over which the object finds the maximum. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`.

Default: `1`.

ROIProcessing

Enable region-of-interest processing

Set this property to `true` to enable calculation of the maximum value within a particular region of an image. This property applies when you set the `Dimension` property to `All` and the `RunningMaximum` property to `false`.

Default:`false`.

ROIForm

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies only when you set the `ROIProcessing` property to `true`.

Default: `Rectangles`.

ROIPortion

Calculate over entire ROI or just perimeter

Specify whether to calculate the maximum over the `Entire ROI` or the `ROI perimeter`. This property applies only when you set the `ROIForm` property to `Rectangles`.

Default: `Entire ROI`.

ROIStatistics

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`.

ValidityOutputPort

Output flag indicating if any part of ROI is outside input image

Set this property to `true` to return the validity of the specified ROI as completely or partially inside of the image. This applies when you set the `ROIForm` property to `Lines` or `Rectangles`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `Label matrix`.

Default: `false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero`.

Default: `Floor`.

OverflowAction

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`.

Default: `Wrap`.

ProductDataType

Data type of product

Specify the product fixed-point data type as `Same as input` or `Custom`.

Default: Same as input.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object. This property applies only when you set the AccumulatorDataType property to Custom.

Default: numerictype(true,32,30).

AccumulatorDataType

Data type of accumulator

Specify the accumulator fixed-point data type as Same as product, Same as input, or Custom.

Default:Same as product.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object. This property applies only when you set the AccumulatorDataType property to Custom.

Default: numerictype(true,32,30).

Methods

clone	Create maximum object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

vision.Maximum

release	Allow property value and input characteristics changes
reset	Reset computation of running maximum
step	Compute maximum value

Examples   Determine the maximum value and its index in a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hmax = vision.Maximum;
[m, ind] = step(hmax, img);
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the 2-D Maximum block reference page.

See Also   vision.Mean | vision.Minimum

Purpose　　　　Create maximum object with same property values

Syntax　　　　C = clone(H)

Description　　C = clone(H) creates a Maximum object *C*, with the same property values as *H*. The clone method creates a new unlocked object with uninitialized states.

vision.Maximum.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.Maximum.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the Maximum System
object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Reset computation of running maximum

Syntax   reset(H)

Description   reset(H) resets the computation of the running maximum for the
                   Maximum object H.

vision.Maximum.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Compute maximum value

Syntax   ```
[VAL,IND] = step(H,X)
VAL = step(H,X)
IND = step(H,X)
VAL = step(H,X,R)
[...] = step(H,I,ROI)
[...] = step(H,I,LABEL,LABELNUMBERS)
[...,FLAG] = step(H,I,ROI)
[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)
```

Description   [VAL,IND] = step(H,X) returns the maximum value, VAL, and the
index or position of the maximum value, IND, along a dimension of X
specified by the value of the Dimension property.

VAL = step(H,X) returns the maximum value, VAL, of the input X.
When the RunningMaximum property is true, VAL corresponds to the
maximum value over a sequence of inputs.

IND = step(H,X) returns the zero- or one-based index IND of
the maximum value. To enable this type of processing, set the
IndexOutputPort property to true and the ValueOutputPort and
RunningMaximum properties to false.

VAL = step(H,X,R) computes the maximum value, VAL, over a
sequence of inputs, and resets the state of H based on the value of
reset signal, R, and the ResetCondition property. To enable this
type of processing, set the RunningMaximum property to true and the
ResetInputPort property to true.

[...] = step(H,I,ROI) computes the maximum of an input image,
I, within the given region of interest, ROI. To enable this type of
processing, set the ROIProcessing property to true and the ROIForm
property to Lines, Rectangles or Binary mask.

[...] = step(H,I,LABEL,LABELNUMBERS) computes the maximum of
an input image, I, for a region whose labels are specified in the vector
LABELNUMBERS. To enable this type of processing, set the ROIProcessing
property to true and the ROIForm property to Label matrix.

[...,FLAG] = step(H,I,ROI) also returns FLAG, indicating whether the given region of interest is within the image bounds. To enable this type of processing, set the ROIProcessing and ValidityOutputPort properties to true and the ROIForm property to Lines, Rectangles or Binary mask.

[...,FLAG] = step(H,I,LABEL,LABELNUMBERS) also returns FLAG, indicating whether the input label numbers are valid. To enable this type of processing, set the ROIProcessing and ValidityOutputPort properties to true and the ROIForm property to Label matrix.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Find mean value of input or sequence of inputs

Description   The Mean object finds the mean of an input or sequence of inputs.

Construction   H = vision.Mean returns an object, H, that computes the mean of an input or a sequence of inputs.

H = vision.Mean(*Name*,*Value*) returns a mean-finding object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **RunningMean**

Calculate over single input or multiple inputs

When you set this property to true, the object calculates the mean over a sequence of inputs. When you set this property to false, the object computes the mean over the current input. The default is false.

ResetInputPort

Additional input to enable resetting of running mean

Set this property to true to enable resetting of the running mean. When you set this property to true, a reset input must be specified to the step method to reset the running mean. This property applies only when you set the RunningMean property to true. The default is false.

ResetCondition

Condition that triggers resetting of running mean

Specify the event that resets the running mean as Rising edge, Falling edge, Either edge, or Non-zero. This property applies only when you set the ResetInputPort property to true. The default is Non-zero.

Dimension

Dimension to operate along

Specify how the mean calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. This property applies only when you set the `RunningMean` property to `false`. The default is `All`.

CustomDimension

Numerical dimension to calculate over

Specify the integer dimension, indexed from one, of the input signal over which the object calculates the mean. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`. The default is `1`.

ROIProcessing

Enable region-of-interest processing

Set this property to `true` to enable calculation of the mean within a particular region of an image. This property applies when you set the `Dimension` property to `All` and the `RunningMean` property to `false`. The default is `false`.

ROIForm

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies only when you set the `ROIProcessing` property to `true`. The default is `Rectangles`.

ROIPortion

Calculate over entire ROI or just perimeter

Specify whether to calculate the mean over the `Entire ROI` or the `ROI perimeter`. This property applies only when you set the `ROIForm` property to `Rectangles`. The default is `Entire ROI`.

ROIStatistics

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`. The default is `Individual statistics for each ROI`.

ValidityOutputPort

Output flag indicating if any part of ROI is outside input image

Set this property to `true` to return the validity of the specified ROI as completely or partially inside of the image. This applies when you set the `ROIForm` property to `Lines` or `Rectangles`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `Label matrix`.

The default is `false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

AccumulatorDataType

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as input`, or `Custom`. The default is `Same as input`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object. This property applies only when you set the AccumulatorDataType property to Custom. The default is numerictype(true,32,30).

OutputDataType

Data type of output

Specify the output fixed-point data type as Same as accumulator, Same as input, or Custom. The default is Same as accumulator.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled numerictype object. This property applies only when you set the OutputDataType property to Custom. The default is numerictype(true,32,30).

Methods

clone	Create mean object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset computation of running mean
step	Compute mean of input

Examples

Determine the mean of a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hmean = vision.Mean;
m = step(hmean,img);
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the 2-D Mean block reference page.

See Also

vision.Maximum | vision.Minimum

vision.Mean.clone

Purpose   Create mean object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Mean object *C*, with the same property values as *H*. The clone method creates a new unlocked object with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns the number of expected inputs, *N* to
the `step` method.

The `getNumInputs` method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the `step` method.
This value will change if you alter any properties that turn inputs on or
off. You must call the `step` method with the number of input arguments
equal to the result of `getNumInputs(`*H*`)`.

vision.Mean.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the Mean System
object..

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.Mean.reset

Purpose   Reset computation of running mean

Syntax   reset(H)

Description   reset(H) resets the computation of the running mean for the Mean
                   object H.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.Mean.step

Purpose   Compute mean of input

Syntax
```
Y = step(H,X)
Y = step(H,X,R)
Y = step(H,X,ROI)
Y = step(H,X,LABEL,LABELNUMBERS)
[Y,FLAG] = step(H,X,ROI)
[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description   Y = step(H,X) computes the mean of input image elements *X*. When you set the RunningMean property to true, the output *Y* corresponds to the mean of the input elements over successive calls to the step method.

Y = step(H,X,R) computes the mean value of the input image elements *X* over successive calls to the step method, and optionally resets the computation of the running mean based on the value of reset signal, *R* and the value of the ResetCondition property. To enable this type of processing, set the RunningMean property to true and the ResetInputPort property to true.

Y = step(H,X,ROI) computes the mean of input image elements *X* within the given region of interest specified by the input *ROI*. To enable this type of processing, set the ROIProcessing property to true and the ROIForm property to Lines, Rectangles or Binary mask.

Y = step(H,X,LABEL,LABELNUMBERS) computes the mean of the input image elements, *X*, for the region whose labels are specified in the vector *LABELNUMBERS*. The regions are defined and labeled in the matrix *LABEL*. To enable this type of processing, set the ROIProcessing property to true and the ROIForm property to Label matrix.

[Y,FLAG] = step(H,X,ROI) also returns the output *FLAG*, indicating whether the given region of interest *ROI*, is within the image bounds. To enable this type of processing, set the the ROIProcessing and ValidityOutputPort properties to true and the ROIForm property to Lines, Rectangles or Binary mask.

[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS) also returns the output *FLAG* which indicates whether the input label numbers are

valid. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Label matrix`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.Median

Purpose   Find median values in an input

Description   The Median object finds median values in an input.

Construction H = vision.Median returns a System object, H, that computes the median of the input or a sequence of inputs.

H = vision.Median(*Name*,*Value*) returns a median System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **SortMethod**

Sort method

Specify the sort method used for calculating the median as Quick sort or Insertion sort.

Dimension

Dimension with which to operate along

Specify how the calculation is performed over the data as All, Row, Column, or Custom. The default is All

CustomDimension

Numerical dimension over which to calculate

Specify the integer dimension of the input signal over which the object calculates the mean. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the Dimension property to Custom. The default is 1.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling | Convergent | Floor | Nearest | Round | Simplest | Zero`. The default is `Floor`.

OverflowAction

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

ProductDataType

Product output word and fraction lengths

Specify the product output fixed-point data type as `Same as input` or `Custom`. The default is `Same as input`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object. This property applies when you set the `ProductDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

AccumulatorDataType

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as input`, or `Custom`. The default is `Same as input`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

OutputDataType

Data type of output

Specify the output fixed-point data type as `Same as accumulator`, `Same as input`, or `Custom`. The default is `Same as accumulator`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object. This property applies only when you set the `OutputDataType` property to `Custom`. The default is `numerictype(true,32,30)`.

Methods	clone	Create median object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	step	Compute median of input

Examples Determine the median in a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hmdn = vision.Median;
med = step(hmdn,img);
```

Algorithms This object implements the algorithm, inputs, and outputs described on the 2-D Median block reference page. The object properties correspond to the block parameters.

Purpose   Create median object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates an Median System object *C*, with the same
property values as *H*. The clone method creates a new unlocked object.

vision.Median.getNumInputs

Purpose	Number of expected inputs to step method
Syntax	N = getNumInputs(H)
Description	N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.
	The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.Median.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the Median System
object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.Median.step

Purpose   Compute median of input

Syntax   Y = step(H,X)

Description   Y = step(H,X) computes median of input *X*.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

Purpose   2D median filtering

Description   The `MedianFilter` object performs 2D median filtering.

Construction   `H = vision.MedianFilter` returns a 2D median filtering object, `H`. This object performs two-dimensional median filtering of an input matrix.

`H = vision.MedianFilter(`*Name*`,`*Value*`)` returns a median filter object, `H`, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

`H = vision.MedianFilter(`*SIZE*`,`*Name*`,`*Value*`)` returns a median filter object, `H`, with the NeighborhoodSize property set to *SIZE* and other properties set to the specified values.

Properties   **NeighborhoodSize**

Size of neighborhood to compute the median

Specify the size of the neighborhood over which the median filter computes the median. When you set this property to a positive integer, the integer represents the number of rows and columns in a square matrix. When you set this property to a two-element vector, the vector represents the number of rows and columns in a rectangular matrix. The median filter does not support fixed-point properties when both neighborhood dimensions are odd. The default is [3 3].

OutputSize

Output matrix size

Specify the output size as one of `Same as input size` | `Valid`. The default is `Same as input size`. When you set this property to `Valid`, the 2D median filter only computes the median where the neighborhood fits entirely within the input image and requires no padding. In this case, the dimensions of the output image are:

output rows = input rows - neighborhood rows + 1

output columns = input columns - neighborhood columns + 1

Otherwise, the object outputs the same dimensions as the input image.

PaddingMethod

Input matrix boundary padding method

Specifies how to pad the boundary of the input matrix as one of `Constant` | `Replicate` | `Symmetric` | `Circular`. When you set this property to `Constant`, the object pads the matrix with a constant value. When you set this property to `Replicate`, the object pads the input matrix by repeating its border values. When you set this property to `Symmetric`, the object pads the input matrix with its mirror image. When you set this property to `Circular` the object pads the input matrix using a circular repetition of its elements. This property applies only when you set the `OutputSize` property to `Same as input size`. The default is `Constant`.

PaddingValueSource

Source of constant boundary value

Specifies how to define the constant boundary value as one of `Property` | `Input port`. This property applies only when you set the `PaddingMethod` property to `Constant`. The default is `Property`.

PaddingValue

Constant padding value for input matrix

Specifies a constant value to pad the input matrix. The default is `0`. This property applies only when you set the `PaddingMethod` property to `Constant` and the `PaddingValueSource` property to `Property`. This property is tunable.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`. This property applies only when the `NeighborhoodSize` property corresponds to even neighborhood options.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Wrap`. This property is applies only when the `NeighborhoodSize` property corresponds to even neighborhood options.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as input` | `Custom`. The default is `Same as input`. This property applies only when the `NeighborhoodSize` property corresponds to even neighborhood options.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `AccumulatorDataType` property is `Custom` and the `NeighborhoodSize` property corresponds to even neighborhood options. The default is `numerictype([],32,30)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input`or `Custom`. This property applies only when the `NeighborhoodSize` property corresponds to even neighborhood options. The default is `Same as input`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom` and the `NeighborhoodSize` property corresponds to even neighborhood options. The default is `numerictype([],16,15)`.

Methods

clone	Create 2D median filter with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Perform median filtering on input image

Examples

Perform Median Filtering on an Image with Additive Salt and Pepper Noise

Read the image and add noise.

```
I = imread('pout.tif');
I = imnoise(I, 'salt & pepper')
```

Create a median filter object.

```
medianFilter = vision.MedianFilter([5 5]);
```

Filter the image.

```
I_filtered = step(medianFilter, I);
```

Display the results with a title.

```
figure; imshowpair(I, I_filtered, 'montage');
title('Noisy image on the left and filtered image on the right');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Median Filter block reference page. The object properties correspond to the block parameters.

See Also   vision.ImageFilter

vision.MedianFilter.clone

Purpose   Create 2D median filter with same property values

Syntax   `C = clone(H)`

Description   `C = clone(H)` creates a `MedianFilter` System object *C*, with the same property values as *H*. The `clone` method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.MedianFilter.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the MedianFilter
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.MedianFilter.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles
or hardware connections) and allows all properties and input
characteristics to be changed.

> **Note** You can use the release method on a System object in code
> generated from MATLAB, but once you release its resources, you cannot
> use that System object again.

Purpose   Perform median filtering on input image

Syntax   I2 = step(H,I1)
                   I2 = step(H,I1,PVAL)

Description   I2 = step(H,I1) performs median filtering on the input image I1
                   and returns the filtered image I2.

                   I2 = step(H,I1,PVAL) performs median filtering on input image I1,
                   using PVAL for the padding value. This option applies when you set
                   the OutputSize property to Same as input size, the PaddingChoice
                   property to Constant, and the PaddingValueSource property to Input
                   port.

                   Note The object performs an initialization the first time the step
                   method is executed. This initialization locks nontunable properties and
                   input specifications, such as dimensions, complexity, and data type
                   of the input data. If you change a nontunable property or an input
                   specification, the System object issues an error. To change nontunable
                   properties or inputs, you must first call the release method to unlock
                   the object.

vision.Minimum

Purpose   Find minimum values in input or sequence of inputs

Description   The `Minimum` object finds minimum values in an input or sequence of inputs.

Construction   `H = vision.Minimum` returns an object, H, that computes the value and index of the minimum elements in an input or a sequence of inputs.

`H = vision.Minimum(`*Name*`,`*Value*`)` returns a minimum-finding object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **ValueOutputPort**

Output minimum value

Set this property to `true` to output the minimum value of the input. This property applies when you set the `RunningMinimum` property to `false`.

Default: `true`

RunningMinimum

Calculate over single input or multiple inputs

When you set this property to `true`, the object computes the minimum value over a sequence of inputs. When you set this property to `false`, the object computes the minimum value over the current input.

Default: `false`

IndexOutputPort

Output the index of the minimum value

Set this property to `true` to output the index of the minimum value of the input. This property applies only when you set the `RunningMinimum` property to `false`.

Default: `true`

ResetInputPort

Additional input to enable resetting of running minimum

Set this property to `true` to enable resetting of the running minimum. When you set this property to `true`, a reset input must be specified to the `step` method to reset the running minimum. This property applies only when you set the `RunningMinimum` property to `true`.

Default: `false`

ResetCondition

Condition that triggers resetting of running minimum

Specify the event that resets the running minimum as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`.

Default: `Non-zero`

Dimension

Dimension to operate along

Specify how the minimum calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. This property applies only when you set the `RunningMinimum` property to `false`.

Default:`Column`.

CustomDimension

Numerical dimension to calculate over

Specify the integer dimension of the input signal over which the object finds the minimum. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`.

Default:`1`.

ROIProcessing

Enable region-of-interest processing

Set this property to `true` to enable calculation of the minimum value within a particular region of an image. This property applies when you set the `Dimension` property to `All` and the `RunningMinimum` property to `false`.

Default: `false`.

ROIForm

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies only when you set the `ROIProcessing` property to `true`.

Default: `Rectangles`.

ROIPortion

Calculate over entire ROI or just perimeter

Specify whether to calculate the minimum over the `Entire ROI` or the `ROI perimeter`. This property applies only when you set the `ROIForm` property to `Rectangles`.

Default: `Entire ROI`.

ROIStatistics

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`.

ValidityOutputPort

Output flag indicating if any part of ROI is outside input image

Set this property to `true` to return the validity of the specified ROI as completely or partially inside of the image. This applies when you set the `ROIForm` property to `Lines` or `Rectangles`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `Label matrix`.

Default: `false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`.

Default: `Floor`.

OverflowAction

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`.

Default: `Wrap`.

ProductDataType

Data type of product

Specify the product fixed-point data type as `Same as input` or `Custom`.

Default:`Same as input`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `Custom`.

Default:numerictype(true,32,30).

AccumulatorDataType

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as product`, `Same as input`, or `Custom`.

Default: `Same as product`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `Custom`.

Default:numerictype(true,32,30).

Methods

clone	Create minimum object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
reset	Reset computation of running minimum
step	Compute minimum value

Examples Determine the minimum value and its index in a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hmax = vision.Minimum;
[m, ind] = step(hmax, img);
```

Algorithms This object implements the algorithm, inputs, and outputs described on the 2-D Minimum block reference page.

See Also vision.Maximum | | vision.Mean

vision.Minimum.clone

Purpose   Create minimum object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Minimum System object *C*, with the same property values as *H*. The clone method creates a new unlocked object with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

vision.Minimum.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the Minimum System
                   object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

vision.Minimum.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Reset computation of running minimum

Syntax   reset(H)

Description   reset(H) resets computation of the running minimum for the Minimum
                   object H.

vision.Minimum.step

Purpose   Compute minimum value

Syntax
```
[VAL,IND] = step(H,X)
VAL = step(H,X)
IND = step(H,X)
VAL = step(H,X,R)
[...] = step(H,I,ROI)
[...] = step(H,I,LABEL,LABELNUMBERS)
[...,FLAG] = step(H,I,ROI)
[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)
```

Description   `[VAL,IND] = step(H,X)` returns the minimum value, VAL, and the index or position of the minimum value, IND, along a dimension of X specified by the value of the `Dimension` property.

`VAL = step(H,X)` returns the minimum value, VAL, of the input X. When the `RunningMinimum` property is `true`, VAL corresponds to the minimum value over a sequence of inputs.

`IND = step(H,X)` returns the zero- or one-based index IND of the minimum value. To enable this type of processing, set the `IndexOutputPort` property to `true` and the `ValueOutputPort` and `RunningMinimum` properties to `false`.

`VAL = step(H,X,R)` computes the minimum value, VAL, over a sequence of inputs, and resets the state of H based on the value of reset signal, R, and the `ResetCondition` property. To enable this type of processing, set the `RunningMinimum` property to `true` and the `ResetInputPort` property to `true`.

`[...] = step(H,I,ROI)` computes the minimum of an input image, I, within the given region of interest, ROI. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[...] = step(H,I,LABEL,LABELNUMBERS)` computes the minimum of an input image, I, for a region whose labels are specified in the vector LABELNUMBERS. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Label matrix`.

[...,FLAG] = step(H,I,ROI) also returns FLAG, indicating whether
the given region of interest is within the image bounds. To enable this
type of processing, set the ROIProcessing and ValidityOutputPort
properties to true and the ROIForm property to Lines, Rectangles
or Binary mask.

[...,FLAG] = step(H,I,LABEL,LABELNUMBERS) also returns FLAG,
indicating whether the input label numbers are valid. To enable this
type of processing, set the ROIProcessing and ValidityOutputPort
properties to true and the ROIForm property to Label matrix.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

---

vision.MorphologicalBottomHat

Purpose　　　Bottom-hat filtering on image

Description　　The `MorphologicalBottomHat` object performs bottom-hat filtering on an intensity or binary image. Bottom-hat filtering is the equivalent of subtracting the input image from the result of performing a morphological closing operation on the input image. The bottom-hat filtering object uses flat structuring elements only.

Construction　　`H = vision.MorphologicalBottomHat` returns a bottom-hat filtering object, `H`, that performs bottom-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element.

`H = vision.MorphologicalBottomHat(`*Name,Value*`)` returns a bottom-hat filtering object, `H`, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties　　**ImageType**

Specify type of input image or video stream

Specify the type of the input image as `Intensity` or `Binary`. The default is `Intensity`.

NeighborhoodSource

Source of neighborhood values

Specify how to enter neighborhood or structuring element values as one of `Property` or `Input port`. If set to `Property`, use the `Neighborhood` property to specify the neighborhood or structuring element values. Otherwise, specify the neighborhood using an input to the `step` method. Note that you can specify structuring elements only by using the `Neighborhood` property. You can not specify structuring elements as inputs to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

This property applies only when you set the `NeighborhoodSource` property to `Property`. If specifying a neighborhood, this property must be a matrix or vector of 1s and 0s. If specifying a structuring element, use the `strel` function. The default is `strel('octagon',15)`.

Methods

clone	Create morphological bottom hat filter with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Perform bottom-hat filtering on input image

Examples

Perform bottom-hat filtering on an image.

```
I = im2single(imread('blobs.png'));
hbot = vision.MorphologicalBottomHat('Neighborhood',strel('disk', 5))
J = step(hbot,I);
figure;
subplot(1,2,1),imshow(I); title('Original image');
subplot(1,2,2),imshow(J);
title('Bottom-hat filtered image');
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the Bottom-hat block reference page. The object properties correspond to the block parameters.

vision.MorphologicalBottomHat

Purpose   Create morphological bottom hat filter with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a MorphologicalBottomHat System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.MorphologicalBottomHat.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                     method.

                     The getNumOutputs method returns a positive integer that is the
                     number of outputs from the step method. This value will change if you
                     alter any properties that turn outputs on or off.

vision.MorphologicalBottomHat.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
MorphologicalBottomHat System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.MorphologicalBottomHat.step

Purpose   Perform bottom-hat filtering on input image

Syntax   Y = step(H,I)
               Y = step(H,I,NHOOD)

Description Y = step(H,I) performs bottom-hat filtering on the input image, *I*, and returns the filtered image *Y*.

Y = step(H,I,NHOOD) performs bottom-hat filtering on the input image, *I* using *NHOOD* as the neighborhood when you set the NeighborhoodSource property to Input port. The object returns the filtered image in the output *Y*.

Purpose   Perform morphological closing on image

Description   The `MorphologicalClose` object performs morphological closing on an intensity or binary image. The MorphologicalClose System object performs a dilation operation followed by an erosion operation using a predefined neighborhood or structuring element. This System object uses flat structuring elements only.

Construction   `H = vision.MorphologicalClose` returns a System object, `H`, that performs morphological closing on an intensity or binary image.

`H = vision.MorphologicalClose(Name,Value)` returns a morphological closing System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **NeighborhoodSource**

Source of neighborhood values

Specify how to enter neighborhood or structuring element values as `Property` or `Input port`. If set to `Property`, use the `Neighborhood` property to specify the neighborhood or structuring element values. Otherwise, specify the neighborhood using an input to the `step` method. Note that structuring elements can only be specified using `Neighborhood` property and they cannot be used as input to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

This property is applicable when the `NeighborhoodSource` property is set to `Property`. If you are specifying a neighborhood, this property must be a matrix or vector of `1s` and `0s`. If you are specifying a structuring element, use the `strel` function. The default is `strel('line',5,45)`.

vision.MorphologicalClose

Methods	clone	Create morphological close object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	step	Perform morphological closing on input image

Examples
Perform morphological closing on an image.

```
img = im2single(imread('blobs.png'));
hclosing = vision.MorphologicalClose;
hclosing.Neighborhood = strel('disk', 10);
closed = step(hclosing, img);
figure;

subplot(1,2,1),imshow(img); title('Original image');
subplot(1,2,2),imshow(closed); title('Closed image');
```

Algorithms
This object implements the algorithm, inputs, and outputs described on the Closing block reference page. The object properties correspond to the block parameters.

See Also
vision.MorphologicalOpen | vision.ConnectedComponentLabeler | vision.Autothresholder | strel

Purpose   Create morphological close object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an MorphologicalClose System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.MorphologicalClose.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
                 method.

                 The getNumOutputs method returns a positive integer that is the
                 number of outputs from the step method. This value will change if you
                 alter any properties that turn outputs on or off.

vision.MorphologicalClose.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
MorphologicalClose System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose	Perform morphological closing on input image
Syntax	IC = step(H,I) IC = step(H,I,NHOOD)
Description	IC = step(H,I) performs morphological closing on input image *I*.

IC = step(H,I,NHOOD) performs morphological closing on input image *I* using the input *NHOOD* as the neighborhood when you set the NeighborhoodSource property to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Perform morphological dilation on an image

Description   The `MorphologicalDilate` object performs morphological dilation on an image.

Construction   `H = vision.MorphologicalDilate` returns a System object, `H`, that performs morphological dilation on an intensity or binary image.

`H = vision.MorphologicalDilate(`*Name*`,`*Value*`)` returns a morphological dilation System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **NeighborhoodSource**

Source of neighborhood values

Specify how to enter neighborhood or structuring element values as `Property` or `Input port`. If set to `Property`, use the `Neighborhood` property to specify the neighborhood or structuring element values. Otherwise, specify the neighborhood using an input to the `step` method. Note that structuring elements can only be specified using `Neighborhood` property and they cannot be used as input to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

This property is applicable when the `NeighborhoodSource` property is set to `Property`. If you are specifying a neighborhood, this property must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the `strel` function. The default is [1 1; 1 1].

vision.MorphologicalDilate

Methods	clone	Create morphological dilate object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	step	Perform morphological dilate on input

Examples   Fuse fine discontinuities on images.

```
x = imread('peppers.png');
hcsc = vision.ColorSpaceConverter;
hcsc.Conversion = 'RGB to intensity';
hautothresh = vision.Autothresholder;
hdilate = vision.MorphologicalDilate('Neighborhood', ones(5,5));
x1 = step(hcsc, x);
x2 = step(hautothresh, x1);
y = step(hdilate, x2);
figure;

subplot(3,1,1),imshow(x); title('Original image');
subplot(3,1,2),imshow(x2); title('Thresholded Image');
subplot(3,1,3),imshow(y); title('Dilated Image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Dilation block reference page. The object properties correspond to the block parameters.

See Also   vision.MorphologicalErode | vision.MorphologicalOpen |
                    vision.MorphologicalClose | strel

vision.MorphologicalDilate.clone

Purpose   Create morphological dilate object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a MorphologicalDilate System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                     the step method.

                     The getNumInputs method returns a positive integer that is the number
                     of expected inputs (not counting the object itself) to the step method.
                     This value will change if you alter any properties that turn inputs on or
                     off. You must call the step method with the number of input arguments
                     equal to the result of getNumInputs(*H*).

vision.MorphologicalDilate.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the `step` method.

The `getNumOutputs` method returns a positive integer that is the number of outputs from the `step` method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                   MorphologicalDilate System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

vision.MorphologicalDilate.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose　　Perform morphological dilate on input

Syntax　　　`ID = step(H,I)`
　　　　　　　`ID = step(H,I,NHOOD)`

Description　`ID = step(H,I)` performs morphological dilation on input image `I` and returns the dilated image `IE`.

`ID = step(H,I,NHOOD)` uses input `NHOOD` as the neighborhood when the `NeighborhoodSource` property is set to `Input port`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.MorphologicalErode

Purpose   Perform morphological erosion on an image

Description   The `MorphologicalErode` object performs morphological erosion on an image using a neighborhood specified by a square structuring element of width `4`.

Construction   `H = vision.MorphologicalErode` returns a System object, `H`, that performs morphological erosion on an intensity or binary image.

`H = vision.MorphologicalErode(`*Name*,*Value*`)` returns a morphological erosion System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **NeighborhoodSource**

Source of neighborhood values

Specify how to enter neighborhood or structuring element values as `Property` or `Input port`. If set to `Property`, use the `Neighborhood` property to specify the neighborhood or structuring element values. Otherwise, specify the neighborhood using an input to the `step` method. Note that structuring elements can only be specified using `Neighborhood` property and they cannot be used as input to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

This property is applicable when the `NeighborhoodSource` property is set to `Property`. If you are specifying a neighborhood, this property must be a matrix or vector of 1s and 0s. If you are specifying a structuring element, use the `strel` function. The default is `strel('square',4)`.

Methods

clone	Create morphological erode object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Perform morphological erosion on input

Examples   Erode an input image.

```
x = imread('peppers.png');
hcsc = vision.ColorSpaceConverter;
hcsc.Conversion = 'RGB to intensity';
hautothresh = vision.Autothresholder;
herode = vision.MorphologicalErode('Neighborhood', ones(5,5));
x1 = step(hcsc, x); % convert input to intensity
x2 = step(hautothresh, x1); % convert input to binary
y = step(herode, x2); % Perform erosion on input
figure;

subplot(3,1,1),imshow(x); title('Original image');
subplot(3,1,2),imshow(x2); title('Thresholded Image');
subplot(3,1,3),imshow(y); title('Eroded Image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Erosion block reference page. The object properties correspond to the block parameters.

vision.MorphologicalErode

See Also   vision.MorphologicalDilate | vision.MorphologicalOpen | vision.MorphologicalClose | strel

Purpose   Create morphological erode object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an MorphologicalErode System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.MorphologicalErode.getNumInputs

Purpose　　　Number of expected inputs to step method

Syntax　　　`N = getNumInputs(H)`

Description　　`N = getNumInputs(H)` returns the number of expected inputs, *N* to the `step` method.

The `getNumInputs` method returns a positive integer that is the number of expected inputs (not counting the object itself) to the `step` method. This value will change if you alter any properties that turn inputs on or off. You must call the `step` method with the number of input arguments equal to the result of `getNumInputs(H)`.

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                     method.

                     The getNumOutputs method returns a positive integer that is the
                     number of outputs from the step method. This value will change if you
                     alter any properties that turn outputs on or off.

vision.MorphologicalErode.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
MorphologicalErode System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.MorphologicalErode.step

Purpose   Perform morphological erosion on input

Syntax   IE = step(H,I)
                IE = step(H,I,NHOOD)

Description IE = step(H,I) performs morphological erosion on input image I and
                returns the eroded image IE.

                IE = step(H,I,NHOOD) uses input NHOOD as the neighborhood when the
                NeighborhoodSource property is set to Input port.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

Purpose   Perform morphological opening on an image

Description   The `MorphologicalOpen` object performs morphological opening
on an image. The `MorphologicalOpen` System object performs an
erosion operation followed by a dilation operation using a predefined
neighborhood or structuring element. This System object uses flat
structuring elements only. For more information about structuring
elements, see the `strel` function reference page in the Image Processing
Toolbox documentation.

Construction   `H = vision.MorphologicalOpen` returns a System object, `H`, that
performs morphological opening on an intensity or binary image.

`H = vision.MorphologicalOpen(`*Name*`,`*Value*`)` returns a
morphological opening System object, `H`, with each specified property
set to the specified value. You can specify additional name-value pair
arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **NeighborhoodSource**

Source of neighborhood values

Specify how to enter neighborhood or structuring element
values as `Property` or `Input port`. If set to `Property`, use the
`Neighborhood` property to specify the neighborhood or structuring
element values. Otherwise, specify the neighborhood using an
input to the `step` method. Note that structuring elements can
only be specified using `Neighborhood` property and they cannot be
used as input to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

This property applies when you set the `NeighborhoodSource`
property to `Property`. If you are specifying a neighborhood, this
property must be a matrix or vector of 1s and 0s. If you are
specifying a structuring element, use the `strel` function. The
default is `strel('disk',5)`.

vision.MorphologicalOpen

Methods	clone	Create morphological open object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	step	Perform morphological opening on input image

Examples   Perform opening on an image

```
img = im2single(imread('blobs.png'));
hopening = vision.MorphologicalOpen;
hopening.Neighborhood = strel('disk', 5);
opened = step(hopening, img);
figure;

subplot(1,2,1),imshow(img); title('Original image');
subplot(1,2,2),imshow(opened); title('Opened image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Opening block reference page. The object properties correspond to the block parameters.

See Also   vision.MorphologicalClose | vision.ConnectedComponentLabeler | vision.Autothresholder | strel

Purpose   Create morphological open object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates an MorphologicalOpen System object *C*, with the
same property values as *H*. The clone method creates a new unlocked
object.

vision.MorphologicalOpen.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.MorphologicalOpen.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
MorphologicalOpen System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.MorphologicalOpen.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.MorphologicalOpen.step

Purpose	Perform morphological opening on input image
Syntax	IO = step(H,I) IO = step(H,I,NHOOD)
Description	IO = step(H,I) performs morphological opening on binary or intensity input image I.

IO = step(H,I,NHOOD) uses input NHOOD as the neighborhood when the NeighborhoodSource property is set to Input port.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Top-hat filtering on image

Description The `MorphologicalTopHat` object performs top-hat filtering on an intensity or binary image. Top-hat filtering is the equivalent of subtracting the result of performing a morphological opening operation on the input image from the input image itself. This top–hat filtering object uses flat structuring elements only.

Construction H = vision.MorphologicalTopHat returns a top–hat filtering object, H, that performs top-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element.

H = vision.MorphologicalTopHat(*Name*,*Value*) returns a top-hat filtering object, H, with each property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties   **ImageType**

Specify type of input image or video stream

Specify the type of input image or video stream as `Intensity` or `Binary`. The default is `Intensity`.

NeighborhoodSource

Source of neighborhood values

Specify how to enter neighborhood or structuring element values as `Property` or `Input port`. If set to `Property`, use the `Neighborhood` property to specify the neighborhood or structuring element values. Otherwise, specify the neighborhood using an input to the `step` method. Note that structuring elements can only be specified using the `Neighborhood` property. You cannot use the structuring elements as an input to the `step` method. The default is `Property`.

Neighborhood

Neighborhood or structuring element values

vision.MorphologicalTopHat

This property applies only when you set the `NeighborhoodSource` property to `Property`. If specifying a neighborhood, this property must be a matrix or vector of `1`s and `0`s. If specifying a structuring element, use the `strel` function. The default is `strel('square',4)`.

Methods

clone	Create morphological top hat filter with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Perform top-hat filtering on input image

Examples

Perform top-hat filtering to correct uneven illumination.

```
I = im2single(imread('rice.png'));
htop = vision.MorphologicalTopHat('Neighborhood',strel('disk', 12));

% Improve contrast of output image
  hc = vision.ContrastAdjuster;  J = step(htop,I);
 J = step(hc,J);
 figure;

subplot(1,2,1),imshow(I); title('Original image');
subplot(1,2,2),imshow(J);
title('Top-hat filtered image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Top-hat block reference page. The object properties correspond to the block parameters.

See Also   strel | vision.MorphologicalBottomHat | vision.MorphologicalOpen

vision.MorphologicalTopHat.clone

Purpose   Create morphological top hat filter with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a MorphologicalTopHat System object C, with
the same property values as H. The clone method creates a new
unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

vision.MorphologicalTopHat.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
                 MorphologicalTopHat System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.MorphologicalTopHat.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
                   or hardware connections) and allows all properties and input
                   characteristics to be changed.

> **Note** You can use the release method on a System object in code
> generated from MATLAB, but once you release its resources, you cannot
> use that System object again.

Purpose   Perform top-hat filtering on input image

Syntax   Y = step(H,I)
                Y = step(H,I,NHOOD)

Description Y = step(H,I) top-hat filters the input image, *I* , and returns the
                filtered image *Y*.

                Y = step(H,I,NHOOD) filters the input image, *I* using the input *NHOOD*
                as the neighborhood when you set the NeighborhoodSource property to
                Input port. The object returns the filtered image in the output *Y*.

                Note The object performs an initialization the first time the step
                method is executed. This initialization locks nontunable properties and
                input specifications, such as dimensions, complexity, and data type
                of the input data. If you change a nontunable property or an input
                specification, the System object issues an error. To change nontunable
                properties or inputs, you must first call the release method to unlock
                the object.

                ---

vision.VideoFileReader

Purpose   Read video frames and audio samples from video file

Description   The VideoFileReader object reads video frames, images, and audio samples from a video file. The object can also read image files.

Supported Platforms and File Types

The supported file formats available to you depend on the codecs installed on your system.

Platform	Supported File Name Extensions
All Platforms	AVI (.avi)
Windows	**Image**: .jpg,.bmp
	Video: MPEG (.mpeg) MPEG-2 (.mp2) MPEG-1 .mpg MPEG-4, including H.264 encoded video (.mp4, .m4v) Windows Media Video (.wmv,.asf, .asx, .asx) and any format supported by Microsoft DirectShow® 9.0 or higher.
	Audio: WAVE (.wav) Windows Media Audio File (.wma) Audio Interchange File Format (.aif, .aiff) Compressed Audio Interchange File Format(.aifc), MP3 (.mp3) Sun Audio (.au) Apple (.snd)

Platform	Supported File Name Extensions
Macintosh	**Video**: `.avi` Motion JPEG 2000 (`.mj2`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (`.mov`) and any format supported by QuickTime as listed on http://support.apple.com/kb/HT3775.
	Audio: Uncompressed `.avi`
Linux	Any format supported by your installed plug-ins for GStreamer 0.10 or above, as listed on http://gstreamer.freedesktop.org/documentation/plugins.html, including Ogg Theora (`.ogg`).

Windows XP and Windows 7 x64 platform ships with a limited set of 64-bit video and audio codecs. If a compressed multimedia file fails to play, try one of the two alternatives:

- Run the 32-bit version of MATLAB on your Windows XP x64 platform. Windows XP x64 ships with many 32-bit codecs.

- Save the multimedia file to a supported file format listed in the table above.

If you use Windows, use Windows Media player Version 11 or later.

Construction   videoFReader = vision.VideoFileReader(FILENAME) returns a video file reader System object, videoFReader. The object can sequentially read video frames and/or audio samples from the input video file, FILENAME. Every call to the step method returns the next video frame.

videoFReader = vision.VideoFileReader(Filename,*Name*,*Value*) configures the video file reader properties, specified as one or more name-value pair arguments. Unspecified properties have default values.

vision.VideoFileReader

To read a file:

1 Define and set up your video file reader object using the constructor.

2 Call the `step` method with the input filename, `FILENAME`, the video file reader object, `videoFReader`, and any optional properties. See the syntax below for using the `step` method.

`I = step(videoFReader)` outputs next video frame.

`[Y,Cb,Cr] = step(videoFReader)` outputs the next frame of YCbCr 4:2:2 format video in the color components Y, Cb, and Cr. This syntax requires that you set the `ImageColorSpace` property to 'YCbCr 4:2:2'.

`[I,AUDIO] = step(videoFReader)` outputs the next video frame, `I`, and one frame of audio samples, `AUDIO`. This syntax requires that you set the `AudioOutputPort` property to `true`.

`[Y,Cb,Cr,AUDIO] = step(videoFReader)` outputs next frame of YCbCr 4:2:2 video in the color components Y, Cb, and Cr, and one frame of audio samples in `AUDIO`. This syntax requires that you set the `AudioOutputPort` property to `true`, and the `ImageColorSpace` property to 'YCbCr 4:2:2'.

`[..., EOF] = step(videoFReader)` returns the end-of-file indicator, `EOF`. The object sets `EOF` to `true` each time the output contains the last audio sample and/or video frame.

Properties

Filename

Name of video file

Specify the name of the video file as a string. The full path for the file needs to be specified only if the file is not on the MATLAB path.

Default: `vipmen.avi`

PlayCount

Number of times to play file

Specify a positive integer or inf to represent the number of times to play the file.

Default: inf

AudioOutputPort

Output audio data

Use this property to control the audio output from the video file reader. This property applies only when the file contains both audio and video streams.

Default: false

ImageColorSpace

RGB, YCbCr, or intensity video

Specify whether you want the video file reader object to output RGB, YCbCr 4:2:2 or intensity video frames. This property applies only when the file contains video. This property can be set to RGB, Intensity, or YCbCr 4:2:2.

Default: RGB

VideoOutputDataType

Output video data type

Set the data type of the video data output from the video file reader object. This property applies if the file contains video. This property can be set to double, single, int8, uint8, int16, uint16, int32, or Inherit.

Default: single

AudioOutputDataType

Output audio samples data type

Set the data type of the audio data output from the video file reader object. This property applies only if the file contains audio. This property can be set to double, single, int16, or uint8.

vision.VideoFileReader

Default: int16

Methods

clone	Create multimedia file reader object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
info	Information about specified video file
isDone	End-of-file status (logical)
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
reset	Reset internal states of multimedia file reader to read from beginning of file
step	Output video frame

Examples **Read and Play a Video File**

```
videoFReader = vision.VideoFileReader('viplanedeparture.avi');
videoPlayer = vision.VideoPlayer;
while ~isDone(videoFReader)
  videoFrame = step(videoFReader);
  step(videoPlayer, videoFrame);
end
release(videoPlayer);
release(videoFReader);
```

See Also vision.VideoFileWriter | vision.VideoPlayer | implay

Purpose   Create multimedia file reader object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a VideoFileReader System object C, with the
same property values as H. The clone method creates a new unlocked
object with uninitialized states.

vision.VideoFileReader.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                    method.

                    The getNumOutputs method returns a positive integer that is the
                    number of outputs from the step method. This value will change if you
                    alter any properties that turn outputs on or off.

vision.VideoFileReader.info

Purpose   Information about specified video file

Syntax   S = info(H)

Description   S = info(H) returns a MATLAB structure, S , with information about the video file specified in the Filename property. The fields and possible values for the structure S are described below:

Audio	Logical value indicating if the file has audio content.
Video	Logical value indicating if the file has video content.
VideoFrameRate	Frame rate of the video stream in frames per second. The value may vary from the actual frame rate of the recorded video, and takes into consideration any synchronization issues between audio and video streams when the file contains both audio and video content. This implies that video frames may be dropped if the audio stream leads the video stream by more than 1/(actual video frames per second).
VideoSize	Video size as a two-element numeric vector of the form: [VideoWidthInPixels, VideoHeightInPixels]
VideoFormat	Video signal format.

Purpose   End-of-file status (logical)

Syntax   TF = isDone(H)

Description   TF = isDone(H) returns a logical value, STATUS , indicating if the
VideoFileReader System object, H , has reached the end of the
multimedia file. STATUS remains true when you set the PlayCount
property to a value greater than 1.

vision.VideoFileReader.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
VideoFileReader System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

> **Note** You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

vision.VideoFileReader.reset

Purpose   Reset internal states of multimedia file reader to read from beginning of file

Syntax   reset(H)

Description reset(H) resets the VideoFileReader object to read from the beginning of the file.

Purpose   Output video frame

Syntax   I = step(H)
                [Y,Cb,Cr] = step(H)
                [I,AUDIO] = step(H)
                [Y,Cb,Cr,AUDIO] = step(H)
                [..., EOF] = step(H)

Description Use the step method with the video file reader object and any optional properties to output the next video frame.

I = step(H) outputs next video frame.

[Y,Cb,Cr] = step(H) outputs next frame of YCbCr 4:2:2 video in the color components Y, Cb, and Cr. This syntax requires that you set the ImageColorSpace property to 'YCbCr 4:2:2'.

[I,AUDIO] = step(H) outputs next video frame, I, and one frame of audio samples, AUDIO. This syntax requires that you set the AudioOutputPort property to true.

[Y,Cb,Cr,AUDIO] = step(H) outputs next frame of YCbCr 4:2:2 video in the color components Y, Cb, and Cr, and one frame of audio samples in AUDIO. This syntax requires that you set the AudioOutputPort property to true, and the ImageColorSpace property to 'YCbCr 4:2:2'.

[..., EOF] = step(H) returns the end-of-file indicator, EOF. The object sets EOF to true each time the output contains the last audio sample and/or video frame.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.VideoFileWriter

Purpose   Write video frames and audio samples to video file

Description   The `VideoFileWriter` object writes video frames and audio samples to
                     a video file. It can write video and/or audio to an AVI or WMV file.
                     WMV files can be written only on Windows. The video and audio can be
                     compressed. The available compression types depend on the encoders
                     installed on the platform.

Note This block supports code generation for platforms that have file
I/O available. You cannot use this block with Real-Time Windows
Target software, because that product does not support file I/O.

This object performs best on platforms with Version 11 or later
of Windows Media Player software. This object supports only
uncompressed RGB24 AVI files on Linux and Mac platforms.

Windows 7 UAC (User Account Control), may require administrative
privileges to encode WMV and WMA files.

The generated code for this object relies on prebuilt library files. You
can run this code outside the MATLAB environment, or redeploy it,
but be sure to account for these extra library files when doing so. The
packNGo function creates a single zip file containing all of the pieces
required to run or rebuild this code. See `packNGo` for more information.

To run an executable file that was generated from an object, you may
need to add precompiled shared library files to your system path. See
"MATLAB Coder" and "Shared Library Dependencies" for details.

This object allows you to write WMA/WMV streams to disk or across
a network connection. Similarly, the `vision.VideoFileReader` object
allows you to read WMA/WMV streams to disk or across a network
connection. If you want to play an MP3/MP4 file, but you do not have
the codecs, you can re-encode the file as WMA/WMV, which is supported
by the Computer Vision System Toolbox.

Construction   videoFWriter = vision.VideoFileWriter returns a video file
writer System object, videoFWriter. It writes video frames to an
uncompressed 'output.avi' video file. Every call to the step method
writes a video frame.

videoFWriter = vision.VideoFileWriter(FILENAME) returns a video
file writer object, videoFWriter that writes video to a file, FILENAME.
The file type can be AVI or WMV, specified by the FileFormat property.

videoFWriter = vision.VideoFileWriter(...,'*Name*',*Value*)
configures the video file writer properties, specified as one or more
name-value pair arguments. Unspecified properties have default values.

To write to a file:

1 Define and set up your video file writer object using the constructor.

2 Call the step method with the optional output filename, FILENAME,
the video file writer object, videoFWriter, and any optional
properties. See the syntax below for using the step method.

step(videoFWriter,I) writes one frame of video, I, to the output file.
The input video can be an *M*-by-*N*-by-3 truecolor RGB video frame, or
an *M*-by-*N* grayscale video frame.

step(videoFWriter,Y,Cb,Cr) writes one frame of YCbCr 4:2:2 video.
The width of Cb and Cr color components must be half of the width of
Y. You must set the value of the FileColorSpace property to 'YCbCr
4:2:2'.

step(videoFWriter,I,AUDIO) writes one frame of the input video, I,
and one frame of audio samples, AUDIO, to the output file. This applies
when you set the AudioInputPort property to true.

step(videoFWriter,Y,Cb,Cr,AUDIO) writes one frame of YCbCr
4:2:2 video, and one frame of audio samples, AUDIO, to the output
file. This applies when you set the AudioInputPort to true and the
FileColorSpace property to 'YCbCr 4:2:2'. The width of Cb and Cr
color components must be half of the width of Y.

vision.VideoFileWriter

Properties   **Filename**

Video output file name

Specify the name of the video file as a string.

Default: `output.avi`

FileFormat

Output file format

Specify the format of the file that is created. On Windows platforms, this may be `AVI` or `WMV`. On other platforms, this value is restricted to `AVI`. These abbreviations correspond to the following file formats:

> `WMV`: Windows Media Video
> `AVI`: Audio-Video Interleave

Default: `AVI`

AudioInputPort

Write audio data

Use this property to control whether the object writes audio samples to the video file. Set this value to `true` to write audio data.

Default: `false`

FrameRate

Video frame rate

Specify the frame rate of the video data in frames per second as a positive numeric scalar.

For videos which also contain audio data, the rate of the audio data will be determined as the rate of the video multiplied by the number of audio samples passed in each invocation of the `step` method. For example, if you use a frame rate of `30`, and pass

1470 audio samples to the `step` method, the object sets the audio sample to `44100`, ($1470 \times 30 = 44100$).

Default: `30`

AudioCompressor

Audio compression encoder

Specify the type of compression algorithm to implement for audio data. This compression reduces the size of the video file. Choose `None (uncompressed)` to save uncompressed audio data to the video file. The other options reflect the available audio compression algorithms installed on your system. This property applies only when writing `AVI` files on Windows platforms.

VideoCompressor

Video compression encoder

Specify the type of compression algorithm to use to compress the video data. This compression reduces the size of the video file. Choose `None (uncompressed)` to save uncompressed video data to the video file. The `VideoCompressor` property can also be set to one of the compressors available on your system. To obtain a list of available video compressors, you can use tab completion. Follow these steps:

1 Instantiate the object:

```
y = vision.VideoFileWriter
```

2 To launch the tab completion functionality, type the following up to the open quote.

```
y.VideoCompressor='
```

A list of compressors available on your system will appear after you press the `Tab` key. For example:

```
>>
>>
>>
>>
>>
>> y.VideoCompressor='
```

DV Video Encoder
MJPEG Compressor
None (uncompressed)
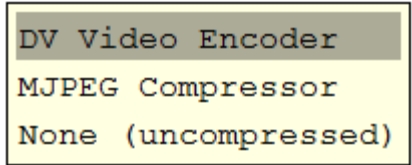
This property applies only when writing AVI files on Windows platforms.

AudioDataType

Uncompressed audio data type

Specify the compressed output audio data type. This property only applies when you write uncompressed WAV files.

FileColorSpace

Color space for output file

Specify the color space of AVI files as RGB or YCbCr 4:2:2. This property applies when you set the FileFormat property to AVI and only on Windows platforms.

Default: RGB

Methods

Examples

Write a Video to an AVI file

Set up the reader and writer objects

```
videoFReader = vision.VideoFileReader('viplanedeparture.avi');
videoFWriter = vision.VideoFileWriter('myFile.avi','FrameRate',videoFReac
```

Write the first 50 frames of the viplanedeparture.avi file into a newly created file, myFile.avi.

```
for i=1:50
    videoFrame = step(videoFReader);
    step(videoFWriter, videoFrame);
end
```

Close the input and output files.

```
release(videoFReader);
release(videoFWriter);
```

See Also   vision.VideoFileReader | vision.VideoPlayer | mplay

vision.VideoFileWriter.clone

Purpose   Create video file writer object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an VideoFileWriter System object C, with the same property values as H. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

vision.VideoFileWriter.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the
                  VideoFileWriter System object.

                  The isLocked method returns a logical value that indicates whether
                  input attributes and nontunable properties for the object are locked. The
                  object performs an internal initialization the first time the step method
                  is executed. This initialization locks nontunable properties and input
                  specifications, such as dimensions, complexity, and data type of the
                  input data. After locking, the isLocked method returns a true value.

vision.VideoFileWriter.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Write input video data to file

Syntax   step(videoFWriter,I)
                 step(videoFWriter,Y,Cb,Cr)
                 step(videoFWriter,I,AUDIO)
                 step(videoFWriter,Y,Cb,Cr,AUDIO)

Description   step(videoFWriter,I) writes one frame of the input video, I, to the
                 output file. The input video can be an *M*-by-*N*-by-3 truecolor RGB video
                 frame, or an *M*-by-*N* grayscale video frame where *M*-by-*N* represents
                 the size of the image.

                 step(videoFWriter,Y,Cb,Cr) writes one frame of YCbCr 4:2:2 video.
                 The width of Cb and Cr color components must be half of the width of
                 Y. You must set the value of the FileColorSpace property to 'YCbCr
                 4:2:2'.

                 step(videoFWriter,I,AUDIO) writes one frame of the input video, I,
                 and one frame of audio samples, AUDIO, to the output file. This applies
                 when you set the AudioInputPort property to true.

                 step(videoFWriter,Y,Cb,Cr,AUDIO) writes one frame of YCbCr
                 4:2:2 video, and one frame of audio samples, AUDIO, to the output
                 file. This applies when you set the AudioInputPort to true and the
                 FileColorSpace property to 'YCbCr 4:2:2'. The width of Cb and Cr
                 color components must be half of the width of Y.

                 Note The object performs an initialization the first time the step
                 method is executed. This initialization locks nontunable properties and
                 input specifications, such as dimensions, complexity, and data type
                 of the input data. If you change a nontunable property or an input
                 specification, the System object issues an error. To change nontunable
                 properties or inputs, you must first call the release method to unlock
                 the object.

vision.OpticalFlow

Purpose Estimate object velocities

Description The `OpticalFlow` System object estimates object velocities from one image or video frame to another. It uses either the Horn-Schunck or the Lucas-Kanade method.

Construction `opticalFlow = vision.OpticalFlow` returns an optical flow System object, `opticalFlow`. This object estimates the direction and speed of object motion from one image to another or from one video frame to another.

`opticalFlow = vision.OpticalFlow(Name,Value)` returns an optical flow System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

To estimate velocity:

1 Define and set up your text inserter using the constructor.

2 Call the `step` method with the input image, `I`, the optical flow object, `opticalFlow`, and any optional properties. See the syntax below for using the `step` method.

`VSQ = step(opticalFlow,I)` computes the optical flow of input image, `I`, from one video frame to another, and returns `VSQ`, specified as a matrix of velocity magnitudes.

`V = step(opticalFlow,I)` computes the optical flow of input image. `I`, from one video frame to another, and returns `V`, specified as a complex matrix of horizontal and vertical components. This applies when you set the `OutputValue` property to 'Horizontal and vertical components in complex form'.

`[...] = step(opticalFlow,I1,I2)` computes the optical flow of the input image I1, using I2 as a reference frame. This applies when you set the `ReferenceFrameSource` property to 'Input port'.

[..., IMV] = step(opticalFlow,I) outputs the delayed input image, IMV. The delay is equal to the latency introduced by the computation of the motion vectors. This property applies when you set the Method property to 'Lucas-Kanade', the TemporalGradientFilter property to 'Derivative of Gaussian', and the MotionVectorImageOutputport property to true.

Properties

Method

Optical flow computation algorithm

Specify the algorithm to compute the optical flow as one of Horn-Schunck | Lucas-Kanade.

Default: Horn-Schunck

ReferenceFrameSource

Source of reference frame for optical flow calculation

Specify computing optical flow as one of Property | Input port. When you set this property to Property, you can use the ReferenceFrameDelay property to determine a previous frame with which to compare. When you set this property to Input port, supply an input image for comparison.

This property applies when you set the Method property to Horn-Schunck. This property also applies when you set the Method property to Lucas-Kanade and the TemporalGradientFilter property to Difference filter [-1 1].

Default: Property

ReferenceFrameDelay

Number of frames between reference frame and current frame

Specify the number of frames between the reference and current frame as a positive scalar integer. This property applies when you set the ReferenceFrameSource property to Current frame and N-th frame back.

Default: 1

Smoothness

Expected smoothness of optical flow

Specify the smoothness factor as a positive scalar number. If the relative motion between the two images or video frames is large, specify a large positive scalar value. If the relative motion is small, specify a small positive scalar value. This property applies when you set the `Method` property to `Horn-Schunck`. This property is tunable.

Default: `1`

IterationTerminationCondition

Condition to stop iterative solution computation

Specify when the optical flow iterative solution stops. Specify as one of `Maximum iteration count` | `Velocity difference threshold` | `Either` . This property applies when you set the `Method` property to `Horn-Schunck`.

Default: `Maximum iteration count`

MaximumIterationCount

Maximum number of iterations to perform

Specify the maximum number of iterations to perform in the optical flow iterative solution computation as a positive scalar integer. This property applies when you set the `Method` property to `Horn-Schunck` and the `IterationTerminationCondition` property to either `Maximum iteration count` or `Either`. This property is tunable.

Default: `10`

VelocityDifferenceThreshold

Velocity difference threshold

Specify the velocity difference threshold to stop the optical flow iterative solution computation as a positive scalar number. This property applies when you set the `Method` property to

Horn-Schunck and the `IterationTerminationCondition` property to either `Maximum iteration count` or `Either`. This property is tunable.

Default: `eps`

OutputValue

Form of velocity output

Specify the velocity output as one of `Magnitude-squared` | `Horizontal and vertical components in complex form`.

Default: `Magnitude-squared`

TemporalGradientFilter

Temporal gradient filter used by Lucas-Kanade algorithm

Specify the temporal gradient filter used by the Lucas-Kanade algorithm as one of `Difference filter [-1 1]` | `Derivative of Gaussian`. This property applies when you set the `Method` property to `Lucas-Kanade`.

Default: `Difference filter [-1 1]`

BufferedFramesCount

Number of frames to buffer for temporal smoothing

Specify the number of frames to buffer for temporal smoothing as an odd integer from `3` to `31`, both inclusive. This property determines characteristics such as the standard deviation and the number of filter coefficients of the Gaussian filter used to perform temporal filtering. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Derivative of Gaussian`.

Default: `3`

ImageSmoothingFilterStandardDeviation

Standard deviation for image smoothing filter

Specify the standard deviation for the Gaussian filter used to smooth the image using spatial filtering. Use a positive scalar number. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Derivative of Gaussian`.

Default: `1.5`

GradientSmoothingFilterStandardDeviation

Standard deviation for gradient smoothing filter

Specify the standard deviation for the filter used to smooth the spatiotemporal image gradient components. Use a positive scalar number. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Derivative of Gaussian`.

Default: `1`

DiscardIllConditionedEstimates

Discard normal flow estimates when constraint equation is ill-conditioned

When the optical flow constraint equation is ill conditioned, set this property to `true` so that the motion vector is set to `0`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Derivative of Gaussian`. This property is tunable.

Default: `false`

MotionVectorImageOutputport

Return image corresponding to motion vectors

Set this property to `true` to output the image that corresponds to the motion vector being output by the System object. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Derivative of Gaussian`.

Default: `false`

NoiseReductionThreshold

Threshold for noise reduction

Specify the motion threshold between each image or video frame as a positive scalar number. The higher the number, the less small movements impact the optical flow calculation. This property applies when you set the `Method` property to `Lucas-Kanade`. This property is tunable.

Default: `0.0039`

Fixed-Point Properties

RoundingMethod

Rounding mode for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

Default: `Nearest`

OverflowAction

Overflow mode for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

Default: `Saturate`

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Custom`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a signed, scaled `numerictype` object. You can apply this property when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`. This property applies when you set the `ProductDataType` property to `Custom`.

Default: `numerictype(true,32,20)`

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product` | `Custom`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

Default: `Same as product`

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled `numerictype` object. You can apply this property when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`. This property applies when you set the `AccumulatorDataType` property to `Custom`.

Default: `numerictype(true,32,20)`

GradientDataType

Gradients word and fraction lengths

Specify the gradient components fixed-point data type as one of `Same as accumulator` | `Same as accumulator` | `Same as product` | `Custom`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

Default: `Same as accumulator`

CustomGradientDataType

Gradients word and fraction lengths

Specify the gradient components fixed-point type as a signed, scaled `numerictype` System object. You can apply this property when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`. This property applies when you set the `GradientDataType` property to `Custom`.

The default is `numerictype(true,32,20)`.

Default: `1`

ThresholdDataType

Threshold word and fraction lengths

Specify the threshold fixed-point data type as one of `Same word length as first input` | `Custom`. This property applies when you set the `Method` property to `Lucas-Kanade` and the `TemporalGradientFilter` property to `Difference filter [-1 1]`.

Default: `Same word length as first input`

CustomThresholdDataType

Threshold word and fraction lengths

Specify the threshold fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. You can apply this property when you set the `Method` property to `Lucas-Kanade` and the

TemporalGradientFilter property to Difference filter [-1 1]. This property applies when you set the ThresholdMode property to Custom.

Default: numerictype([],16,12)

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as Custom. This property applies when you set the Method property to Lucas-Kanade and the TemporalGradientFilter property to Difference filter [-1 1].

CustomOutputDataType

Output word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. The numerictype object should be unsigned if you set the OutputValue property to Magnitude-squared. It should be signed if set to Horizontal and vertical components in complex form. You can apply this property when you set the Method property to Lucas-Kanade and the TemporalGradientFilter property to Difference filter [-1 1]. This property applies when you set the OutputDataType property to Custom.

Default: numerictype(false,32,20)

Methods		
clone	Create optical flow object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Estimate direction and speed of object motion between video frames

Examples

Track Cars Using Optical Flow

Set up objects.

```
videoReader = vision.VideoFileReader('viptraffic.avi','ImageColorSpace
converter = vision.ImageDataTypeConverter;
opticalFlow = vision.OpticalFlow('ReferenceFrameDelay', 1);
opticalFlow.OutputValue = 'Horizontal and vertical components in compl
shapeInserter = vision.ShapeInserter('Shape','Lines','BorderColor','Cu
videoPlayer = vision.VideoPlayer('Name','Motion Vector');
```

Convert the image to single precision, then compute optical flow for
the video. Generate coordinate points and draw lines to indicate flow.
Display results.

```
while ~isDone(videoReader)
    frame = step(videoReader);
    im = step(converter, frame);
    of = step(opticalFlow, im);
    lines = videooptflowlines(of, 20);
    if ~isempty(lines)
      out =  step(shapeInserter, im, lines);
      step(videoPlayer, out);
    end
end
```

Close the video reader and player

```
release(videoPlayer);
release(videoReader);
```

Algorithms

To compute the optical flow between two images, you must solve the following optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

In this equation, the following values are represented:

- I_x, I_y and I_t are the spatiotemporal image brightness derivatives
- u is the horizontal optical flow
- v is the vertical optical flow

Because this equation is underconstrained, there are several methods to solve for u and v:

- Horn-Schunck Method
- Lucas-Kanade Method

See the following two sections for descriptions of these methods

Horn-Schunck Method

By assuming that the optical flow is smooth over the entire image, the Horn-Schunck method computes an estimate of the velocity field, $[u \quad v]^T$, that minimizes this equation:

$$E = \iint (I_x u + I_y v + I_t)^2 dx dy + \alpha \iint \left\{ \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right\} dx dy$$

In this equation, $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ are the spatial derivatives of the optical velocity component u, and α scales the global smoothness term. The

Horn-Schunck method minimizes the previous equation to obtain the velocity field, [u v], for each pixel in the image, which is given by the following equations:

$$u_{x,y}^{k+1} = \bar{u}_{x,y}^{k} - \frac{I_x[I_x\bar{u}_{x,y}^{k} + I_y\bar{v}_{x,y}^{k} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v_{x,y}^{k+1} = \bar{v}_{x,y}^{k} - \frac{I_y[I_x\bar{u}_{x,y}^{k} + I_y\bar{v}_{x,y}^{k} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

In this equation, $\begin{bmatrix} u_{x,y}^{k} & v_{x,y}^{k} \end{bmatrix}$ is the velocity estimate for the pixel at

(x,y), and $\begin{bmatrix} \bar{u}_{x,y}^{k} & \bar{v}_{x,y}^{k} \end{bmatrix}$ is the neighborhood average of $\begin{bmatrix} u_{x,y}^{k} & v_{x,y}^{k} \end{bmatrix}$.
For *k=0*, the initial velocity is 0.

When you choose the Horn-Schunck method, u and v are solved as follows:

1 Compute I_x and I_y using the Sobel convolution kernel:

$\begin{bmatrix} -1 & -2 & -1; & 0 & 0 & 0; & 1 & 2 & 1 \end{bmatrix}$, and its transposed form for each pixel in the first image.

2 Compute I_t between images 1 and 2 using the $\begin{bmatrix} -1 & 1 \end{bmatrix}$ kernel.

3 Assume the previous velocity to be 0, and compute the average

velocity for each pixel using $\begin{bmatrix} 0 & 1 & 0; & 1 & 0 & 1; & 0 & 1 & 0 \end{bmatrix}$ as a convolution kernel.

4 Iteratively solve for u and v.

Lucas-Kanade Method

To solve the optical flow constraint equation for u and v, the Lucas-Kanade method divides the original image into smaller sections

and assumes a constant velocity in each section. Then, it performs a weighted least-square fit of the optical flow constraint equation to a constant model for $[u \quad v]^T$ in each section, Ω, by minimizing the following equation:

$$\sum_{x \in \Omega} W^2 [I_x u + I_y v + I_t]^2$$

Here, W is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is given by the following equation:

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

When you choose the Lucas-Kanade method, I_t is computed using a difference filter or a derivative of a Gaussian filter.

The two following sections explain how I_x, I_y, I_t, and then u and v are computed.

Difference Filter

When you set the **Temporal gradient filter** to Difference filter [-1 1], u and v are solved as follows:

1 Compute I_x and I_y using the kernel $\begin{bmatrix} -1 & 8 & 0 & -8 & 1 \end{bmatrix}/12$ and its transposed form.

If you are working with fixed-point data types, the kernel values are signed fixed-point values with word length equal to 16 and fraction length equal to 15.

2 Compute I_t between images 1 and 2 using the $\begin{bmatrix} -1 & 1 \end{bmatrix}$ kernel.

3 Smooth the gradient components, I_x, I_y, and I_t, using a separable and isotropic 5-by-5 element kernel whose effective 1-D coefficients

are $\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}/16$. If you are working with fixed-point data types, the kernel values are unsigned fixed-point values with word length equal to 8 and fraction length equal to 7.

4 Solve the 2-by-2 linear equations for each pixel using the following method:

- If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

  Then the eigenvalues of A are $\lambda_i = \dfrac{a+c}{2} \pm \dfrac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

  In the fixed-point diagrams, $P = \dfrac{a+c}{2}, Q = \dfrac{\sqrt{4b^2 + (a-c)^2}}{2}$

- The eigenvalues are compared to the threshold, τ, that corresponds to the value you enter for the threshold for noise reduction. The results fall into one of the following cases:

  Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

  A is nonsingular, the system of equations are solved using Cramer's rule.

  Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

  A is singular (noninvertible), the gradient flow is normalized to calculate u and v.

  Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

  The optical flow, u and v, is 0.

Derivative of Gaussian

If you set the temporal gradient filter to `Derivative of Gaussian`, u and v are solved using the following steps. You can see the flow chart for this process at the end of this section:

1 Compute I_x and I_y using the following steps:

    a Use a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the `BufferedFramesCount` property.

    b Use a Gaussian filter and the derivative of a Gaussian filter to smooth the image using spatial filtering. Specify the standard deviation and length of the image smoothing filter using the `ImageSmoothingFilterStandardDeviationt` property.

2 Compute I_t between images 1 and 2 using the following steps:

    a Use the derivative of a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the `BufferedFramesCount` parameter.

    b Use the filter described in step 1b to perform spatial filtering on the output of the temporal filter.

3 Smooth the gradient components, I_x, I_y, and I_t, using a gradient smoothing filter. Use the `GradientSmoothingFilterStandardDeviation` property to specify the standard deviation and the number of filter coefficients for the gradient smoothing filter.

4 Solve the 2-by-2 linear equations for each pixel using the following method:

- If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

  Then the eigenvalues of A are $\lambda_i = \dfrac{a+c}{2} \pm \dfrac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

- When the object finds the eigenvalues, it compares them to the threshold, τ, that corresponds to the value you enter for the NoiseReductionThreshold parameter. The results fall into one of the following cases:

  Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

  A is nonsingular, so the object solves the system of equations using Cramer's rule.

  Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

  A is singular (noninvertible), so the object normalizes the gradient flow to calculate u and v.

  Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

  The optical flow, u and v, is 0.

See Also   vision.Pyramid

Purpose   Create optical flow object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an OpticalFlow System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

For many System objects, this method is a no-op. Objects that have internal states will describe in their help what the reset method does for that object.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

vision.OpticalFlow.getNumOutputs

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   isLocked(h)

Description isLocked(h) returns the locked status, TF of the OpticalFlow System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

vision.OpticalFlow.release

Purpose Allow property value and input characteristics changes

Syntax release(H)

Description release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose   Estimate direction and speed of object motion between video frames

Syntax   VSQ = step(opticalFlow,I)
                  V = step(opticalFlow,I)
                  [...] = step(opticalFlow,I1,I2)
                  [..., IMV] = step(opticalFlow,I)

Description   VSQ = step(opticalFlow,I) computes the optical flow of input image,
                  I, from one video frame to another, and returns VSQ, specified as a
                  matrix of velocity magnitudes.

                  V = step(opticalFlow,I) computes the optical flow of input image. I,
                  from one video frame to another, and returns V, specified as a complex
                  matrix of horizontal and vertical components. This applies when you set
                  the OutputValue property to 'Horizontal and vertical components
                  in complex form'.

                  [...] = step(opticalFlow,I1,I2) computes the optical flow of the
                  input image I1, using I2 as a reference frame. This applies when you
                  set the ReferenceFrameSource property to 'Input port'.

                  [..., IMV] = step(opticalFlow,I) outputs the delayed input image,
                  IMV. The delay is equal to the latency introduced by the computation
                  of the motion vectors. This property applies when you set the Method
                  property to 'Lucas-Kanade', the TemporalGradientFilter property to
                  'Derivative of Gaussian', and the MotionVectorImageOutputport
                  property to true.

                  Note The object performs an initialization the first time the step
                  method is executed. This initialization locks nontunable properties and
                  input specifications, such as dimensions, complexity, and data type
                  of the input data. If you change a nontunable property or an input
                  specification, the System object issues an error. To change nontunable
                  properties or inputs, you must first call the release method to unlock
                  the object.

                  ---

vision.PeopleDetector

Purpose   Detect upright people using HOG features

Description   The people detector object detects people in an input image using the
Histogram of Oriented Gradient (HOG) features and a trained Support
Vector Machine (SVM) classifier. The object detects unoccluded people
in an upright position.

Construction   peopleDetector = vision.PeopleDetector returns a System object,
peopleDetector, that tracks a set of points in a video.

peopleDetector = vision.PeopleDetector(MODEL)
creates a peopleDetector System object and sets the
ClassificationModel property to MODEL. The input MODEL can be
either 'UprightPeople_128x64' or 'UprightPeople_96x48'.

peopleDetector = vision.PeopleDetector(Name,Value) configures
the tracker object properties, specified as one or more name-value pair
arguments. Unspecified properties have default values.

To detect people:

1 Define and set up your people detector object using the constructor.

2 Call the step method with the input image, I, and the people detector
object, peopleDetector. See the syntax below for using the step
method.

BBOXES = step(peopleDetector,I) performs multiscale object
detection on the input image, I. The method returns an *M*-by-4 matrix
defining *M* bounding boxes, where *M* represents the number of detected
people. Each row of the output matrix, BBOXES, contains a four-element
vector, [*x y* width height]. This vector specifies, in pixels, the upper-left
corner and size, of a bounding box. When no people are detected, the
step method returns an empty vector. The input image, I, must be a
grayscale or truecolor (RGB) image.

[BBOXES, SCORES] = step(peopleDetector,I) returns a confidence
value for the detections. The *M*-by-1 vector, SCORES, contain positive

values for each bounding box in BBOXES. Larger score values indicate a higher confidence in the detection.

Properties

ClassificationModel

Name of classification model

Specify the model as either 'UprightPeople_128x64' or 'UprightPeople_96x48'. The pixel dimensions indicate the image size used for training.

The images used to train the models include background pixels around the person. Therefore, the actual size of a detected person is smaller than the training image size.

Default: 'UprightPeople_128x64'

ClassificationThreshold

People classification threshold

Specify a nonnegative scalar threshold value. Use this threshold to control the classification of individual image subregions during multiscale detection. The threshold controls whether a subregion gets classified as a person. You can increase this value when there are many false detections. The higher the threshold value, the more stringent the requirements are for the classification. Vary the threshold over a range of values to find the optimum value for your data set. Typical values range from 0 to 4. This property is tunable.

Default: 1

MinSize

Smallest region containing a person

Specify a two-element vector, [height width], in pixels for the smallest region containing a person. When you know the minimum person size to detect, you can reduce computation time.

To do so, set this property to a value larger than the image size used to train the classification model. When you do not specify this property, the object sets it to the image size used to train the classification model. This property is tunable.

Default: [ ]

MaxSize

Largest region containing a person

Specify a two-element vector, [height width], in pixels for the largest region containing a person. When you know the maximum person size to detect, you can reduce computation time. To do so, set this property to a value smaller than the size of the input image. When you do not specify this property, the object sets it to the input image size. This property is tunable.

Default: [ ]

ScaleFactor

Multiscale object detection scaling

Specify a factor to incrementally scale the detection resolution between `MinSize` and `MaxSize`. The object calculates the detection resolution at each increment.

$$\text{round}(\textit{TrainingSize}*(\texttt{ScaleFactor}^N))$$

In this case, the *TrainingSize* is [128 64] for the 'UprightPeople_128x64' model and [96 48] for the 'UprightPeople_96x48' model. *N* is the increment. Decreasing the scale factor can increase the detection accuracy. However, doing so increases the computation time. This property is tunable.

Default: 1.05

WindowStride

Detection window stride

Specify a scalar or a two-element vector [*x y*] in pixels for the detection window stride. The object uses the window stride to slide the detection window across the image. When you specify this value as a vector, the first and second elements are the stride size in the *x* and *y* directions. When you specify this value as a scalar, the stride is the same for both *x* and *y*. Decreasing the window stride can increase the detection accuracy. However, doing so increases computation time. Increasing the window stride beyond [8 8] can lead to a greater number of missed detections. This property is tunable.

Default: [8 8]

MergeDetections

Merge detection control

Specify a logical scalar to control whether similar detections are merged. Set this property to true to merge bounding boxes using a mean-shift based algorithm. Set this property to false to output the unmerged bounding boxes.

Default: true

Methods		
	clone	Create alpha blender object with same property values
	isLocked	Locked status for input attributes and nontunable properties
	release	Allow property value and input characteristics changes
	step	Detect upright people using HOG features

vision.PeopleDetector

Examples

Detect People

1

Create a people detector, and load the input image.

```
peopleDetector = vision.PeopleDetector;
I = imread('visionteam1.jpg');
```

2

Detect people using the people detector object.

```
[bboxes, scores] = step(peopleDetector, I);
```

3

Create a shape inserter and a score inserter.

```
shapeInserter = vision.ShapeInserter('BorderColor','Custom','CustomBorder
```

```
scoreInserter = vision.TextInserter('Text',' %f','Color', [0 80 255],'Loc
```

4

Draw detected bounding boxes, and insert scores using the inserter objects.

```
 I = step(shapeInserter, I, int32(bboxes));
 I = step(scoreInserter, I, scores, int32(bboxes(:,1:2)));
```

```
figure, imshow(I)
title('Detected people and detection scores');
```

References   Dalal, N. and B. Triggs. "Histograms of Oriented Gradients for Human Detection,"*Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2005, pp. 886-893.

See Also   vision.ShapeInserter **|** vision.TextInserter **|** vision.CascadeObjectDetector **|** round | size

Purpose   Create alpha blender object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates an PeopleDetector System object *C*, with the
                    same property values as *H*. The clone method creates a new unlocked
                    object.

vision.PeopleDetector.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the PeopleDetector
                   System object.

                   The isLocked method returns a logical value that indicates whether
                   input attributes and nontunable properties for the object are locked. The
                   object performs an internal initialization the first time the step method
                   is executed. This initialization locks nontunable properties and input
                   specifications, such as dimensions, complexity, and data type of the
                   input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You cannot use the release method on System objects in Embedded MATLAB.

---

vision.PeopleDetector.step

Purpose　　　Detect upright people using HOG features

Syntax　　　　```
BBOXES = step(peopleDetector,I)
[BBOXES, SCORES] = step(peopleDetector,I)
```

**Description**　　BBOXES = step(peopleDetector,I) performs multi-scale object detection on the input image, I. The method returns an *M*-by-4 matrix defining *M* bounding boxes, where *M* represents the number of detected people. Each row of the output matrix, BBOXES, contains a four-element vector, [x y width height], that specifies in pixels, the upper-left corner and size of a bounding box. When no people are detected, the step method returns an empty vector. The input image I, must be a grayscale or truecolor (RGB) image.

[BBOXES, SCORES] = step(peopleDetector,I) additionally returns a confidence value for the detections. The *M*-by-1 vector, SCORES, contain positive values for each bounding box in BBOXES. Larger score values indicate a higher confidence in the detection.

---

**Note** The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

**Purpose**    Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm

**Description**    The point tracker object tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm. You can use the point tracker for video stabilization, camera motion estimation, and object tracking. It works particularly well for tracking objects that do not change shape and for those that exhibit visual texture. The point tracker is often used for short-term tracking as part of a larger tracking framework.

As the point tracker algorithm progresses over time, points can be lost due to lighting variation, out of plane rotation, or articulated motion. To track an object over a long period of time, you may need to reacquire points periodically.

**Construction**    pointTracker = vision.PointTracker returns a System object, pointTracker, that tracks a set of points in a video.

pointTracker = vision.PointTracker(Name,Value) configures the tracker object properties, specified as one or more name-value pair arguments. Unspecified properties have default values.

### Initialize Tracking Process

To initialize the tracking process, you must use the initialize method to specify the initial locations of the points and the initial video frame.

initialize(pointTracker,POINTS,I) initializes points to track and sets the initial video frame. The initial locations POINTS, must be an *M*-by-2 array of [x y] coordinates. The initial video frame, I, must be a 2-D grayscale or RGB image and must be the same size and data type as the video frames passed to the step method.

The vision.CornerDetector System object and the detectSURFFeatures, detectHarrisFeatures, and detectMinEigenFeatures functions are few of the many ways to obtain the initial points for tracking.

## To track a set of points:

**1** Define and set up your point tracker object using the constructor.

**2** Call the `step` method with the input image, `I`, and the point tracker object, `pointTracker`. See the following syntax for using the `step` method.

After initializing the tracking process, use the `step` method to track the points in subsequent video frames. You can also reset the points at any time by using the `setPoints` method.

`[POINTS,POINT_VALIDITY] = step(pointTracker,I)` tracks the points in the input frame, `I` using the point tracker object, `pointTracker`. The output `POINTS` contain an *M*-by-2 array of [x y] coordinates that correspond to the new locations of the points in the input frame, `I`. The output, `POINT_VALIDITY` provides an *M*-by-1 logical array, indicating whether or not each point has been reliably tracked.

A point can be invalid for several reasons. The point can become invalid if it falls outside of the image. Also, it can become invalid if the spatial gradient matrix computed in its neighborhood is singular. If the bidirectional error is greater than the `MaxBidirectionalError` threshold, this condition can also make the point invalid.

`[POINTS,POINT_VALIDITY,SCORES] = step(pointTracker,I)` additionally returns the confidence score for each point. The *M*-by-1 output array, `SCORE`, contains values between `0` and `1`. These values correspond to the degree of similarity between the neighborhood around the previous location and new location of each point. These values are computed as a function of the sum of squared differences between the previous and new neighborhoods. The greatest tracking confidence corresponds to a perfect match score of `1`.

`setPoints(pointTracker, POINTS)` sets the points for tracking. The method sets the *M*-by-2 `POINTS` array of [*x y*] coordinates with the points to track. You can use this method if the points need to be redetected because too many of them have been lost during tracking.

setPoints(pointTracker,POINTS,POINT_VALIDITY) additionally lets you mark points as either valid or invalid. The input logical vector POINT_VALIDITY of length *M*, contains the true or false value corresponding to the validity of the point to be tracked. The length *M* corresponds to the number of points. A false value indicates an invalid point that should not be tracked. For example, you can use this method with the estimateGeometricTransform function to determine the transformation between the point locations in the previous and current frames. You can mark the outliers as invalid.

**Properties**    **NumPyramidLevels**

Number of pyramid levels

Specify an integer scalar number of pyramid levels. The point tracker implementation of the KLT algorithm uses image pyramids. The object generates an image pyramid, where each level is reduced in resolution by a factor of two compared to the previous level. Selecting a pyramid level greater than 1, enables the algorithm to track the points at multiple levels of resolution, starting at the lowest level. Increasing the number of pyramid levels allows the algorithm to handle larger displacements of points between frames. However, computation cost also increases. Recommended values are between 1 and 4.

Pyramid Levels

Each pyramid level is formed by down-sampling the previous level by a factor of two in width and height. The point tracker begins tracking each point in the lowest resolution level, and continues tracking until convergence. The object propagates the result of that level to the next level as the initial guess of the point locations. In this way, the tracking is refined with each level, up to the original image. Using the pyramid levels allows the point tracker to handle large pixel motions, which can comprise distances greater than the neighborhood size.

**Default:** 3

### MaxBidirectionalError

Forward-backward error threshold

Specify a numeric scalar for the maximum bidirectional error. If the value is less than inf, the object tracks each point from the

previous to the current frame. It then tracks the same points back to the previous frame. The object calculates the bidirectional error. This value is the distance in pixels from the original location of the points to the final location after the backward tracking. The corresponding points are considered invalid when the error is greater than the value set for this property. Recommended values are between 0 and 3 pixels.

Using the bidirectional error is an effective way to eliminate points that could not be reliably tracked. However, the bidirectional error requires additional computation. When you set the MaxBidirectionalError property to inf, the object does not compute the bidirectional error.



**Default:** inf

### BlockSize

Size of neighborhood

Specify a two-element vector, [height, width] to represent the neighborhood around each point being tracked. The height and width must be odd integers. This neighborhood defines the area for the spatial gradient matrix computation. The minimum value for BlockSize is [5 5]. Increasing the size of the neighborhood, increases the computation time.

**Default:** [31 31]

### MaxIterations

Maximum number of search iterations

Specify a positive integer scalar for the maximum number of search iterations for each point. The KLT algorithm performs an iterative search for the new location of each point until convergence. Typically, the algorithm converges within 10 iterations. This property sets the limit on the number of search iterations. Recommended values are between 10 and 50.

**Default:** 30

## Methods

| | |
|---|---|
| initialize | Initialize video frame and points to track |
| setPoints | Set points to track |
| step | Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm |

## Examples

### Track a Face

**1**

Create System objects for reading and displaying video and for drawing a bounding box of the object.

```
videoFileReader = vision.VideoFileReader('visionface.avi');
videoPlayer = vision.VideoPlayer('Position', [100, 100, 680, 520]);
```

2

Read the first video frame, which contains the object, define the region.

```
objectFrame = step(videoFileReader);
objectRegion = [264, 122, 93, 93];
```

As an alternative, you can use the following commands to select the object region using a mouse. The object must occupy the majority of the

region.

```
figure; imshow(objectFrame);
objectRegion=round(getPosition(imrect))
```

3

Show initial frame with a red bounding box.

```
objectImage = insertShape(objectFrame, 'Rectangle', objectRegion,'Col
figure; imshow(objectImage); title('Yellow box shows object region');
```

4

Detect interest points in the object region.

```
points = detectMinEigenFeatures(rgb2gray(objectFrame), 'ROI', objectRe
```

5

Display the detected points.

```
pointImage = insertMarker(objectFrame, points.Location, '+', 'Color',
figure, imshow(pointImage), title('Detected interest points');
```

6

Create a tracker object.

```
markerInserter = vision.MarkerInserter('Shape','Plus','BorderColor','V
```

7

Create a tracker object.

```
tracker = vision.PointTracker('MaxBidirectionalError', 1);
```

8

Initialize the tracker.

```
initialize(tracker, points.Location, objectFrame);
```

9

Read, track, display points, and results in each video frame.

```
while ~isDone(videoFileReader)
      frame = step(videoFileReader);
      [points, validity] = step(tracker, frame);
      out = insertMarker(frame, points(validity, :), '+');
      step(videoPlayer, out);
end
```

10

Release the video reader and player.

```
release(videoPlayer);
release(videoFileReader);
```

References Lucas, Bruce D. and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision,"*Proceedings of the 7th International Joint Conference on Artificial Intelligence*, April, 1981, pp. 674–679.

Tomasi, Carlo and Takeo Kanade. *Detection and Tracking of Point Features*, Computer Science Department, Carnegie Mellon University, April, 1991.

Shi, Jianbo and Carlo Tomasi. "Good Features to Track," *IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. "Forward-Backward Error: Automatic Detection of Tracking Failures," *Proceedings of the 20th International Conference on Pattern Recognition*, 2010, pages 2756–2759, 2010.

See Also insertMarkervision.HistogramBasedTracker **|** | detectSURFFeatures | detectHarrisFeatures | detectMinEigenFeatures | estimateGeometricTransform | imrect

Purpose   Initialize video frame and points to track

Syntax   initialize(H,POINTS,I)

Description   initialize(H,POINTS,I) initializes points to track and sets the
                    initial video frame. The method sets the *M*-by-2 POINTS array of [x y]
                    coordinates with the points to track, and sets the initial video frame, I.
                    The input, POINTS, must be an *M*-by-2 array of [x y] coordinates. The
                    input, I, must be a 2-D grayscale or RGB image, and must be the same
                    size as the images passed to the step method.

vision.PointTracker.setPoints

Purpose   Set points to track

Syntax   setPoints(H,POINTS)
                   setPoints(H,POINTS,POINT_VALIDITY)

Description   setPoints(H,POINTS) sets the points for tracking. The method sets the
                   M-by-2 POINTS array of [x y] coordinates with the points to track. This
                   method can be used if the points need to be re-detected because too
                   many of them have been lost during tracking.

                   setPoints(H,POINTS,POINT_VALIDITY) additionally lets you
                   mark points as either valid or invalid. The input logical vector
                   POINT_VALIDITY of length *M*, contains the true or false value
                   corresponding to the validity of the point to be tracked. The length *M*
                   corresponds to the number of points. A false value indicates an invalid
                   point, and it should not be tracked. You can use this method with
                   the vision.GeometricTransformEstimator object to determine the
                   transformation between the point locations in the previous and current
                   frames, and then mark the outliers as invalid.

Purpose   Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm

Syntax   [POINTS,POINT_VALIDITY] = step(H,I)
                 [POINTS,POINT_VALIDITY,SCORES] = step(H,I)

Description [POINTS,POINT_VALIDITY] = step(H,I) tracks the points in the
                 input frame, I. The input POINTS contain an *M*-by-2 array of [x y]
                 coordinates that correspond to the new locations of the points in the
                 input frame, I. The input, POINT_VALIDITY provides an *M*-by-1 logical
                 array, indicating whether or not each point has been reliably tracked.
                 The input frame, I must be the same size as the image passed to the
                 initialize method.

                 [POINTS,POINT_VALIDITY,SCORES] = step(H,I) additionally
                 returns the confidence score for each point. The *M*-by-1 output
                 array, SCORE, contains values between 0 and 1. These values are
                 computed as a function of the sum of squared differences between the
                 BlockSize-by-*BlockSize* neighborhood around the point in the previous
                 frame, and the corresponding neighborhood in the current frame. The
                 greatest tracking confidence corresponds to a perfect match score of 1.

                 Note The object performs an initialization the first time the step
                 method is executed. This initialization locks nontunable properties and
                 input specifications, such as dimensions, complexity, and data type
                 of the input data. If you change a nontunable property or an input
                 specification, the System object issues an error. To change nontunable
                 properties or inputs, you must first call the release method to unlock
                 the object.

vision.PSNR

Purpose   Compute peak signal-to-noise ratio (PSNR) between images

Description   The PSNR object computes the peak signal-to-noise ratio (PSNR) between images. This ratio is often used as a quality measurement between an original and a compressed image.

Construction   H = vision.PSNR returns a System object, H, that computes the peak signal-to-noise ratio (PSNR) in decibels between two images.

Methods

clone	Create peak signal to noise ratio object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Compute peak signal-to-noise ratio

Examples   Compute the PSNR between an original image and its reconstructed image.

```
hdct2d = vision.DCT;
hidct2d = vision.IDCT;
hpsnr = vision.PSNR;
I = double(imread('cameraman.tif'));
J = step(hdct2d, I);
J(abs(J) < 10) = 0;
It = step(hidct2d, J);
psnr = step(hpsnr, I,It)
```

```
imshow(I, [0 255]), title('Original image');
figure, imshow(It,[0 255]), title('Reconstructed image');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the PSNR block reference page. The object properties correspond to the block parameters.

See Also   vision.DCT | vision.IDCT

vision.PSNR.clone

Purpose   Create peak signal to noise ratio object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a PSNR System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
                the step method.

                The getNumInputs method returns a positive integer that is the number
                of expected inputs (not counting the object itself) to the step method.
                This value will change if you alter any properties that turn inputs on or
                off. You must call the step method with the number of input arguments
                equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the PSNR System
object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.PSNR.release

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
or hardware connections) and allows all properties and input
characteristics to be changed.

> **Note** You can use the release method on a System object in code
> generated from MATLAB, but once you release its resources, you cannot
> use that System object again.

Purpose   Compute peak signal-to-noise ratio

Syntax   Y = step(H,X1,X2)

Description Y = step(H,X1,X2) computes the peak signal-to-noise ratio, *Y* , between images *X1* and *X2*. The two images *X1* and *X2* must have the same size.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.Pyramid

Purpose   Perform Gaussian pyramid decomposition

Description   The Pyramid object computes Gaussian pyramid reduction or expansion.
The image reduction step involves lowpass filtering and downsampling
the image pixels. The image expansion step involves upsampling the
image pixels and lowpass filtering.

Construction   gaussPyramid = vision.Pyramid returns a System object,
gaussPyramid, that computes a Gaussian pyramid reduction or
expansion of an image.

gaussPyramid = vision.Pyramid(*Name,Value*) configures the System
object properties, specified as one or more name-value pair arguments.
Unspecified properties have default values.

To compute a Gaussian pyramid:

1 Define and set up your pyramid object using the constructor.

2 Call the step method with the input image, I and the pyramid object,
gaussPyramid. See the syntax below for using the step method.

J = step(gaussPyramid,I) computes J, the Gaussian pyramid
decomposition of input I.

Properties   **Operation**

Reduce or expand the input image

Specify whether to reduce or expand the input image as Reduce
or Expand. If this property is set to Reduce, the object applies
a lowpass filter and then downsamples the input image. If this
property is set to Expand, the object upsamples and then applies a
lowpass filter to the input image.

Default: Reduce

PyramidLevel

Level of decomposition

Specify the number of times the object upsamples or downsamples each dimension of the image by a factor of 2.

Default: 1

SeparableFilter

How to specify the coefficients of low pass filter

Indicate how to specify the coefficients of the lowpass filter as `Default` or `Custom`.

Default: `Default`

CoefficientA

Coefficient 'a' of default separable filter

Specify the coefficients in the default separable filter `1/4-a/2 1/4 a 1/4 1/4-a/2` as a scalar value. This property applies when you set the `SeparableFilter` property to `Default`.

Default: `0.375`

CustomSeparableFilter

Separable filter coefficients

Specify separable filter coefficients as a vector. This property applies when you set the `SeparableFilter` property to `Custom`.

Default: `[0.0625 0.25 0.375 0.25 0.0625]`

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`.

Default: `Floor`

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`.

Default: `Wrap`

SeparableFilterDataType

CustomSeparableFilter word and fraction lengths

Specify the coefficients fixed-point data type as `Same word length as input`, `Custom`.

Default: `Custom`

CustomSeparableFilterDataType

CustomSeparableFilter word and fraction lengths

Specify the coefficients fixed-point type as a signed `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SeparableFilterDataType` property to `Custom`.

Default: `numerictype([],16,14)`

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Same as input`, or `Custom`.

Default: `Custom`

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`.

Default: `numerictype([],32,10)`

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Same as input`, or `Custom`.

Default: `Same as product`

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`.

Default: `numerictype([],32,0)`

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input`, or `Custom`.

Default: `Custom`

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a signed, scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`.

Default: `numerictype([],32,10)`

Methods

clone	Create pyramid object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Compute Gaussian pyramid decomposition of input

Examples   **Resize Image Using Gaussian Pyramid Decomposition**

Set the level of decomposition

```
gaussPyramid = vision.Pyramid('PyramidLevel', 2);
```

Cast to single, otherwise the returned output J, will be a fixed point object.

```
I = im2single(imread('cameraman.tif'));
J = step(gaussPyramid, I);
```

Display results.

```
figure, imshow(I); title('Original Image');
figure, imshow(J); title('Reduced Image');
```

See Also   vision.GeometricScaler | impyramid

Purpose   Create pyramid object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Pyramid System object *C*, with the same
                     property values as *H*. The clone method creates a new unlocked object.

vision.Pyramid.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.Pyramid.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the Pyramid System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose　　　Allow property value and input characteristics changes

Syntax　　　　release(H)

Description　　release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note　You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.Pyramid.step

Purpose   Compute Gaussian pyramid decomposition of input

Syntax   J = step(gaussPyramid,I)

Description   J = step(gaussPyramid,I) computes J, the Gaussian pyramid
                     decomposition of input I.

> **Note** The object performs an initialization the first time the step
> method is executed. This initialization locks nontunable properties and
> input specifications, such as dimensions, complexity, and data type
> of the input data. If you change a nontunable property or an input
> specification, the System object issues an error. To change nontunable
> properties or inputs, you must first call the release method to unlock
> the object.

Purpose   Draw rectangles, lines, polygons, or circles on an image

Description   The ShapeInserter object can draw multiple rectangles, lines, polygons, or circles in a 2-D grayscale or truecolor RGB image. The output image can then be displayed or saved to a file.

Construction shapeInserter = vision.ShapeInserter returns a System object, shapeInserter, that draws multiple rectangles, lines, polygons, or circles on images by overwriting pixel values.

shapeInserter = vision.ShapeInserter(*Name*,*Value*) returns a shape inserter System object, shapeInserter, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

To insert a shape:

1 Define and set up your shape inserter using the constructor.

2 Call the step method with the input image, I, the shape inserter object, shapeInserter, points PTS, and any optional properties. See the syntax below for using the step method.

J = step(shapeInserter,I,PTS) draws the shape specified by the Shape property on input image I. The input PTS specify the coordinates for the location of the shape. The shapes are embedded on the output image J.

J = step(shapeInserter,I,PTS,ROI) draws a shape inside an area defined by the ROI input. This applies only when you set the ROIInputPort property to true. The ROI input defines a rectangular area as [*x y width height*], where [*x y*] determine the upper-left corner location of the rectangle, and *width* and *height* specify the size.

J = step(shapeInserter,I,PTS,...,CLR) draws the shape with the border or fill color specified by the input CLR. This applies when you set the BorderColorSource property or the FillColorSource property to 'Input port'.

Properties   **Shape**

Shape to draw

You can specify the type of shape as Rectangles, Lines, Polygons, or Circles. When you specify the type of shape to draw, you must also specify the location. The PTS input specifies the location of the points. The table shows the format for the points input for the different shapes. For a more detailed explanation on how to specify shapes and lines, see "Draw Shapes and Lines".

Shape property	PTS input
'Rectangles'	*M*-by-4 matrix, where *M* specifies the number of rectangles. Each row specifies a rectangle in the format [*x y width height*], where [*x y*] determine the upper-left corner location of the rectangle, and *width* and *height* specify the size.
'Lines'	*M*-by-2*L* matrix, where *M* specifies the number of polylines. Each row specifies a polyline as a series of consecutive point locations, [$x_1, y_1, x_2, y_2, ..., x_L, y_L$].

Shape property	PTS input
`'Polygons'`	M-by-$2L$ matrix, where M specifies the number of polygons. Each row specifies a polygon as an array of consecutive point locations, $[x_1, y_1, x_2, y_2, ..., x_L, y_L]$. The point define the vertices of a polygon.
`'Circles'`	M-by-3 matrix, where M specifies the number of circles. Each row specifies a circle in the format $[x\ y\ radius]$, where $[x\ y]$ define the coordinates of the center of the circle.

Default: `Rectangles`

Fill

Enable filling shape

Set this property to `true` to fill the shape with an intensity value or a color. This property applies when you set the `Shape` property to `Lines`.

Default: `false`

BorderColorSource

Source of border color

Specify how the shape's border color is provided as `Input port` or `Property`. This property applies when you set the `Shape` property to `Lines` or, when you do not set the `Shape` property to `Lines` and you set the `Fill` property to `false`. When you set the`BorderColorSource` property to `Input port`, a border color vector must be provided as an input to the System object's `step` method.

Default: `Property`

BorderColor

Border color of shape

Specify the appearance of the shape's border as `Black`, `White`, or `Custom`. When you set this property to `Custom`, you can use the `CustomBorderColor` property to specify the value. This property applies when you set the `BorderColorSource` property to `Property`.

Default: `Black`

CustomBorderColor

Intensity or color value for shape's border

Specify an intensity or color value for the shape's border. For an intensity image input, this property can be set to either a scalar intensity value for one shape, or an R-element vector where R specifies the number of shapes. For a color input image, you can set this property to:

A P-element vector where P is the number of color planes.
A P-by-R matrix where P is the number of color planes and R is the number of shapes.

This property applies when you set the `BorderColor` property to `Custom`. This property is tunable when you set the `Antialiasing` property to `false`.

Default: `[200 255 100]`

FillColorSource

Source of fill color

Specify how to provide the shape's fill color as `Input port` or `Property`. This property applies when you set the `Fill` property to `true`, and you do not set the `Shape` property to `Lines`. When you set the `FillColorSource` to `Input port`, you must provide a fill color vector as an input to the System object's `step` method.

Default: `Property`

FillColor

Fill color of shape

Specify the intensity of the shading inside the shape as `Black`, `White`, or `Custom`. When you set this property to `Custom`, you can use the `CustomFillColor` property to specify the value. This property applies when set the `FillColorSource` property to `Property`.

Default: `Black`

CustomFillColor

Intensity or color value for shape's interior

Specify an intensity or color value for the shape's interior. For an intensity input image, you can set this property to a scalar intensity value for one shape, or an R-element vector where R specifies the number of shapes. For a color input image, you can set this property to:

A P-element vector where P is the number of color planes.
A P-by-R matrix where P is the number of color planes and R is the number of shapes.

This property applies when you set the `FillColor` property to `Custom`. This property is tunable when you set the `Antialiasing` property to `false`.

Default: `[200 255 100]`

Opacity

Opacity of the shading inside shapes

Specify the opacity of the shading inside the shape by a scalar value between `0` and `1`, where `0` is transparent and `1` is opaque. This property applies when you set the `Fill` property to `true`.

Default: `0.6`

ROIInputPort

Enable definition of area for drawing shapes via input

Set this property to `true` to define the area in which to draw the shapes via an input to the `step` method. Specify the area as a four-element vector, [*r c height width*], where *r* and *c* are the row and column coordinates of the upper-left corner of the area, and *height* and *width* represent the height (in rows) and width (in columns) of the area. When you set the property to `false`, the entire image will be used as the area in which to draw.

Default: `false`

Antialiasing

Enable performing smoothing algorithm on shapes

Set this property to `true` to perform a smoothing algorithm on the line, polygon, or circle. This property applies when you set the `Shape` property to `Lines`, `Polygons`, or `Circles`.

Default: `false`

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`.

Default: `Floor`

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` , or `Saturate`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`.

Default: `Wrap`

OpacityDataType

Opacity word length

Specify the opacity fixed-point data type as `Same word length as input`, or `Custom`. This property applies when you set the `Fill` property to `true`.

Default: `Custom`

CustomOpacityDataType

Opacity word length

Specify the opacity fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Fill` property to `true` and the `OpacityDataType` property to `Custom`.

Default: `numerictype([],16)`

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as `Same as first input`, or `Custom`. This property applies when you set the `Fill` property to true and/or the `Antialiasing` property to `true`.

Default: `Custom`

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`, and the `ProductDataType` property to `Custom`.

Default: `numerictype(true,32,14)`

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product`, `Same as first input`, or `Custom`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`.

Default: `Same as product`

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Fill` property to `true` and/or the `Antialiasing` property to `true`, and the `AccumulatorDataType` property to `Custom`.

Default: `numerictype([],32,14)`

Methods

clone	Create shape inserter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
step	Draw specified shape on image

Examples   **Draw a Black Rectangle in a Grayscale Input Image**

Create the shape inserter object.

```
shapeInserter = vision.ShapeInserter;
```

Read the input image.

```
I = imread('cameraman.tif');
```

Define the rectangle dimensions as [x y width height].

```
rectangle = int32([10 10 30 30]);
```

Draw the rectangle and display the result.

```
J = step(shapeInserter, I, rectangle);
imshow(J);
```

Draw Two Yellow Circles in a Grayscale Input I-mage.

Class of yellow must match class of the input, I.

```
yellow = uint8([255 255 0]); % [R G B]; class of yellow must match cla
```

Create the shape inserter object.

```
shapeInserter = vision.ShapeInserter('Shape','Circles','BorderColor',
```

Read the input image.

```
I = imread('cameraman.tif');
```

Define the circle dimensions

```
circles = int32([30 30 20; 80 80 25]); %  [x1 y1 radius1;x2 y2 radius2
```

Convert I to an RGB image.

```
RGB = repmat(I,[1,1,3]); % convert I to an RGB image
```

Draw the circles and display the result.

```
J = step(shapeInserter, RGB, circles);
```

```
imshow(J);
```

Draw a Red Triangle in a Color Image

Create a shape inserter object and read the image file.

```
shapeInserter = vision.ShapeInserter('Shape','Polygons','BorderColor','Cu
I = imread('autumn.tif');
```

Define vertices which will form a triangle: [x1 y1 x2 y2 x3 y3].

```
polygon = int32([50 60 100 60 75 30]);
```

Draw the triangles and display the result.

```
J = step(shapeInserter, I, polygon);
imshow(J);
```

See Also   vision.MarkerInserter | vision.TextInserter

Purpose   Create shape inserter object with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a ShapeInserter System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.ShapeInserter.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   `N = getNumInputs(H)`

Description   `N = getNumInputs(H)` returns the number of expected inputs, *N* to the `step` method.

The `getNumInputs` method returns a positive integer that is the number of expected inputs (not counting the object itself) to the `step` method. This value will change if you alter any properties that turn inputs on or off. You must call the `step` method with the number of input arguments equal to the result of `getNumInputs(H)`.

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

vision.ShapeInserter.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the ShapeInserter
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.ShapeInserter.step

Purpose   Draw specified shape on image

Syntax   J = step(shapeInserter,I,PTS)
J = step(shapeInserter,I,PTS,ROI)
J = step(shapeInserter,I,PTS,...,CLR)

Description J = step(shapeInserter,I,PTS) draws the shape specified by the
Shape property on input image I. The input PTS specify the coordinates
for the location of the shape. The shapes are embedded on the output
image J.

J = step(shapeInserter,I,PTS,ROI) draws a shape inside an
area defined by the ROI input. This applies only when you set the
ROIInputPort property to true. The ROI input defines a rectangular
area as [*x y width height*], where [*x y*] determine the upper-left corner
location of the rectangle, and *width* and *height* specify the size.

J = step(shapeInserter,I,PTS,...,CLR) draws the shape with the
border or fill color specified by the input CLR. This applies when you set
the BorderColorSource property or the FillColorSource property
to 'Input port'.

Note The object performs an initialization the first time the step
method is executed. This initialization locks nontunable properties and
input specifications, such as dimensions, complexity, and data type
of the input data. If you change a nontunable property or an input
specification, the System object issues an error. To change nontunable
properties or inputs, you must first call the release method to unlock
the object.

**Input
Arguments**

shapeInserter

Shape inserter object with shape and properties specified.

I

Input *M*-by-*N* matrix of *M* intensity values or an *M*-by-*N*-by-*P* matrix of *M* color values where *P* is the number of color planes.

- Double-precision floating point

- Single-precision floating point

- Fixed point

- Boolean

- 8-, 16-, and 32-bit signed integer

- 8-, 16-, and 32-bit unsigned integer

PTS

Input matrix coordinates describing location and dimension of shape. This property must be an integer value. If you enter non-integer value, the object rounds it to the nearest integer.

You can specify the type of shape as `Rectangles`, `Lines`, `Polygons`, or `Circles`. When you specify the type of shape to draw, you must also specify the location. The `PTS` input specifies the location of the points. The table shows the format for the points input for the different shapes. For a more detailed explanation on how to specify shapes and lines, see "Draw Shapes and Lines".

Depending on the shape you choose by setting the `Shape` property, the input matrix `PTS` must be in one of the following formats:

Shape	Format of PTS
Lines	*M*-by-4 matrix of *M* number of lines. $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$$ Each row of the matrix corresponds to a different line, and of the same form as the
Rectangles	*M*-by-4 matrix of *M* number of rectangles. $$\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \end{bmatrix}$$
Polygons	*M*-by-2*L* matrix $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} & y_{1L} \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} & y_{2L} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} & y_{ML} \end{bmatrix}$$ Each row of the matrix corresponds to a different
Circles	*M*-by-3 matrix $$\begin{bmatrix} x_1 & y_1 & radius_1 \\ x_2 & y_2 & radius_2 \\ \vdots & \vdots & \vdots \\ x_M & y_M & radius_M \end{bmatrix}$$ Each row of the matrix corresponds to a different circle and of the same form as the vector for a single circle.

The table below shows the data types required for the inputs, *I* and *PTS*.

Input image *I*	Input matrix *PTS*
built-in integer	built-in or fixed-point integer
fixed-point integer	built-in or fixed-point integer
double	double, single, or build-in integer
single	double, single, or build-in integer

RGB

Scalar, vector, or matrix describing one plane of the RGB input video stream.

ROI

Input *4*-element vector of integers [x y width height], that define a rectangular area in which to draw the shapes. The first two elements represent the one-based coordinates of the upper-left corner of the area. The second two elements represent the width and height of the area.

- Double-precision floating point

- Single-precision floating point

- 8-, 16-, and 32-bit signed integer

- 8-, 16-, and 32-bit unsigned integer

CLR

This port can be used to dynamically specify shape color.

P-element vector or an *M*-by-*P* matrix, where *M* is the number of shapes, and *P*, the number of color planes. You can specify a color (RGB), for each shape, or specify one color for all shapes. The data type for the CLR input must be the same as the input image.

Output Arguments

J

Output image. The shapes are embedded on the output image.

Purpose　　　　Find standard deviation of input or sequence of inputs

Description　　　The `StandardDeviation` object finds standard deviation of input or sequence of inputs.

Construction　　`H = vision.StandardDeviation` returns a System object, `H`, that computes the standard deviation of the entire input.

`H = vision.StandardDeviation(`*`Name`*`,`*`Value`*`)` returns a standard deviation System object, `H`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties　　　**RunningStandardDeviation**

　　　　　　　　Enable calculation over successive calls to step method

　　　　　　　　Set this property to `true` to enable the calculation of standard deviation over successive calls to the `step` method. The default is `false`.

　　　　　　ResetInputPort

　　　　　　　　Reset in running standard deviation mode

　　　　　　　　Set this property to `true` to enable resetting the running standard deviation. When the property is set to `true`, a reset input must be specified to the `step` method to reset the running standard deviation. This property applies when you set the `RunningStandardDeviation` property to `true`. The default is `false`.

　　　　　　ResetCondition

　　　　　　　　Reset condition for running standard deviation mode

　　　　　　　　Specify the event to reset the running standard deviation to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`. The default is `Non-zero`.

Dimension

Numerical dimension to operate along

Specify how the standard deviation calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. The default is `All`.

CustomDimension

Numerical dimension to operate along

Specify the dimension (one-based value) of the input signal, over which the standard deviation is computed. The value of this property cannot exceed the number of dimensions in the input signal. This property applies when you set the `Dimension` property to `Custom`. The default is `1`.

ROIProcessing

Enable region of interest processing

Set this property to `true` to enable calculating the standard deviation within a particular region of each image. This property applies when you set the `Dimension` property to `All` and the `RunningStandardDeviation` property to `false`. The default is `false`.

ROIForm

Type of region of interest

Specify the type of region of interest to `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies when you set the `ROIProcessing` property to `true`. The default is `Rectangles`.

ROIPortion

Calculate over entire ROI or just perimeter

Specify the region over which to calculate the standard deviation to `Entire ROI`, or `ROI perimeter`. This property applies when you set the `ROIForm` property to `Rectangles`. The default is `Entire ROI`.

ROIStatistics

Statistics for each ROI, or one for all ROIs

Specify what statistics to calculate as `Individual statistics for each ROI`, or `Single statistic for all ROIs`. This property does not apply when you set the `ROIForm` property to `Binary mask`. The default is `Individual statistics for each ROI`

ValidityOutputPort

Output flag indicating if any part of ROI is outside input image

Set this property to `true` to return the validity of the specified ROI as completely or partially inside of the image. This applies when you set the `ROIForm` property to `Lines` or `Rectangles`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `Label matrix`.

The default is `false`.

Methods

clone	Create standard deviation object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes

vision.StandardDeviation

reset	Reset running standard deviation state
step	Compute standard deviation of input

Examples Determine the standard deviation in a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hstd2d = vision.StandardDeviation;
std = step(hstd2d,img);
```

Algorithms This object implements the algorithm, inputs, and outputs described on the Standard Deviation block reference page. The object properties correspond to the block parameters.

Purpose   Create standard deviation object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a StandardDeviation System object *C*, with the same property values as *H*. The clone method creates a new unlocked object with uninitialized states.

vision.StandardDeviation.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to
                   the step method.

                   The getNumInputs method returns a positive integer that is the number
                   of expected inputs (not counting the object itself) to the step method.
                   This value will change if you alter any properties that turn inputs on or
                   off. You must call the step method with the number of input arguments
                   equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

vision.StandardDeviation.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the
StandardDeviation System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles
                   or hardware connections) and allows all properties and input
                   characteristics to be changed.

Note You can use the release method on a System object in code
generated from MATLAB, but once you release its resources, you cannot
use that System object again.

---

vision.StandardDeviation.reset

Purpose   Reset running standard deviation state

Syntax   reset(H)

Description   reset(H) resets the internal states of System object *H*, used for computing running standard deviation when the RunningStandardDeviation property is true.

Purpose　　　Compute standard deviation of input

Syntax　　　```
Y = step(H,X)
Y = step(H,X,R)
Y = step(H,X,ROI)
Y = step(H,X,LABEL,LABELNUMBERS)
[Y, FLAG] = step(H,X,ROI)
[Y, FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description　　Y = step(H,X) computes the standard deviation of input elements *X*. When you set the RunningStandardDeviation property to true, the output *Y* corresponds to the standard deviation of the input elements over successive calls to the step method.

Y = step(H,X,R) computes the standard deviation of the input elements over successive calls to the step method, and optionally resets its state based on the value of reset signal *R*, the ResetInputPort property and the ResetCondition property. This option applies when you set the RunningStandardDeviation property to true and the ResetInputPort property to true.

Y = step(H,X,ROI) uses additional input *ROI* as the region of interest when you set the ROIProcessing property to true and the ROIForm property to Lines, Rectangles or Binary mask.

Y = step(H,X,LABEL,LABELNUMBERS) computes the standard deviation of input image *X* for region labels contained in vector *LABELNUMBERS*, with matrix *LABEL* marking pixels of different regions. This option applies when you set the ROIProcessing property to true and the ROIForm property to Label matrix.

[Y, FLAG] = step(H,X,ROI) also returns output *FLAG*, which indicates whether the given region of interest is within the image bounds. This applies when you set the ValidityOutputPort property to true.

[Y, FLAG] = step(H,X,LABEL,LABELNUMBERS) also returns the output *FLAG* which indicates whether the input label numbers are valid when you set the ValidityOutputPort property to true.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Purpose   Locate template in image

Description The template matcher object locates a template in an image. Use the
step syntax below with input image I, template T, template matcher
object, H, and any optional properties.

LOC = step(H,I,T) computes the [x y] location coordinates, LOC, of the
best template match between the image matrix, I, and the template
matrix, T. The step method outputs the coordinates relative to the top
left corner of the image. The LOC [x y] coordinates correspond to the
center of the template. The object centers the location slightly different
for templates with an odd or even number of pixels, (see the table below
for details). The object computes the location by shifting the template in
single-pixel increments throughout the interior of the image.

METRIC = step(H,I,T) computes the match metric values for input
image, I, with T as the template. This applies when you set the
OutputValue property to Metric matrix.

LOC = step(H,I,T,ROI) computes the location of the best template
match, LOC, in the specified region of interest, ROI. This applies when
you set the OutputValue property to Best match location and the
ROIInputPort property to true. The input ROI must be a four element
vector, [x y width height], where the first two elements represent the [x
y] coordinates of the upper-left corner of the rectangular ROI.

[LOC,ROIVALID] = step(H,I,T,ROI) computes the location of the best
template match, LOC, in the specified region of interest, ROI. The step
method also returns a logical flag ROIVALID indicating if the specified
ROI is outside the bounds of the input image I. This applies when you
set the OutputValue property to Best match location, and both the
ROIInputPort and ROIValidityOutputPort properties to true.

[LOC,NVALS,NVALID] = step(H,I,T) returns the location of the
best template match LOC, the metric values around the best match
NVALS, and a logical flag NVALID. A false value for NVALID indicates
that the neighborhood around the best match extended outside the
borders of the metric value matrix NVALS. This applies when you

set the `OutputValue` property to `Best match location` and the `BestMatchNeighborhoodOutputPort` property to `true`.

`[LOC,NVALS,NVALID,ROIVALID] = step(H,I,T,ROI)` returns the location of the best template match `LOC`, the metric values around the best match `NVALS`, and two logical flags, `NVALID` and `ROIVALID`. A `false` value for `NVALID` indicates that the neighborhood around the best match extended outside the borders of the metric value matrix `NVALS`. A `false` value for `ROIVALID` indicates that the specified `ROI` is outside the bounds of the input image `I`. This applies when you set the `OutputValue` property to `Best match location`, and the `BestMatchNeighborhoodOutputPort`, `ROIInputPort`, and `ROIValidityOutputPort` properties to `true`.

The object outputs the best match coordinates, relative to the top-left corner of the image. The [x y] coordinates of the location correspond to the center of the template. When you use a template with an odd number of pixels, the object uses the center of the template. When you use a template with an even number of pixels, the object uses the centered upper-left pixel for the location. The following table shows how the block outputs the location (`LOC`), of odd and even templates:

Odd number of pixels in template	Even number of pixels in template

Construction   H = vision.TemplateMatcher returns a template matcher System
object, H. This object performs template matching by shifting a template
in single-pixel increments throughout the interior of an image.

H = vision.TemplateMatcher(*Name*,*Value*) returns a template
matcher object, H, with each specified property set to the specified value.

Properties   **Metric**

Metric used for template matching

Specify the metric to use for template matching as one of Sum
of absolute differences | Sum of squared differences |
Maximum absolute difference. The default is Sum of absolute
differences.

OutputValue

Type of output

Specify the output you want the object to return as one of `Metric matrix` | `Best match location`. The default is `Best match location`.

SearchMethod

Specify search criteria to find minimum difference between two inputs

Specify how the object searches for the minimum difference between the two input matrices as `Exhaustive` or `Three-step`. If you set this property to `Exhaustive`, the object searches for the minimum difference pixel by pixel. If you set this property to `Three-step`, the object searches for the minimum difference using a steadily decreasing step size. The `Three-step` method is computationally less expensive than the `Exhaustive` method, but sometimes does not find the optimal solution. This property applies when you set the `OutputValue` property to `Best match location`.

The default is `Exhaustive`.

BestMatchNeighborhoodOutputPort

Enable metric values output

Set this property to `true` to return two outputs, *NMETRIC* and *NVALID*. The output *NMETRIC* denotes an N-by-N matrix of metric values around the best match, where N is the value of the `NeighborhoodSize` property. The output NVALID is a logical indicating whether the object went beyond the metric matrix to construct output *NMETRIC*. This property applies when you set the `OutputValue` property to `Best match location`.

The default is `false`.

NeighborhoodSize

Size of the metric values

Specify the size, *N*, of the *N*-by-*N* matrix of metric values as an odd number. For example, if the matrix size is 3-by-3 set this property to `3`. This property applies when you set the `OutputValue` property to `Best match location` and the `BestMatchNeighborhoodOutputPort` property to `true`.

The default is `3`.

ROIInputPort

Enable ROI specification through input

Set this property to `true` to define the Region of Interest (ROI) over which to perform the template matching. If you set this property to `true`, the ROI is specified using an input to the `step` method. Otherwise the entire input image is used.

The default is `false`.

ROIValidityOutputPort

Enable output of a flag indicating if any part of ROI is outside input image

When you set this property to `true`, the object returns an ROI flag. The flag, when set to `false`, indicates a part of the ROI is outside of the input image. This property applies when you set the `ROIInputPort` property to `true`.

The default is`false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` | `Saturate`. The default is `Wrap`.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as Same as first input, `Custom`. This property applies when you set the `Metric` property to Sum of squared differences.

The default is `Custom`.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `Metric` property to Sum of squared differences, and the `ProductDataType` property to `Custom`.

The default is `numerictype([],32,30)`.

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as first input` | `Custom`. The default is `Custom`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`.

The default is `numerictype([],32,0)`.

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as first input` | `Custom`. The default is `Same as first input`.This property applies when you set the `OutputValue` property to `Metric matrix`. This property applies when you set the `OutputValue` property to `Best match location`, and the `BestMatchNeighborhoodOutputPort` property to `true`.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`.

The default is `numerictype([],32,0)`.

Methods

clone	Create template matcher object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Finds the best template match within image

Examples

Find the location of a particular chip on an image of an electronic board:

```
htm=vision.TemplateMatcher;
hmi = vision.MarkerInserter('Size', 10, ...
    'Fill', true, 'FillColor', 'White', 'Opacity', 0.75); I = imread(
```

```
% Input image
  I = I(1:200,1:200,:);

% Use grayscale data for the search
  Igray = rgb2gray(I);

% Use a second similar chip as the template
  T = Igray(20:75,90:135)

% Find the [x y] coordinates of the chip's center
  Loc=step(htm,Igray,T);

% Mark the location on the image using white disc
  J = step(hmi, I, Loc);

imshow(T); title('Template');
figure; imshow(J); title('Marked target');
```

Algorithms   This object implements the algorithm, inputs, and outputs described on the Template Matching block reference page. The object properties correspond to the block parameters.

See Also   vision.OpticalFlow | vision.MarkerInserter

Purpose   Create template matcher object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a TemplateMatcher System object *C*, with the
same property values as *H*. The clone method creates a new unlocked
object.

vision.TemplateMatcher.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
the step method.

The getNumInputs method returns a positive integer that is the number
of expected inputs (not counting the object itself) to the step method.
This value will change if you alter any properties that turn inputs on or
off. You must call the step method with the number of input arguments
equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
method.

The getNumOutputs method returns a positive integer that is the
number of outputs from the step method. This value will change if you
alter any properties that turn outputs on or off.

vision.TemplateMatcher.isLocked

Purpose   Locked status for input attributes and nontunable properties

Syntax   `TF = isLocked(H)`

Description   `TF = isLocked(H)` returns the locked status, `TF` of the `TemplateMatcher` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the `isLocked` method returns a `true` value.

Purpose   Allow property value and input characteristics changes

Syntax   release(H)

Description   release(H) releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the release method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.TemplateMatcher.step

Purpose   Finds the best template match within image

Syntax
```
LOC = step(H,I,T)
METRIC = step(H,I,T)
LOC = step(H,I,T,ROI)
[LOC,ROIVALID] = step(H,I,T,ROI)
[LOC,NVALS,NVALID] = step(H,I,T)
[LOC,NVALS,NVALID,ROIVALID] = step(H,I,T,ROI)
```

Description   `LOC = step(H,I,T)` computes the [x y] location coordinates, `LOC`, of the best template match between the image matrix, `I`, and the template matrix, `T`. The `step` method outputs the coordinates relative to the top left corner of the image. The object computes the location by shifting the template in single-pixel increments throughout the interior of the image.

`METRIC = step(H,I,T)` computes the match metric values for input image, `I`, with `T` as the template. This applies when you set the `OutputValue` property to `Metric matrix`.

`LOC = step(H,I,T,ROI)` computes the location of the best template match, `LOC`, in the specified region of interest, `ROI`. This applies when you set the `OutputValue` property to `Best match location` and the `ROIInputPort` property to `true`. The input `ROI` must be a four element vector, [x y width height], where the first two elements represent the [x y] coordinates of the upper-left corner of the rectangular ROI.

`[LOC,ROIVALID] = step(H,I,T,ROI)` computes the location of the best template match, `LOC`, in the specified region of interest, `ROI`. The step method also returns a logical flag `ROIVALID` indicating if the specified `ROI` is outside the bounds of the input image `I`. This applies when you set the `OutputValue` property to `Best match location`, and both the `ROIInputPort` and `ROIValidityOutputPort` properties to `true`.

`[LOC,NVALS,NVALID] = step(H,I,T)` returns the location of the best template match `LOC`, the metric values around the best match `NVALS`, and a logical flag `NVALID`. A `false` value for `NVALID` indicates that the neighborhood around the best match extended outside the borders of the metric value matrix `NVALS`. This applies when you

set the `OutputValue` property to `Best match location` and the `BestMatchNeighborhoodOutputPort` property to `true`.

`[LOC,NVALS,NVALID,ROIVALID] = step(H,I,T,ROI)` returns the location of the best template match `LOC`, the metric values around the best match `NVALS`, and two logical flags, `NVALID` and `ROIVALID`. A `false` value for `NVALID` indicates that the neighborhood around the best match extended outside the borders of the metric value matrix `NVALS`. A `false` value for `ROIVALID` indicates that the specified `ROI` is outside the bounds of the input image `I`. This applies when you set the `OutputValue` property to `Best match location`, and the `BestMatchNeighborhoodOutputPort`, `ROIInputPort`, and `ROIValidityOutputPort` properties to `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.TextInserter

Purpose　　　　Draw text on image or video stream

Description　　The text inserter System object draws text in an image. The output image can then be displayed or saved to a file. It can draw one or more arbitrary ASCII strings at multiple locations in an image. The object uses the FreeType 2.3.5 library, an open-source font engine, to produce stylized text bitmaps. To learn more about the FreeType Project, visit http://www.freetype.org/. The text inserter object does not support character sets other than ASCII.

Construction　　`txtInserter = vision.TextInserter('string')` returns a text inserter object, `txtInserter` with the input text provided in quotes added to the image. You can specify the location for the text, font style, and other text characteristics with the properties described below.

`txtInserter = vision.TextInserter(...,Name,Value)` returns a text inserter object, `txtInserter`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

To insert text:

1 Define and set up your text inserter using the constructor.

2 Call the `step` method with the input image, `I`, the text inserter object, `txtInserter`, and any optional properties. See the syntax below for using the `step` method.

`J = step(txtInserter,I)` draws the text specified by the `Text` property onto input image `I` and returns the modified image `J`. The input image `I`, can be a 2-D grayscale image or a truecolor RGB image.

`J = step(txtInserter,I,LOC)` places the text at the [x y] coordinates specified by the `LOC` input. This applies when you set the `LocationSource` property to `'Input port'`.

`J = step(txtInserter,I,VARS)` uses the data in the input `VARS` for variable substitution. This applies when the `Text` property contains ANSI C printf-style format specifications (%d, %.2f, etc.). The input

VARS can be a scalar or a vector having length equal to the number of format specifiers in each element in the specified text string.

J = step(txtInserter,I,CELLIDX) draws the text string selected by the index CELLIDX input. This applies when you set the Text property to a cell array of strings. For example, if you set the Text property to {'str1','str2'}, and the CELLIDX property to 1, the object inserts the string 'str1' in the image. Values of CELLIDX outside of a valid range result in no text drawn on the image.

J = step(txtInserter,I,COLOR) sets the text intensity or color, specified by the Color property. The Color property can be a scalar or 3-element vector. This property applies when you set the ColorSource property to Input port.

J = step(txtInserter,I,OPAC) sets the text opacity, specified by the Opacity property. This property applies when you set the OpacitySource property to 'Input port'.

J = step(txtInserter,I,CELLIDX,VARS,COLOR,LOC,OPAC) draws the text specified by the Text property onto input image I, using the index CELLIDX. It also uses the text variable substitution data VARS, and color value COLOR. The object draws the text at location specified by LOC with opacity OPAC. You can use any combination or all possible inputs. Properties must be set appropriately.

Properties

Text

Text string to draw on image or video stream

Specify the text string to be drawn on image or video stream as a single text string or a cell array of strings. The string(s) can include ANSI C printf-style format specifications, such as %d, %f, or %s.

ColorSource

Source of intensity or color of text

Specify the intensity or color value of the text as Property or Input port. The default is Property.

Color

Text color or intensity

Specify the intensity or color of the text as a scalar integer value or a 3-element vector respectively. Alternatively, if the `Text` property is a cell array of *N* number of strings, specify a *M*-by-1 vector of intensity values or *M*-by-3 matrix of color values that correspond to each string. The default is [0 0 0]. This property applies when you set the `ColorSource` property to `Property`. This property is tunable.

LocationSource

Source of text location

Specify the location of the text as `Property` or `Input port`. The default is `Property`.

Location

Top-left corner of text bounding box

Specify the top-left corner of the text bounding box as a 2-element vector of integers, [x y]. The default is [1 1]. This property applies when you set the `LocationSource` property to `Property`. This property is tunable.

OpacitySource

Source of opacity of text

Specify the opacity of the text as `Property` or `Input port`. The default is `Property`.

Opacity

Opacity of text

Specify the opacity of the text as numeric scalar between 0 and 1. This property applies when you set the `OpacitySource` property to `Property`. The default is 1. This property is tunable.

TransposedInput

Specifies if input image data order is row major

Set this property to `true` to indicate that the input image data order is row major.

The default is `false`.

Font

Font face of text

Specify the font of the text as the available fonts installed on the system.

FontSize

Font size in points

Specify the font size as any positive integer value. The default is `12`.

Antialiasing

Perform smoothing algorithm on text edges

Set this property to `true` to smooth the edges of the text. The default is `true`.

Methods		
	clone	Create text inserter object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties

release	Allow property value and input characteristics changes
step	Draws the specified text onto input image

Examples   **Display a Text String in an Image**

1

Set up text [red, green, blue] formatted color and [x y] location.

```
textColor    = [255, 255, 255];
textLocation = [100 315];
```

2

Insert the text using the text inserter object.

```
textInserter = vision.TextInserter('Peppers are good for you!','Color', t
```

3

Display the original image and the image with text.

```
I = imread('peppers.png');
J = step(textInserter, I);
imshow(J);
```

Display Four Numeric Values at Different Locations

1

Set up the text using the text inserter object.

```
textInserter = vision.TextInserter('%d','LocationSource','Input port','Co
```

2

Read the image.

```
I = imread('peppers.png');
```

3

Insert the text.

```
J = step(textInserter, I, int32([1 2 3 4]),int32([1 1; 500 1; 1 350; 5
```

4

Display the image with the numeric values.

```
imshow(J);
```

Display Two Strings Inserted at Different Locations on an Image

1

Set up the text inserter.

```
textInserter = vision.TextInserter('%s','LocationSource','Input port',
```

2

Read the image.

```
I = imread('peppers.png');
```

3

Create null separated strings.

```
strings = uint8(['left' 0 'right']);
```

4

Insert the text.

```
J = step(textInserter, I, strings, int32([10 1; 450 1]));
```

5

Display the image with the numeric values.

```
imshow(J);
```

See Also   vision.AlphaBlender | vision.MarkerInserter |
                  vision.ShapeInserter

Purpose   Create text inserter object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a TextInserter System object *C*, with the same property values as *H*. The clone method creates a new unlocked object.

vision.TextInserter.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method.

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of outputs, *N* from the step
                   method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.TextInserter.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description TF = isLocked(H) returns the locked status, TF of the TextInserter System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.TextInserter.step

Purpose   Draws the specified text onto input image

Syntax   TXTIMG = step(H,IMG)
                   TXTIMG = step(H,IMG)
                   TXTIMG = step(H,IMG,CELLIDX)
                   TXTIMG = step(H,IMG,VARS)
                   TXTIMG = step(H,IMG,COLOR)
                   TXTIMG = step(H,IMG,LOC)
                   TXTIMG = step(H,IMG,OPAC)
                   TXTIMG = step(H,IMG,CELLIDX,VARS,COLOR,LOC,OPAC)

Description   TXTIMG = step(H,IMG) draws the specified text onto input image *IMG*
                   and returns the modified image *Y* . The image *IMG* can either be an
                   M-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* array color video
                   signal where *P* is the number of color planes.

                   TXTIMG = step(H,IMG) draws the specified text onto input image IMG
                   and returns the modified image TXTIMG. The image IMG can either be an
                   M-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* array color video
                   signal where *P* is the number of color planes.

                   TXTIMG = step(H,IMG,CELLIDX) draws the text string selected by
                   index, CELLIDX, when you set the Text property to a cell array of
                   strings. For example, if you set the Text property to {'str1','str2'}, and
                   the CELLIDX property to 1, the step method inserts the string 'str1' in
                   the image. Values of CELLIDX outside of valid range result in no text
                   drawn on the image.

                   TXTIMG = step(H,IMG,VARS) uses the data in VARS for variable
                   substitution, when the Text property contains ANSI C printf-style
                   format specifications (%d, %.2f, etc.). VARS is a scalar or a vector having
                   length equal to the number of format specifiers in each element in the
                   specified text string.

                   TXTIMG = step(H,IMG,COLOR) uses the given scalar or 3-element
                   vector COLOR for the text intensity or color respectively, when you set
                   the ColorSource property to Input port.

TXTIMG = step(H,IMG,LOC) places the text at the [x y] coordinates specified by LOC. This applies when you set the LocationSource property to Input port.

TXTIMG = step(H,IMG,OPAC) uses OPAC for the text opacity when you set the OpacitySource property to set to Input port.

TXTIMG = step(H,IMG,CELLIDX,VARS,COLOR,LOC,OPAC) draws the specified text onto image IMG using text string selection index CELLIDX , text variable substitution data VARS , intensity or color value COLOR at location LOC with opacity OPAC. You can use any combination or all possible inputs. Properties must be set appropriately.

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

---

vision.Variance

Purpose	Find variance values in an input or sequence of inputs
Description	The Variance object finds variance values in an input or sequence of inputs.
Construction	H = vision.Variance returns a System object, H, that computes the variance of an input or a sequence of inputs.

H = vision.Variance(*Name*,*Value*) returns a variance System object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (*Name1*, *Value1*,...,*NameN*,*ValueN*).

Properties

RunningVariance

Enable calculation over successive calls to step method

Set this property to true to enable the calculation of the variance over successive calls to the step method. The default is false.

ResetInputPort

Enable resetting via an input in running variance mode

Set this property to true to enable resetting the running variance. When the property is set to true, a reset input must be specified to the step method to reset the running variance. This property applies when you set the RunningVariance property to true. The default is false.

ResetCondition

Reset condition for running variance mode

Specify the event to reset the running variance as Rising edge, Falling edge, Either edge, or Non-zero. This property applies when you set the ResetInputPort property to true. The default is Non-zero.

CustomDimension

Numerical dimension to operate along

Specify the dimension (one-based value) of the input signal, over which the variance is computed. The value of this property cannot exceed the number of dimensions in the input signal. This property applies when you set the `Dimension` property to `Custom`. The default is `1`.

Dimension

Numerical dimension to operate along

Specify how the variance calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. This property applies only when you set the `RunningVariance` property to `false`. The default is `All`.

ROIForm

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies when you set the `ROIProcessing` property to `true`. Full ROI processing support requires a Computer Vision System Toolbox license. If you only have the DSP System Toolbox license, the `ROIForm` property value options are limited to `Rectangles`. The default is `Rectangles`.

ROIPortion

Calculate over entire ROI or just perimeter

Specify the region over which to calculate the variance as `Entire ROI`, or `ROI perimeter`. This property applies when you set the `ROIForm` property to `Rectangles`. The default is `Entire ROI`.

ROIProcessing

Enable region of interest processing

Set this property to `true` to enable calculating the variance within a particular region of each image. This property applies when you set the `Dimension` property to `All` and the `RunningVariance` property to `false`. Full ROI processing support requires a Computer Vision System Toolbox license. If you only have the

DSP System Toolbox license, the `ROIForm` property value options are limited to `Rectangles`. The default is `false`.

ROIStatistics

Statistics for each ROI, or one for all ROIs

Specify what statistics to calculate as `Individual statistics for each ROI`, or `Single statistic for all ROIs`. This property does not apply when you set the `ROIForm` property to `Binary mask`. The default is`Individual statistics for each ROI`.

ValidityOutputPort

Output flag indicating if any part of ROI is outside input image

Set this property to `true` to return the validity of the specified ROI as completely or partially inside of the image. This applies when you set the `ROIForm` property to `Lines` or `Rectangles`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `Label matrix`.

The default is `false`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling` | `Convergent` | `Floor` | `Nearest` | `Round` | `Simplest` | `Zero`. The default is `Floor`.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default is `Wrap`.

InputSquaredProductDataType

Input squared product and fraction lengths

Specify the input-squared product fixed-point data type as `Same as input` or `Custom`. The default is `Same as input`.

CustomInputSquaredProductDataType

Input squared product word and fraction lengths

Specify the input-squared product fixed-point type as a scaled `numerictype` object. This property applies when you set the `InputSquaredProductDataType` property to `Custom`. The default is `numerictype(true,32,15)`.

InputSumSquaredProductDataType

Input-sum-squared product and fraction lengths

Specify the input-sum-squared product fixed-point data type as `Same as input-squared product` or `Custom`. The default is `Same as input-squared product`.

CustomInputSumSquaredProductDataType

Input sum-squared product and fraction lengths

Specify the input-sum-squared product fixed-point type as a scaled `numerictype` object. This property applies when you set the `InputSumSquaredProductDataType` property to `Custom`. The default is `numerictype(true,32,23)`.

AccumulatorDataType

Data type of the accumulator

Specify the accumulator fixed-point data type as `Same as input`, or `Custom`. The default is `Same as input`.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled
numerictype object. This property applies when you set the
AccumulatorDataType property to Custom. The default is
numerictype(true,32,30).

OutputDataType

Data type of output

Specify the output fixed-point data type as Same as accumulator,
Same as input, or Custom. The default is Same as accumulator.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled numerictype object.

This property applies when you set the OutputDataType property
to Custom. The default is numerictype(true,32,30).

Methods		
	clone	Create variance object with same property values
	getNumInputs	Number of expected inputs to step method
	getNumOutputs	Number of outputs from step method
	isLocked	Locked status for input attributes and non-tunable properties
	release	Allow property value and input characteristics changes
	reset	Reset the internal states of the variance object
	step	Compute variance of input

Examples   Determine the variance in a grayscale image.

```
img = im2single(rgb2gray(imread('peppers.png')));
hvar2d = vision.Variance;
var2d = step(hvar2d,img);
```

vision.Variance.clone

Purpose   Create variance object with same property values

Syntax   C = clone(H)

Description   C = clone(H) creates a Variance System object C, with the same property values as H. The clone method creates a new unlocked object with uninitialized states.

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description N = getNumInputs(H) returns the number of expected inputs, *N* to
                the step method.

                The getNumInputs method returns a positive integer that is the number
                of expected inputs (not counting the object itself) to the step method.
                This value will change if you alter any properties that turn inputs on or
                off. You must call the step method with the number of input arguments
                equal to the result of getNumInputs(*H*).

vision.Variance.getNumOutputs

Purpose Number of outputs from step method

Syntax N = getNumOutputs(H)

Description N = getNumOutputs(H) returns the number of outputs *N* for the step method.

The getNumOutputs method returns a positive integer that is the number of outputs from the step method. This value will change if you alter any properties that turn outputs on or off.

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the Variance
System object.

The isLocked method returns a logical value that indicates whether
input attributes and nontunable properties for the object are locked. The
object performs an internal initialization the first time the step method
is executed. This initialization locks nontunable properties and input
specifications, such as dimensions, complexity, and data type of the
input data. After locking, the isLocked method returns a true value.

vision.Variance.release

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description   `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose　　　Reset the internal states of the variance object

Syntax　　　`reset(H)`

Description　　`reset(H)` resets the internal states of System object `H` to their initial values.

vision.Variance.step

Purpose   Compute variance of input

Syntax
```
Y = step(H,X)
Y = step(H,X,R)
Y = step(H,X,ROI)
Y = step(H,X,LABEL,LABELNUMBERS)
[Y,FLAG] = step(H,X,ROI)
[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description   Y = step(H,X) computes the variance of input *X*. When you set the RunningVariance property to true, the output *Y* corresponds to the standard deviation of the input elements over successive calls to the step method.

Y = step(H,X,R) computes the variance of the input elements *X* over successive calls to the step method, and optionally resets its state based on the value of the reset signal *R*, the ResetInputPort property and the ResetCondition property. This option applies when you set the RunningVariance property to true and the ResetInputPort to true.

Y = step(H,X,ROI) computes the variance of input image *X* within the given region of interest *ROI* when you set the ROIProcessing property to true and the ROIForm property to Lines, Rectangles or Binary mask.

Y = step(H,X,LABEL,LABELNUMBERS) computes the variance of input image *X* for region labels contained in vector *LABELNUMBERS*, with matrix *LABEL* marking pixels of different regions. This option applies when you set the ROIProcessing property to true and the ROIForm property to Label matrix.

[Y,FLAG] = step(H,X,ROI) also returns the output *FLAG*, which indicates whether the given region of interest is within the image bounds. This applies when you set both the ROIProcessing and the ValidityOutputPort properties to true and the ROIForm property to Lines, Rectangles or Binary mask.

[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS) also returns the *FLAG*, which indicates whether the input label numbers are valid. This applies

when you set both the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Label matrix`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

---

vision.VideoPlayer

Purpose   Play video or display image

Description   The `VideoPlayer` object can play a video or display image sequences.

Note If you own the MATLAB Coder product, you can generate C or C++ code from MATLAB code in which an instance of this system object is created. When you do so, the scope system object is automatically declared as an *extrinsic* variable. In this manner, you are able to see the scope display in the same way that you would see a figure using the `plot` function, without directly generating code from it. For the full list of system objects supporting code generation, see "Code Generation with System Objects" in the MATLAB Coder documentation.

Construction   `videoPlayer = vision.VideoPlayer` returns a video player object, `videoPlayer`, for displaying video frames. Each call to the `step` method displays the next video frame.

`videoPlayer = vision.VideoPlayer(Name,Value)` configures the video player properties, specified as one or more name-value pair arguments. Unspecified properties have default values.

To display video frames:

1 Define and set up your video player object using the constructor.

2 Call the `step` method with the shape inserter object, `textInserter`, and any optional properties. See the syntax below for using the `step` method.

`step(videoPlayer, I)` displays one grayscale or truecolor RGB video frame,`I`, in the video player.

Properties   **Name**

Caption display on video player window

Specify the caption to display on the video player window as a string.

Default: `Video`

Position

Size and position of the video player window in pixels

Specify the size and position of the video player window in pixels as a four-element vector of the form: `[left bottom width height]`. This property is tunable.

Default: Dependent on the screen resolution. Window positioned in the center of the screen with size of 410 pixels in width by 300 pixels in height.

Methods

clone	Create video player with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and non-tunable properties
release	Allow property value and input characteristics changes
reset	Reset displayed frame number to zero
step	Play video or image sequence

Examples   **Play a Video File**

Read video from a file and set up player object.

```
videoFReader = vision.VideoFileReader('viplanedeparture.avi');
videoPlayer = vision.VideoPlayer;
```

Play video. Every call to the `step` method reads another frame.

```
while ~isDone(videoFReader)
   frame = step(videoFReader);
   step(videoPlayer,frame);
end
```

Close the file reader and video player.

```
release(videoFReader);
release(videoPlayer);
```

See Also   "Video Display in a Custom User Interface" |
`vision.DeployableVideoPlayer` | `vision.VideoFileReader` |
`vision.VideoFileWriter` | `imshow` | `implay`

Purpose   Create video player with same property values

Syntax   C = clone(H)

Description C = clone(H) creates a VideoPlayer System object *C*, with the same
                 property values as *H*. The clone method creates a new unlocked object
                 with uninitialized states.

vision.VideoPlayer.getNumInputs

Purpose   Number of expected inputs to step method

Syntax   N = getNumInputs(H)

Description   N = getNumInputs(H) returns the number of expected inputs, *N* to the step method

The getNumInputs method returns a positive integer that is the number of expected inputs (not counting the object itself) to the step method. This value will change if you alter any properties that turn inputs on or off. You must call the step method with the number of input arguments equal to the result of getNumInputs(*H*).

Purpose   Number of outputs from step method

Syntax   N = getNumOutputs(H)

Description   N = getNumOutputs(H) returns the number of arguments *N* from the
                   step method.

                   The getNumOutputs method returns a positive integer that is the
                   number of outputs from the step method. This value will change if you
                   alter any properties that turn outputs on or off.

vision.VideoPlayer.isLocked

Purpose   Locked status for input attributes and non-tunable properties

Syntax   TF = isLocked(H)

Description   TF = isLocked(H) returns the locked status, TF of the VideoPlayer System object.

The isLocked method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. After locking, the isLocked method returns a true value.

Purpose   Allow property value and input characteristics changes

Syntax   `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

vision.VideoPlayer.reset

Purpose   Reset displayed frame number to zero

Syntax   reset(H)

Description   reset(H) resets the displayed frame number of the video player to zero.

Purpose   Play video or image sequence

Syntax   step(videoPlayer,I)

Description   step(videoPlayer,I) displays one grayscale or truecolor RGB video
                   frame,I, in the video player.

matlab.System.processInputSizeChangeImpl

Purpose	Action when input size changes
Syntax	`processInputSizeChangeImpl(obj,input1,...,inputN)`
Description	`processInputSizeChangeImpl(obj,input1,...,inputN)` specifies the actions to perform when any `step` method input changes size (after the first call to `step`).
	`processInputSizeChangeImpl` is called by the `step` method before the `stepImpl` method is called.

> **Note** You must set `Access=protected` for this method.

Tips	Use this method when property values or settings depend on the size of the input. For example, you may want to reset some or all of the states in the object when the input sizes change.
Input Arguments	**obj**
	System object handle
	input1,...,inputN
	Inputs to the System object `step` method.
Examples	Use `processInputSizeChangeImpl` to have the object reset if the input size changes. In this case, `ResetOnSizeChange` is a property of the object. If `ResetOnSizeChange` is true, then reset is called what an input size changes.

```
methods (Access=protected)
   function processInputSizeChangeImpl(obj, ~)
     if obj.ResetOnSizeChange
       reset(obj);
     end
   end
```

```
end
```

See Also   resetImpl | stepImpl

How To   • "Process Input Size Change"

matlab.System

Purpose   Base class for System objects

Description `matlab.System` is the base class for System objects. In your class
definition file, you must subclass your object from this base class (or
from another class that derives from this base class). Subclassing
allows you to use the implementation and service methods provided by
this base class to build your object. You use this syntax as the first line
of your class definition file to directly inherit from the `matlab.System`
base class, where `ObjectName` is the name of your object:

```
classdef ObjectName < matlab.System
```

Note You must set `Access=protected` for each `matlab.System`
method you use in your code.

Methods

cloneImpl	Copy System object
getDiscreteStateImpl	Discrete state property values
getNumInputsImpl	Number of input arguments passed to step and setup methods
getNumOutputsImpl	Number of outputs returned by method
isInactivePropertyImpl	Active or inactive flag for properties
loadObjectImpl	Load saved System object from MAT file
processInputSizeChangeImpl	Action when input size changes
processTunedPropertiesImpl	Action when tunable properties change
releaseImpl	Release resources

resetImpl	Reset System object states
saveObjectImpl	Save System object in MAT file
setProperties	Set property values from name-value pair inputs
setupImpl	Initialize System object
stepImpl	System output and state update equations
validateInputsImpl	Validate inputs to step method
validatePropertiesImpl	Validate property values

Attributes   In addition to the attributes available for MATLAB objects, you can apply the following attributes to any property of a custom System object.

Nontunable	After an object is locked (after `step` or `setup` has been called), use `Nontunable` to prevent a user from changing that property value. By default, all properties are tunable. The `Nontunable` attribute is useful to lock a property that has side effects when changed. This attribute is also useful for locking a property value assumed to be constant during processing. You should always specifiy properties that affect the number of input or output ports as `Nontunable`.
Logical	Use `Logical` to limit the property value to a logical, scalar value. Any scalar value that can be converted to a logical is also valid, such as 0 or 1.

PositiveInteger	Use `PositiveInteger` to limit the property value to a positive integer value.
DiscreteState	Use `DiscreteState` to mark a property so it will display its state value when you use the `getDiscreteState` method.

To learn more about attributes, see "Property Attributes" in the MATLAB Object-Oriented Programming documentation.

Examples

Create a simple System object, `AddOne`, which subclasses from `matlab.System`. You place this code into a MATLAB file, `AddOne.m`.

```
classdef AddOne < matlab.System
%ADDONE Compute an output value that increments the input by one

    methods (Access=protected)
        % stepImpl method is called by the step method.
        function y = stepImpl(~,x)
            y = x + 1;
        end
    end
end
```

To use this object, create an instance of `AddOne`, provide an input, and use the `step` method:

```
hAdder = AddOne;
x = 1;
y = step(hAdder,x)
```

Assign the `Nontunable` attribute to the `InitialValue` property, which you define in your class definition file.

```
properties (Nontunable)
   InitialValue
```

```
end
```

See Also   matlab.system.StringSet | matlab.system.mixin.FiniteSource

How To   • "Object-Oriented Programming"

              • Class Attributes

              • Property Attributes

              • "Method Attributes"

              • "Define Basic System Objects"

              • "Define Property Attributes"

matlab.System.cloneImpl

Purpose   Copy System object

Syntax   `cloneImpl(obj)`

Description   `cloneImpl(obj)` copies a System object by using the `saveObjectImpl`
                     and `loadObjectImpl` methods. The default `cloneImpl` copies an
                     object and its current state but does not copy any private or protected
                     properties. If the object you clone is locked and you use the default
                     `cloneImpl`, the new object will also be locked. If you define your own
                     `cloneImpl` and the associated `saveObjectImpl` and `loadObjectImpl`,
                     you can specify whether to clone the object's state and whether to clone
                     the object's private and protected properties.

                     `cloneImpl` is called by the `clone` method.

                     Note You must set `Access=protected` for this method.

Input   **obj**
Arguments
                         System object handle of object to clone.

Examples   Use the `cloneImpl` method to copy a System object

```
methods (Access=protected)
   function obj2 = cloneImpl(obj1)
      s = saveObject (obj1);
      obj2 = loadObject(s);
   end
end
```

See Also   saveObjectImpl | loaObjectImpl

How To   • "Clone System Object"

matlab.System.getDiscreteStateImpl

Purpose

Discrete state property values

Syntax

s = getDiscreteStateImpl(obj)

Description

s = getDiscreteStateImpl(obj) returns a struct s of state values. The field names of the struct are the object's DiscreteState property names. To restrict or change the values returned by getDiscreteState method, you can override this getDiscreteStateImpl method.

getDiscreteStatesImpl is called by the getDiscreteState method, which is called by the setup method.

Note You must set Access=protected for this method.

Input Arguments

obj

   System object handle

Output Arguments

s

   Struct of state values.

Examples

```
methods (Access=protected)
   function s = getDiscreteState(obj)
   end
end
```

See Also

setupImpl

How To

• "Define Property Attributes"

matlab.System.getNumInputsImpl

Purpose　　　Number of input arguments passed to step and setup methods

Syntax　　　`num = getNumInputsImpl(obj)`

Description　`num = getNumInputsImpl(obj)` returns the number of inputs `num` (excluding the System object handle) expected by the `step` method. The default implementation returns 1, which requires one input from the user, in addition to the System object handle. To specify a value other than 1, you must use include the `getNumInputsImpl` method in your class definition file.

`getNumInputsImpl` is called by the `getNumInputs` method and by the `setup` method if the number of inputs has not been determined already.

Note　You must set `Access=protected` for this method.

Do not set any object properties in this `getNumInputsImpl` method.

Input Arguments　**obj**

　　System object handle

Output Arguments　**num**

　　Number of inputs expected by the `step` method for the specified object.

　　Default: 1

Examples　Specify the number of inputs (2, in this case) expected by the `step` method.

```
methods (Access=protected)
   function num = getNumInputsImpl(obj)
      num = 2;
   end
```

```
end
```

Specify that the step method will not accept any inputs.

```
methods (Access=protected)
   function num = getNumInputsImpl(~)
      num = O;
   end
end
```

See Also   setupImpl | stepImpl | getNumOutputsImpl

How To   • "Change Number of Step Inputs or Outputs"

matlab.System.getNumOutputsImpl

Purpose	Number of outputs returned by step method
Syntax	num = getNumOutputsImpl (obj)
Description	num = getNumOutputsImpl (obj) returns the number of outputs from the step method. The default implementation returns 1 output. To specify a value other than 1, you must use include the getNumOutputsImpl method in your class definition file.

getNumOutputsImpl is called by the getNumOutputs method, if the number of outputs has not been determined already.

Note You must set Access=protected for this method.

Do not set any object properties in this getNumOutputsImpl method.

Input Arguments	**obj**
	System object handle
Output Arguments	**num**
	Number of outputs to be returned by the step method for the specified object.
Examples	Specify the number of outputs (2, in this case) returned from the step method.

```
methods (Access=protected)
   function num = getNumOutputsImpl(obj)
      num = 2;
   end
end
```

Specify that the step method does not return any outputs.

```
methods (Access=protected)
   function num = getNumOutputsImpl(~)
      num = O;
   end
end
```

See Also   stepImpl | getNumInputsImpl | setupImpl

How To   • "Change Number of Step Inputs or Outputs"

matlab.System.isInactivePropertyImpl

Purpose   Active or inactive flag for properties

Syntax   `flag = isInactivePropertyImpl(obj,prop)`

Description `flag = isInactivePropertyImpl(obj,prop)` specifies whether
a public, non-state property is inactive for the current object
configuration. An *inactive property* is a property that is not relevant to
the object, given the values of other properties. Inactive properties are
not shown if you use the `disp` method to display object properties. If
you attempt to use public access to directly access or use `get` or `set` on
an inactive property, a warning occurs.

`isInactiveProperty` is called by the `disp` method and by the `get` and
`set` methods.

Note You must set `Access=protected` for this method.

Input Arguments

obj

   System object handle

prop

   Public, non-state property name

Output Arguments

flag

   Logical scalar value indicating whether the input property `prop` is
   inactive for the current object configuration.

Examples   Display the `InitialValue` property only when the
`UseRandomInitialValue` property value is `false`.

```
methods (Access=protected)
  function flag = isInactivePropertyImpl(obj,propertyName)
    if strcmp(propertyName,'InitialValue')
      flag = obj.UseRandomInitialValue;
```

```
      else
        flag = false;
      end
    end
end
```

See Also   setProperties

How To   • "Hide Inactive Properties"

matlab.System.processTunedPropertiesImpl

Purpose	Action when tunable properties change
Syntax	`processTunedPropertiesImpl(obj)`
Description	`processTunedPropertiesImpl(obj)` specifies the actions to perform when one or more tunable property values change. This method is called as part of the next call to the `step` method after a tunable property value changes. A property is tunable only if its `Nontunable` attribute is `false`, which is the default.

`processTunedPropertiesImpl` is called by the `step` method.

> **Note** You must set `Access=protected` for this method.

Tips Use this method when a tunable property affects a different property value. For example, two property values determine when to calculate a lookup table. You want to perform that calculation when either property changes. You also want the calculation to be done only once if both properties change before the next call to the `step` method.

Input Arguments **obj**

  System object handle

Examples Use `processTunedPropertiesImpl` to recalculate the lookup table if the value of either the `NumNotes` or `MiddleC` property changes.

```
methods (Access=protected)
   function processTunedPropertiesImpl(obj)
      % Generate a lookup table of note frequencies
      obj.pLookupTable = obj.MiddleC * (1+log(1:obj.NumNotes)/log(12));
    end
end
```

See Also `validatePropertiesImpl` | `setProperties`

How To
- "Validate Property and Input Values"
- "Define Property Attributes"

matlab.System.loadObjectImpl

Purpose	Load saved System object from MAT file
Syntax	`loadObjectImpl(obj)`
Description	`loadObjectImpl(obj)` loads a saved System object, `obj`, from a MAT file. Your `loadObjectImpl` method should correspond to your `saveObjectImpl` method to ensure that all saved properties and data are loaded.
Input Arguments	**obj** System object handle
Examples	Load a saved System object. In this case, the object contains a child object, protected and private properties, and a discrete state.

```
methods(Access=protected)
  function loadObjectImpl(obj, s, wasLocked)
    % Load child System objects
    obj.child = matlab.System.loadObject(s.child);

    % Save protected & private properties
    obj.protected = s.protected;
    obj.pdependentprop = s.pdependentprop;

    % Save state only if locked when saved
    if wasLocked
      obj.state = s.state;
    end

    % Call base class method
    loadObjectImpl@matlab.System(obj,s,wasLocked);
  end
end
```

How To	• "Load System Object"

- "Save System Object"

matlab.System.saveObjectImpl

Purpose   Save System object in MAT file

Syntax   saveObjectImpl(obj)

Description   saveObjectImpl(obj) defines what System object obj property and
                    state values are saved in a MAT file when a user calls save on that
                    object. save calls saveObject, which then calls saveObjectImpl. If
                    you do not define a saveObjectImpl method for your System object
                    class, only public properties are saved. To save any private or protected
                    properties or state information, you must define a saveObjectImpl in
                    your class definition file.

                    You should save the state of an object only if the object is locked. When
                    the user loads that saved object, it loads in that locked state.

                    To save child object information, you use the associated saveObject
                    method within the saveObjectImpl method.

                    End users can use load, which calls loadObjectImpl to load a System
                    object into their workspace.

Input   **obj**
Arguments
                        System object handle

Examples   Define what is saved for the System object. Call the base class version
                    of saveObjectImpl to save public properties. Then, save any child
                    System objects and any protected and private properties. Finally, save
                    the state, if the object is locked.

```
methods(Access=protected)
   function s = saveObjectImpl(obj)
      s = saveObjectImpl@matlab.System(obj);
      s.child = matlab.System.saveObject(obj.child);
      s.protected = obj.protected;
      s.pdependentprop = obj.pdependentprop;
      if isLocked(obj)
         s.state = obj.state;
```

```
            end
        end
    end
```

How To
- "Save System Object"
- "Load System Object"

matlab.System.releaseImpl

Purpose   Release resources

Syntax   releaseImpl(obj)

Description releaseImpl(obj) releases any resources used by the System object, such as file handles. This method also performs any necessary cleanup tasks. To release resources for a System object, you must use releaseImpl instead of a destructor.

releaseImpl is called by the release method. releaseImpl is also called when the object is deleted or cleared from memory, or when all references to the object have gone out of scope.

Note You must set Access=protected for this method.

Input Arguments **obj**

System object handle

Examples   Use the releaseImpl method to close a file.

```
methods (Access=protected)
  function releaseImpl(obj)
    fclose(obj.pFileID);
  end
end
```

How To   • "Release System Object Resources"

Purpose   Reset System object states

Syntax   resetImpl(obj)

Description   resetImpl(obj) defines the state reset equations for the System object.
                     Typically you reset the states to a set of initial values.

                     resetImpl is called by the reset method. It is also called by the setup
                     method, after the setupImpl method.

                     Note You must set Access=protected for this method.

Input   obj
Arguments   System object handle

Examples   Use the reset method to reset the counter pCount property to zero.

                     ```
                     methods (Access=protected)
                        function resetImpl(obj)
                            obj.pCount = O;
                        end
                     end
                     ```

See Also   releaseImpl

How To   • "Reset Algorithm State"

matlab.System.setProperties

Purpose	Set property values from name-value pair inputs
Syntax	`setProperties(obj,numargs,name1,value1,name2,value2,...)`
	`setProperties(obj,numargs,arg1,...,argm,name1,value1,name2,`
	`value2,...)`

Description   `setProperties(obj,numargs,name1,value1,name2,value2,...)`
provides the name-value pair inputs to the System object constructor.
Use this syntax if every input must specify both name and value.

Note To allow standard name-value pair handling at construction,
define `setProperties` for your System object.

`setProperties(obj,numargs,arg1,...,argm,name1,value1,name2,value2,...)`
provides the value-only inputs, followed by the name-value pair inputs
to the System object during object construction. Use this syntax if you
want to allow users to specify one or more inputs by their values only.

Input Arguments

obj

    System objectSystem object handle

numargs

    Number of inputs passed in by the object constructor

name*

    Name of property

value*

    Value of the property

arg*

    Value of property (for value-only input to the object constructor)

Examples   Set up the object so users can specify property values via name-value
                 pairs when constructing the object.

```
methods
   function obj = MyFile(varargin)
      setProperties(obj,nargin,varargin{:});
   end
end
```

How To   • "Set Property Values at Construction Time"

matlab.System.setupImpl

Purpose   Initialize System object

Syntax   `setupImpl(obj,input1, input2,...)`

Description   `setupImpl(obj,input1, input2,...)` sets up a System object. To acquire resources for a System object, you must use `setupImpl` instead of a constructor. `setupImpl` executes the first time the `step` method is called on an object after that object has been created. It also executes the next time `step` is called after an object has been released. . The number of inputs must match the number of inputs defined in the `getNumInputsImpl` method. You pass the inputs into `setupImpl` to use the input sizes, datatypes, etc. in the one-time calculations.

`setupImpl` is called by the `setup` method, which is done automatically as the first subtask of the `step` method on an unlocked System object.

Note You must set `Access=protected` for this method.

Tips   To validate properties or inputs use the `validatePropertiesImpl`, `validateInputsImpl`, or `setProperties` methods. Do not include validation in `setupImpl`.

Input Arguments

obj

    System object handle

input*

    Inputs to the `setup` method

Examples   Open a file for writing using the `setupImpl` method.

```
methods (Access=protected)
  function setupImpl(obj,data)
    obj.pFileID = fopen(obj.Filename, 'wb');
    if obj.pFileID < 0
```

```
                error('Opening the file failed');
            end
        end
end
```

See Also   validatePropertiesImpl | validateInputsImpl | setProperties

How To
- "Initialize Properties and Setup One-Time Calculations"
- "Set Property Values at Construction Time"

matlab.System.stepImpl

Purpose	System output and state update equations
Syntax	`[output1,output2,...] = stepImpl(obj,input1,input2,...)`
Description	`[output1,output2,...]  = stepImpl(obj,input1,input2,...)` defines the algorithm to execute when you call the step method on the specified object `obj`. The step method calculates the outputs and updates the object's state values using the inputs, properties, and state update equations.

`stepImpl` is called by the step method.

Note You must set `Access=protected` for this method.

Tips The number of input arguments and output arguments must match the values returned by the `getNumInputsImpl` and `getNumOutputsImpl` methods, respectively

Input Arguments

obj

System object handle

input*

Inputs to the step method

Output Arguments

output

Output returned from the step method.

Examples Use the `stepImpl` method to increment two numbers.

```
methods (Access=protected)
  function [y1,y2] = stepImpl(obj,x1,x2)
     y1 = x1 + 1;
     y2 = x2 + 1;
   end
```

See Also   getNumInputsImpl | getNumOutputsImpl | validateInputsImpl

How To
- "Define Basic System Objects"
- "Change Number of Step Inputs or Outputs"

matlab.System.validateInputsImpl

Purpose	Validate inputs to step method
Syntax	`validateInputsImpl(obj,input1,input2,...)`

Description   `validateInputsImpl(obj,input1,input2,...)` validates inputs to the `step` method at the beginning of initialization Validation includes checking data types, complexity, cross-input validation, and validity of inputs controlled by a property value.

`validateInputsImpl` is called by the `setup` method before `setupImpl`. `validateInputsImpl` executes only once.

Note You must set `Access=protected` for this method.

Input Arguments

obj

    System object handle

input*

    Inputs to the `setup` method

Examples   Validate that the input is numeric.

```
methods (Access=protected)
   function validateInputsImpl(~,x)
      if ~isnumeric(x)
         error('Input must be numeric');
      end
   end
end
```

See Also   `validatePropertiesImpl | setupImpl`

How To   • "Validate Property and Input Values"

matlab.System.validatePropertiesImpl

Purpose   Validate property values

Syntax   validatePropertiesImpl(obj)

Description   validatePropertiesImpl(obj) validates interdependent or
                   interrelated property values at the beginning of object initialization,
                   such as checking that the dependent or related inputs are the same size.

                   validatePropertiesImpl is the first method called by the setup
                   method. validatePropertiesImpl also is called before the
                   processTunablePropertiesImpl method.

                   Note You must set Access=protected for this method.

Input   **obj**
Arguments   System object handle

Examples   Validate that the useIncrement property is true and that the value of
                   the increment property is greater than zero.

```
methods (Access=protected)
   function validatePropertiesImpl(obj)
      if obj.useIncrement && obj.increment < 0
         error('The increment value must be positive');
      end
    end
end
```

See Also   processTunedPropertiesImpl | setupImpl | validateInputsImpl

How To   • "Validate Property and Input Values"

matlab.system.mixin.FiniteSource

Purpose   Finite source mixin class

Description   `matlab.system.mixin.FiniteSource` is a class that defines the `isDone` method, which reports the state of a finite data source, such as an audio file.

To use this method, you must subclass from this class in addition to the `matlab.System` base class. You use the following syntax as the first line of your class definition file, where `ObjectName` is the name of your object:

```
classdef ObjectName < matlab.System &...
    matlab.system.mixin.FiniteSource
```

Methods   isDoneImpl   End-of-data flag

See Also   `matlab.System`

Tutorials   • "Define Finite Source Objects"

How To   • "Object-Oriented Programming"
                 • Class Attributes
                 • Property Attributes

matlab.system.mixin.FiniteSource.isDoneImpl

Purpose	End-of-data flag
Syntax	`status = isDoneImpl(obj)`

Description   `status = isDoneImpl(obj)` indicates if an end-of-data condition has occurred. The `isDone` method should return `false` when data from a finite source has been exhausted, typically by having read and output all data from the source. You should also define the result of future reads from an exhausted source in the `isDoneImpl` method.

`isDoneImpl` is called by the `isDone` method.

Input Arguments   **obj**

   System object handle

Output Arguments   **status**

   Logical value, `true` or `false`, that indicates if an end-of-data condition has occurred or not, respectively.

Examples   Set up `isDoneImpl` so the `isDone` method checks whether the object has completed eight iterations.

```
methods (Access=private)
   function bdone = isDoneImpl(obj)
      bdone = obj.NumIters==8;
   end
end
```

See Also   matlab.system.mixin.FiniteSource

How To   • "Define Finite Source Objects"

matlab.system.StringSet

Purpose

Set of valid string values

Description

`matlab.system.StringSet` defines a list of valid string values for a property. This class validates the string in the property and enables tab completion for the property value. A *StringSet* allows only predefined or customized strings as values for the property.

A `StringSet` uses two linked properties, which you must define in the same class. One is a public property that contains the current string value. This public property is displayed to the user. The other property is a hidden property that contains the list of all possible string values. This hidden property should also have the transient attribute so its value is not saved to disk when you save the System object.

The following considerations apply when using StringSets:

- The string property that holds the current string can have any name.

- The property that holds the `StringSet` must use the same name as the string property with the suffix "Set" appended to it. The string set property is an instance of the `matlab.system.StringSet` class.

- Valid strings, defined in the `StringSet`, must be declared using a cell array. The cell array cannot be empty nor can it have any empty strings. Valid strings must be unique and are case-insensitive.

- The string property must be set to a valid `StringSet` value.

Examples

Set the string property, `Flavor`, and the `StringSet` property, `FlavorSet`, in this example.

```
properties
  Flavor='Chocolate';
end

properties (Hidden,Transient)
  FlavorSet = ...
    matlab.system.StringSet({'Vanilla','Chocolate'});
end
```

See Also   `matlab.System`

How To

- "Object-Oriented Programming"
- Class Attributes
- Property Attributes
- "Limit Property Values to Finite String Set"

matlab.system.StringSet

3

Functions Alphabetical

visionlib

Purpose   Assign detections to tracks for multiobject tracking

Syntax
```
[assignments,unassignedTracks,
    unassignedDetections] = assignDetectionsToTracks( costMatri
    x,costOfNonAssignment)
[assignments,unassignedTracks,
    unassignedDetections] = assignDetectionsToTracks(costMatrix
    , unassignedTrackCost,unassignedDetectionCost)
```

Description   `[assignments,unassignedTracks,unassignedDetections] =
                   assignDetectionsToTracks(costMatrix,costOfNonAssignment)`
                   assigns detections to tracks in the context of multiple object tracking
                   using the James Munkres's variant of the Hungarian assignment
                   algorithm. It also determines which tracks are missing and which
                   detections should begin new tracks. It returns the indices of assigned
                   and unassigned tracks, and unassigned detections. The `costMatrix`
                   must be an M-by-N matrix. In this matrix, M represents the number
                   of tracks, and N is the number of detections. Each value represents
                   the cost of assigning the Nth detection to the Mth track. The lower the
                   cost, the more likely that a detection gets assigned to a track. The
                   `costOfNonAssignment` scalar input represents the cost of a track or a
                   detection remaining unassigned.

                   `[assignments,unassignedTracks,unassignedDetections]
                   = assignDetectionsToTracks(costMatrix,
                   unassignedTrackCost,unassignedDetectionCost)` specifies
                   the cost of unassigned tracks and detections separately. The
                   `unassignedTrackCost` must be a scalar value, or an M-element
                   vector, where M represents the number of tracks. For the M-element
                   vector, each element represents the cost of not assigning any detection
                   to that track. The `unassignedDetectionCost` must be a scalar value
                   or an N-element vector, where N represents the number of detections.

**Input
Arguments**   **costMatrix - Cost of assigning detection to track**
                   M-by-*N* matrix

                   Cost of assigning a detection to a track, specified as an M-by-N matrix,
                   where M represents the number of tracks, and N is the number of

detections. The cost matrix value must be real and nonsparse. The lower the cost, the more likely that a detection gets assigned to a track. Each value represents the cost of assigning the Nth detection to the Mth track. If there is no likelihood of an assignment between a detection and a track, the `costMatrix` input is set to `Inf`. Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. The function applies the Hungarian assignment algorithm to the padded matrix.
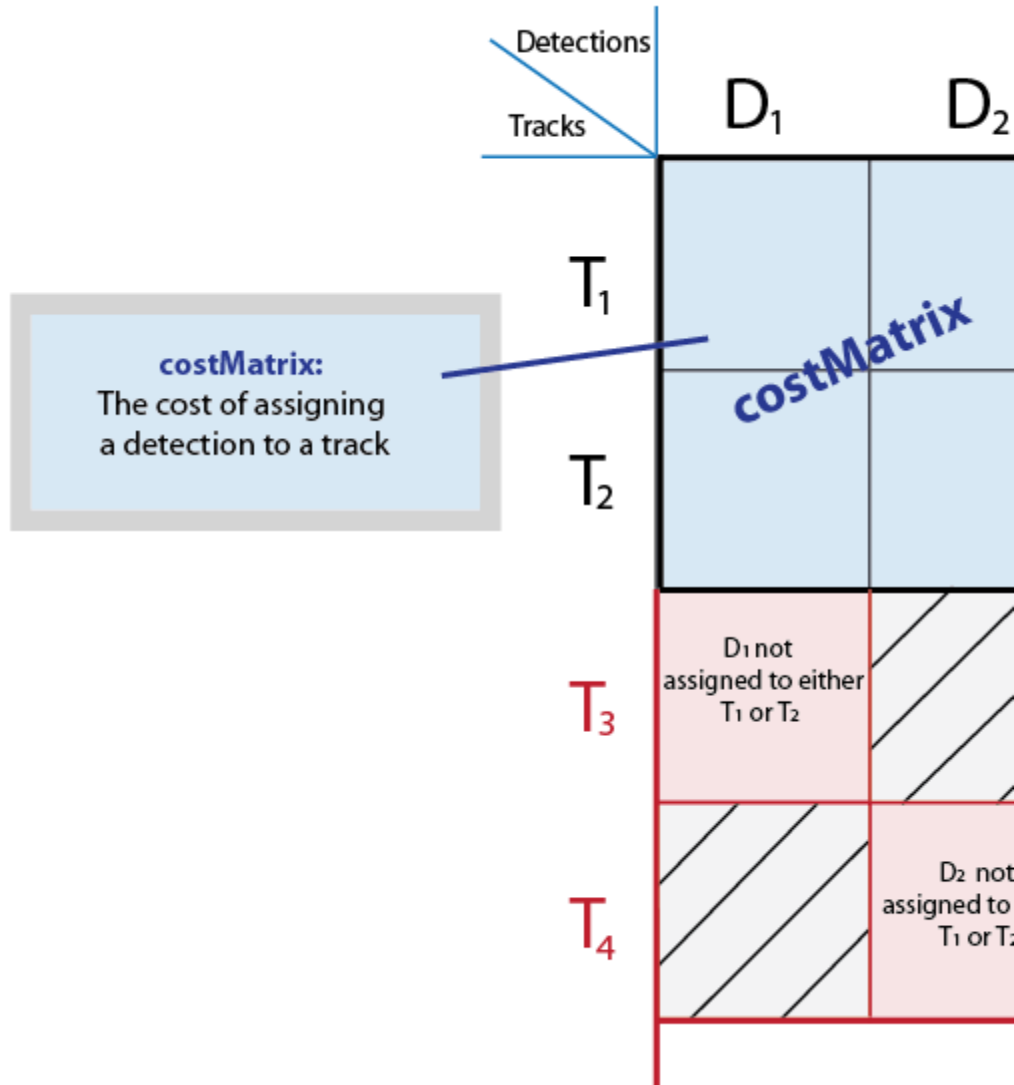
Data Types
```
int8 | uint8 | int16 | uint16 | int32 | uint32 | single |
double
```

costOfNonAssignment - Cost of not assigning detection to any track or track to any detection
scalar | finite

Cost of not assigning detection to any track or track to detection. You can specify this value as a scalar value representing the cost of a track or a detection remaining unassigned. An unassigned detection may become the start of a new track. If a track is unassigned, the object does not appear. The higher the `costOfNonAssignment` value, the higher the likelihood that every track will be assigned a detection.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input. To vary the values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs.

Data Types
`int8 | uint8 | int16 | uint16 | int32 | uint32 | single | double`

unassignedTrackCost - Cost or likelihood of an unassigned track
M-element vector | scalar | finite

Cost or likelihood of an unassigned track. You can specify this value as a scalar value, or an *M*-element vector, where *M* represents the number of tracks. For the *M*-element vector, each element represents the cost of not assigning any detection to that track. A scalar input represents the same cost of being unassigned for all tracks. The cost may vary depending on what you know about each track and the scene. For example, if an object is about to leave the field of view, the cost of the corresponding track being unassigned should be low.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To vary the values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input.

Detections

Tracks

D_1 D_2 D_3 D_4

T_1

T_2

costMatrix

T_1 not assigned to either D_1 or D_2

T_2 not assigned to eith D_1 or D_2

T_3

D_1 not assigned to either T_1 or T_2

0   0

T_4

D_2 not assigned to either T_1 or T_2

0   0

Data Types
```
int8 | uint8 | int16 | uint16 | int32 | uint32 | single |
double
```

unassignedDetectionCost - Cost of unassigned detection
N-element vector | scalar | finite

Cost of unassigned detection, specified as a scalar value or an *N*-element vector, where *N* represents the number of detections. For the *N*- element vector, each element represents the cost of starting a new track for that detection. A scalar input represents the same cost of being unassigned for all tracks. The cost may vary depending on what you know about each detection and the scene. For example, if a detection appears close to the edge of the image, it is more likely to be a new object.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To vary the values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input.

	D_1	D_2	D_3	D_4
T_1		costMatrix	T_1 not assigned to either D_1 or D_2	
T_2				T_2 not assigned to either D_1 or D_2
T_3	D_1 not assigned to either T_1 or T_2		0	0
T_4		D_2 not assigned to either T_1 or T_2	0	0

unassignedDetectionCost:
values vary for each
unassigned detecion to track

Data Types
`int8 | uint8 | int16 | uint16 | int32 | uint32 | single | double`

Output Arguments

assignments - Index pairs of tracks and corresponding detections
L-by-2 matrix

Index pairs of tracks and corresponding detections. This value is returned as an *L*-by-2 matrix of index pairs, with *L* number of pairs. The first column represents the track index and the second column represents the detection index.

Data Types
`uint32`

unassignedTracks - Unassigned tracks
P-element vector

Unassigned tracks, returned as a *P*-element vector. *P* represents the number of unassigned tracks. Each element represents a track to which no detections are assigned.

Data Types
`uint32`

unassignedDetections - Unassigned detections
Q-element vector

Unassigned detections, returned as a *Q*-element vector, where *Q* represents the number of unassigned detections. Each element represents a detection that was not assigned to any tracks. These detections can begin new tracks.

Data Types
`uint32`

Examples

Assign Detections to Tracks in a Single Video Frame

This example shows you how the `assignDetectionsToTracks` function works for a single video frame.

Set the predicted locations of objects in the current frame. Obtain predictions using the `vision.KalmanFilter` System object.

```
predictions = [1,1; 2,2];
```

Set the locations of the objects detected in the current frame. For this example, there are 2 tracks and 3 new detections. Thus, at least one of the detections is unmatched, which can indicate a new track.

```
detections = [1.1, 1.1; 2.1, 2.1; 1.5, 3];
```

Preallocate a cost matrix.

```
cost = zeros(size(predictions,1),size(detections,1));
```

Compute the cost of each prediction matching a detection. The cost here, is defined as the Euclidean distance between the prediction and the detection.

```
for i = 1:size(predictions, 1)
      diff = detections - repmat(predictions(i, :), [size(detections,
      cost(i, :) = sqrt(sum(diff .^ 2, 2));
end
```

Associate detections with predictions. Detection 1 should match to track 1, and detection 2 should match to track 2. Detection 3 should be unmatched.

```
[assignment, unassignedTracks, unassignedDetections] = assignDetection
  figure;
  plot(predictions(:, 1), predictions(:, 2), '*',detections(:, 1),dete
  hold on;
  legend('predictions', 'detections');
  for i = 1:size(assignment, 1)
    text(predictions(assignment(i, 1), 1)+0.1,predictions(assignment(:
    text(detections(assignment(i, 2), 1)+0.1,detections(assignment(i,
  end
  for i = 1:length(unassignedDetections)
    text(detections(unassignedDetections(i), 1)+0.1,detections(unassig
```

```
             end
             xlim([O, 4]);
             ylim([O, 4]);
```

See Also   `vision.KalmanFilter` **|** `configureKalmanFilter`

Purpose   Create Kalman filter for object tracking

Syntax
```
kalmanFilter = configureKalmanFilter(MotionModel,
    InitialLocation,InitialEstimateError, MotionNoise,
    MeasurementNoise)
```

Description `kalmanFilter =`
`configureKalmanFilter(MotionModel,InitialLocation,InitialEstimateError`
`MotionNoise,MeasurementNoise)` returns a `vision.KalmanFilter`
object configured to track a physical object. This object moves with
constant velocity or constant acceleration in an *M*-dimensional
Cartesian space. The function determines the number of dimensions,
M, from the length of the `InitialLocation` vector.

This function provides a simple approach for configuring the
`vision.KalmanFilter` object for tracking a physical object in a
Cartesian coordinate system. The tracked object may move with either
constant velocity or constant acceleration. The statistics are the same
along all dimensions. If you need to configure a Kalman filter with
different assumptions, use the `vision.KalmanFilter` object directly.

Input
Arguments
MotionModel - Motion model
`'ConstantVelocity'` | `'ConstantAcceleration'`

Motion model, specified as the string `'ConstantVelocity'` or
`'ConstantAcceleration'`. The motion model you select applies to all
dimensions. For example, for the 2-D Cartesian coordinate system.
This mode applies to both *X* and *Y* directions.

Data Types
char

InitialLocation - Initial location of object
vector

Initial location of object, specified as a numeric vector. This argument
also determines the number of dimensions for the coordinate system.

For example, if you specify the initial location as a two-element vector, $[x_0, y_0]$, then a 2-D coordinate system is assumed.

Data Types
`double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

InitialEstimateError - Initial estimate uncertainty variance
2-element vector | 3-element vector

Initial estimate uncertainty variance, specified as a two- or three-element vector. The initial estimate error specifies the variance of the initial estimates of location, velocity, and acceleration of the tracked object. The function assumes a zero initial velocity and acceleration for the object, at the location you set with the `InitialLocation` property. You can set the `InitialEstimateError` to an approximated value:

$$(assumed\ values - actual\ values)^2 + the\ variance\ of\ the\ values$$

The value of this property affects the Kalman filter for the first few detections. Later, the estimate error is determined by the noise and input data. A larger value for the initial estimate error helps the Kalman filter to adapt to the detection results faster. However, a larger value also prevents the Kalman filter from removing noise from the first few detections.

Specify the initial estimate error as a two-element vector for constant velocity or a three-element vector for constant acceleration:

MotionModel	InitialEstimateError
ConstantVelocity	[*LocationVariance*, *VelocityVariance*]
ConstantAcceleration	[*LocationVariance*, *VelocityVariance*, *AccelerationVariance*]

Data Types
`double | single`

MotionNoise - Deviation of selected and actual model
2-element vector | 3-element vector

Deviation of selected and actual model, specified as a two- or three-element vector. The motion noise specifies the tolerance of the Kalman filter for the deviation from the chosen model. This tolerance compensates for the difference between the object's actual motion and that of the model you choose. Increasing this value may cause the Kalman filter to change its state to fit the detections. Such an increase may prevent the Kalman filter from removing enough noise from the detections. The values of this property stay constant and therefore may affect the long-term performance of the Kalman filter.

MotionModel	InitialEstimateError
ConstantVelocity	[*LocationVariance*, *VelocityVariance*]
ConstantAcceleration	[*LocationVariance*, *VelocityVariance*, *AccelerationVariance*]

Data Types
double | single

MeasurementNoise - Variance inaccuracy of detected location
scalar

Variance inaccuracy of detected location, specified as a scalar. It is directly related to the technique used to detect the physical objects. Increasing the MeasurementNoise value enables the Kalman filter to remove more noise from the detections. However, it may also cause the Kalman filter to adhere too closely to the motion model you chose, putting less emphasis on the detections. The values of this property stay constant, and therefore may affect the long-term performance of the Kalman filter.

Data Types
double | single

configureKalmanFilter

Output Arguments	**kalmanFilter - Configured Kalman filter tracking** object
	Configured Kalman filter, returned as a vision.KalmanFilter object for tracking.
Algorithms	This function provides a simple approach for configuring the vision.KalmanFilter object for tracking. The Kalman filter implements a discrete time, linear State-Space System. The configureKalmanFilter function sets the vision.KalmanFilter object properties.

The InitialLocation property corresponds to the measurement vector used in the Kalman filter state-space model. This table relates the measurement vector, *M*, to the state-space model for the Kalman filter.

State transition model, *A*, and Measurement model, *H*

The state transition model, *A*, and the measurement model, *H* of the state-space model, are set to block diagonal matrices made from *M* identical submatrices *A*s and *H*s, respectively:

$A = \texttt{blkdiag}(As\_1, As\_2, ..., As\_M)$

$H = \texttt{blkdiag}(Hs\_1, Hs\_2, ..., Hs\_M)$

The submatrices *A*s and *H*s are described below:

MotionModel	*A*s	*H*s
`ConstantVelocity'	[1 1; 0 1]	[1 0]
`ConstantAcceleration'	[1 1 0; 0 1 1; 0 0 1]	[1 0 0]

The Initial State, *x*:		
MotionModel	**Initial state, *x***	
`ConstantVelocity'	[InitialLocation(1), 0, ..., InitialLocation(*M*), 0]	

`ConstantAcceleration`	[InitialLocation(1), 0, 0, ..., InitialLocation(*M*), 0, 0]

The initial state estimation error covariance matrix, *P*:

P = diag(repmat(InitialError, [1, *M*]))

The process noise covariance, *Q*:

Q = diag(repmat(MotionNoise, [1, *M*]))

The measurement noise covariance, *R*:

R = diag(repmat(MeasurementNoise, [1, *M*])).

Examples   **Track an Occluded Object**

Detect and track a ball using Kalman filtering, foreground detection, and blob analysis.

Create System objects to read the video frames, detect foreground physical objects, and display results.

```
videoReader = vision.VideoFileReader('singleball.avi');
videoPlayer = vision.VideoPlayer('Position', [100, 100, 500, 400]);
foregroundDetector = vision.ForegroundDetector('NumTrainingFrames', 10
blobAnalyzer = vision.BlobAnalysis('AreaOutputPort', false, 'MinimumB]
```

Process each video frame to detect and track the ball. After reading the current video frame, the example searches for the ball by using background subtraction and blob analysis. When the ball is first detected, the example creates a Kalman filter. The Kalman filter determines the ball's location, whether it is detected or not. If the ball is detected, the Kalman filter first predicts its state at the current video frame. The filter then uses the newly detected location to correct the state, producing a filtered location. If the ball is missing, the Kalman

filter solely relies on its previous state to predict the ball's current
location.

```
kalmanFilter = []; isTrackInitialized = false;
 while ~isDone(videoReader)
   colorImage  = step(videoReader);

   foregroundMask = step(foregroundDetector, rgb2gray(colorImage));
   detectedLocation = step(blobAnalyzer, foregroundMask);
   isObjectDetected = size(detectedLocation, 1) > 0;

   if ~isTrackInitialized
     if isObjectDetected
       kalmanFilter = configureKalmanFilter('ConstantAcceleration',dete
       isTrackInitialized = true;
     end
     label = ''; circle = [];
   else
     if isObjectDetected
       % Reduce the measurement noise by calling predict, then correct
       predict(kalmanFilter);
       trackedLocation = correct(kalmanFilter, detectedLocation(1,:));
       label = 'Corrected';
     else % Object is missing
       trackedLocation = predict(kalmanFilter);
       label = 'Predicted';
     end
     circle = [trackedLocation, 5];
   end

   colorImage = insertObjectAnnotation(colorImage, 'circle', ...
     circle, label, 'Color', 'red');
   step(videoPlayer, colorImage);
 end % while
```

Release resources

```
release(videoPlayer);
```

```
release(videoReader);
```

See Also   vision.BlobAnalysis **|** vision.ForegroundDetector **|**
vision.KalmanFilter **|**

detectFASTFeatures

Purpose	Find corners using FAST algorithm
Syntax	points = detectFASTFeatures(I) points = detectFASTFeatures(I,Name,Value)

Description   points = detectFASTFeatures(I) returns a cornerPoints object, points. The object contains information about the feature points detected in a 2-D grayscale input image, I. The detectFASTFeatures function uses the Features from Accelerated Segment Test (FAST) algorithm to find feature points.

points = detectFASTFeatures(I,Name,Value)) uses additional options specified by one or more Name,Value pair arguments.

Input Arguments

I - Input image
real | nonsparse

Input image, specified in 2-D grayscale.

Example:

Data Types
single | double | int16 | uint8 | uint16 | logical

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'MinQuality','0.01','ROI', [50,150,100,200] specifies that the detector must use a 1% minimum accepted quality of corners within the designated region of interest. This region of interest is located at *x*=50, *y*=150. The ROI has a width of 100 pixels, and a height of 200 pixels.

'MinQuality' - Minimum accepted quality of corners

0.01 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of 'MinQuality' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'MinContrast' - Minimum intensity
0.2 (default)

Minimum intensity difference between corner and surrounding region, specified as the comma-separated pair consisting of 'MinContrast' and a scalar value in the range (0,1).

The minimum intensity represents a fraction of the maximum value of the image class. Increasing the value reduces the number of detected corners.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'ROI' - Rectangular region
[1 1 size(I,2) size(I,1)] (default) | vector

Rectangular region for corner detection, specified as the comma-separated pair consisting of 'ROI' and a vector of the format [*x y width height*].

The first two values [*x y*] represent the location of the upper-left corner of the region of interest. The last two values represent the width and height.

Example: 'ROI', [50,150,100,200]

detectFASTFeatures

Data Types
```
single | double | int8 | int16 | int32 | int64 | uint8 |
uint16 | uint32 | uint64
```
Complex Number Support: No

Output Arguments

points - Corner points
object

Corner points object, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D grayscale input image.

Examples

Find Corner Points in an Image Using the FAST Algorithm

Read the image.

```
I = imread('cameraman.tif');
```

Find the corners.

```
corners = detectFASTFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

See Also
cornerPoints **|** detectHARRISFeatures | detectMinEigenFeatures | detectSURFFeaturesSURFPoints **|** | extractFeatures | matchFeatures

**Related
Examples**

- "Find Corner Points using the Eigenvalue Algorithm" on page 3-32
- "Find Corner Points Using the Harris–Stephens Algorithm" on page 3-28

detectHarrisFeatures

Purpose	Find corners using Harris–Stephens algorithm
Syntax	points = detectHarrisFeatures(I) points = detectHarrisFeatures(I,Name,Value)
Description	points = detectHarrisFeatures(I) returns a `cornerPoints` object, `points`. The object contains information about the feature points detected in a 2-D grayscale input image, I. The detectHarrisFeatures function uses the Harris–Stephens algorithm to find these feature points. points = detectHarrisFeatures(I,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

Input Arguments

I - Input image

real | nonsparse

Input image, specified in 2-D grayscale.

Example:

Data Types

single | double | int16 | uint8 | uint16 | logical

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'MinQuality','0.01','ROI', [50,150,100,200] specifies that the detector must use a 1% minimum accepted quality of corners within the designated region of interest. This region of interest is located at *x*=50, *y*=150. The ROI has a width of 100 pixels and a height of 200 pixels.

'MinQuality' - Minimum accepted quality of corners

0.01 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of 'MinQuality' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Example: 'MinQuality', 0.01

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 |
uint16 | uint32 | uint64

'FilterSize' - Gaussian filter dimension
5 (default)

Gaussian filter dimension, specified as the comma-separated pair consisting of 'FilterSize' and an odd integer value in the range [3, inf).

The Gaussian filter smooths the gradient of the input image.

The function uses the FilterSize value to calculate the filter's dimensions, FilterSize-by-FilterSize. It also defines the standard deviation of the Gaussian filter as FilterSize/3.

Example: 'FilterSize', 5

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 |
uint16 | uint32 | uint64

'ROI' - Rectangular region
[1 1 size(I,2) size(I,1)] (default) | vector

Rectangular region for corner detection, specified as the comma-separated pair consisting of 'ROI' and a vector of the format [*x y width height*].

The first two values [*x y*] represent the location of the upper-left corner of the region of interest. The last two values represent the width and height.

Example: `'ROI', [50,150,100,200]`

Data Types
`single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`
Complex Number Support: No

Output Arguments

points - Corner points
object

Corner points object, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D grayscale input image.

Examples

Find Corner Points Using the Harris–Stephens Algorithm

Read the image.

```
I = checkerboard;
```

Find the corners.

```
corners = detectHarrisFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

See Also   cornerPoints **|** detectFASTFeatures | detectMinEigenFeatures
| detectSURFFeaturesSURFPoints **|** | extractFeatures |
matchFeatures

**Related
Examples**
- "Find Corner Points using the Eigenvalue Algorithm" on page 3-32
- "Find Corner Points in an Image Using the FAST Algorithm" on
page 3-24

detectMinEigenFeatures

Purpose	Find corners using minimum eigenvalue algorithm

Syntax

```
points = detectMinEigenFeatures(I)
points = detectMinEigenFeatures(I,Name,Value)
```

Description   points = detectMinEigenFeatures(I) returns a cornerPoints
object, points. The object contains information about the
feature points detected in a 2-D grayscale input image, I. The
detectMinEigenFeatures function uses the minimum eigenvalue
algorithm developed by Shi and Tomasi to find feature points.

points = detectMinEigenFeatures(I,Name,Value)) uses additional
options specified by one or more Name,Value pair arguments.

**Input
Arguments**

I - Input image

real | nonsparse

Input image, specified in 2-D grayscale.

Example:

Data Types
single | double | int16 | uint8 | uint16 | logical

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: 'MinQuality','0.01','ROI', [50,150,100,200] specifies
that the detector must use a 1% minimum accepted quality of corners
within the designated region of interest. This region of interest is
located at *x*=50, *y*=150. The ROI has a width of 100 pixels, and a height
of 200 pixels.

'MinQuality' - Minimum accepted quality of corners

0.01 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of 'MinQuality' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Example: 'MinQuality', 0.01

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'FilterSize' - Gaussian filter dimension
5 (default)

Gaussian filter dimension, specified as the comma-separated pair consisting of 'FilterSize' and an odd integer value in the range [3, inf).

The Gaussian filter smooths the gradient of the input image.

The function uses the FilterSize value to calculate the filter's dimensions, FilterSize-by-FilterSize. It also defines the standard deviation as FilterSize/3.

Example: 'FilterSize', 5

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'ROI' - Rectangular region
[1 1 size(I,2) size(I,1)] (default) | vector

Rectangular region for corner detection, specified as the comma-separated pair consisting of 'ROI' and a vector of the format [*x y width height*].

detectMinEigenFeatures

The first two values [*x y*] represent the location of the upper-left corner of the region of interest. The last two values represent the width and height.

Example: `'ROI', [50,150,100,200]`

Data Types
`single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`
Complex Number Support: No

Output Arguments

points - Corner points
object

Corner points, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D grayscale input image.

Examples

Find Corner Points using the Eigenvalue Algorithm

Read the image.

```
I = checkerboard;
```

Find the corners.

```
corners = detectMinEigenFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

See Also   cornerPoints **|** detectFASTFeatures | detectHarrisFeatures
                 | detectSURFFeaturesSURFPoints **|** | extractFeatures |
                 matchFeatures

**Related
Examples**
• "Find Corner Points Using the Harris–Stephens Algorithm" on page
3-28
• "Find Corner Points in an Image Using the FAST Algorithm" on
page 3-24

detectMSERFeatures

Purpose	Detect MSER features
Syntax	`REGIONS = detectMSERFeatures(I)` `REGIONS = detectMSERFeatures(I,Name,Value)`

Description `REGIONS = detectMSERFeatures(I)` returns an MSER regions object, `REGIONS`, containing information about MSER features detected in the 2-D grayscale input image, `I`. This object uses Maximally Stable Extremal Regions (MSER) algorithm to find regions.

`REGIONS = detectMSERFeatures(I,Name,Value)` sets additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

I

Input grayscale image. The input image can be uint8, int16, uint16, single or double. It must be real and nonsparse.

Name-Value Pair Arguments

'ThresholdDelta'

Scalar value set within the range:

```
0 < ThresholdDelta <= 100
```

This value is expressed as a percentage of the input data type range. The percentage specifies the step size between intensity threshold levels used in selecting extremal regions while testing for their stability. Decrease this value to return more regions. Typical values range from 0.8 to 4.

> **Default:** 2

'RegionAreaRange'

Two-element vector, [minArea maxArea], which specifies the size of the regions in pixels. This value allows the selection of regions containing pixels to be between minArea and maxArea, inclusive.

Default: `[30 14000]`

'MaxAreaVariation'

Positive scalar, which specifies the maximum allowed area variation between extremal regions at varying intensity thresholds. Increasing this value returns a greater number of regions, but they may be less stable. Stable regions are very similar in size over varying intensity thresholds. Typical values range from 0.1 to 1.0.

Default: `0.25`

Output Arguments

REGIONS

An MSER regions object containing information about MSER features detected in the grayscale input image.

Examples

Detect MSER Interest Points

Detect interest points, and mark their locations.

```
I = imread('cameraman.tif');
regions = detectMSERFeatures(I);
imshow(I); hold on;
plot(regions);
```

Algorithms

Intensity Threshold Levels

The MSER detector incrementally steps through the intensity range of the input image to detect stable regions. The ThresholdDelta parameter determines the number of increments the detector tests for stability. You can think of the threshold delta value as the size of a cup to fill a bucket with water. The smaller the cup, the more number of increments it takes to fill up the bucket. The bucket can be thought of as the intensity profile of the region.

detectMSERFeatures
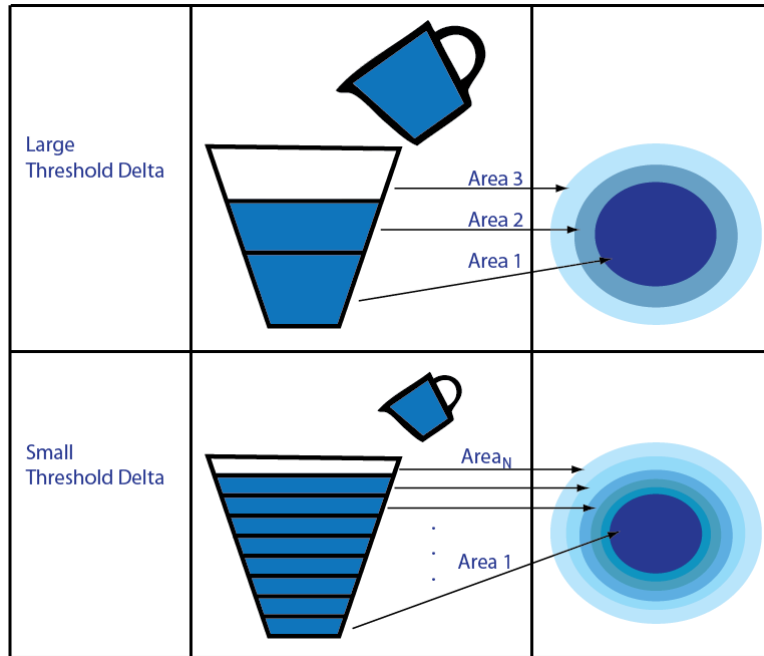
The MSER object checks the variation of the region area size between different intensity thresholds. The variation must be less than the value of the MaxAreaVariation parameter to be considered stable.

At a high level, MSER can be explained, by thinking of the intensity profile of an image representing a series of buckets. Imagine the tops of the buckets flush with the ground, and a hose turned on at one of the buckets. As the water fills into the bucket, it overflows and the next bucket starts filling up. Smaller regions of water join and become bigger bodies of water, and finally the whole area gets filled. As water is filling up into a bucket, it is checked against the MSER stability criterion. Regions appear, grow and merge at different intensity thresholds.

References   [1] Nister, D., and H. Stewenius, "Linear Time Maximally Stable Extremal Regions", *Lecture Notes in Computer Science.* 10th European

Conference on Computer Vision, Marseille, France: 2008, no. 5303, pp. 183–196.

[2] Matas, J., O. Chum, M. Urba, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions." *Proceedings of British Machine Vision Conference*, pages 384-396, 2002.

[3] Obdrzalek D., S. Basovnik, L. Mach, and A. Mikulik. "Detecting Scene Elements Using Maximally Stable Colour Regions," *Communications in Computer and Information Science*, La Ferte-Bernard, France; 2009, vol. 82 CCIS (2010 12 01), pp 107–115.

[4] Mikolajczyk, K., T. Tuytelaars, C. Schmid, A. Zisserman, T. Kadir, and L. Van Gool, "*A Comparison of Affine Region Detectors*"; International Journal of Computer Vision, Volume 65, Numbers 1–2 / November, 2005, pp 43–72 .

See Also   MSERRegions **|** detectSURFFeaturesSURFPoints **|** | detectMinEigenFeatures | detectHarrisFeatures | detectFASTFeatures | extractFeatures | matchFeatures

Related Examples
- "Detect SURF Interest Points in a Grayscale Image" on page 3-40
- "Detect MSER Features in an Image" on page 2-5
- "Display MSER Feature Regions from the MSERRegions Object" on page 2-6
- "Detect SURF Features in an Image" on page 2-23
- "Mix FAST Corner Detector with SURF Descriptors" on page 2-23
- "Combine MSER Region Detector with SURF Descriptors" on page 2-6

detectSURFFeatures

Purpose   Detect SURF features

Syntax
```
POINTS = detectSURFFeatures(I)
POINTS = detectSURFFeatures(I,Name,Value)
```

Description   `POINTS = detectSURFFeatures(I)` returns a `SURFPoints` object, `POINTS`, containing information about SURF features detected in the 2-D grayscale input image `I`. The `detectSURFFeatures` function implements the Speeded-Up Robust Features (SURF) algorithm to find blob features.

`POINTS = detectSURFFeatures(I,Name,Value)` Additional control for the algorithm requires specification of parameters and corresponding values. An additional option is specified by one or more `Name,Value` pair arguments.

Input Arguments

I

Grayscale input image. The input image can be `logical`, `uint8`, `int16`, `uint16`, `single`, or `double`, and it must be real and nonsparse.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'MetricThreshold'

A non-negative scalar which specifies a threshold for selecting the strongest features. To return more blobs, decrease the value of this threshold.

    Default: `1000.0`

'NumOctaves'

Number of octaves to use. This parameter must be an integer scalar greater than or equal to 1. Increase this value to detect larger blobs. Recommended values are between 1 and 4.

Each octave spans a number of scales that are analyzed using varying size filters:

Octave	Filter sizes
1	9-by-9, 15-by-15, 21-by-21, 27-by-27, ...
2	15-by-15, 27-by-27, 39-by-39, 51-by-51, ...
3	27-by-27, 51-by-51, 75-by-75, 99-by-99, ...
4	....

Higher octaves use larger filters and subsample the image data. Larger number of octaves will result in finding larger size blobs. Set the NumOctaves parameter appropriately for the image size. For example, a 50-by-50 image should not require you to set the NumOctaves parameter, greater than 2. The NumScaleLevels parameter controls the number of filters used per octave. At least three levels are required to analyze the data in a single octave.

Default: 3

'NumScaleLevels'

Number of scale levels to compute per octave. This parameter must be an integer scalar, greater than or equal to 3. Increase this number to detect more blobs at finer scale increments. Recommended values are between 3 and 6.

Default: 4

Output Arguments

POINTS

SURFPoints object containing information about SURF features detected in a grayscale image.

detectSURFFeatures

Examples

Detect SURF Interest Points in a Grayscale Image

Detect interest points in a grayscale, image and mark their locations.

```
I = imread('cameraman.tif');
points = detectSURFFeatures(I);
imshow(I); hold on;
plot(points.selectStrongest(10));
```

References

[1] Bradski, G. and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*. O'Reilly: Sebastopol, CA, 2008.

[2] Bay, Herbert, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, SURF: "Speeded Up Robust Features", *Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008

See Also

detectMSERFeatures | extractFeatures | matchFeatures | detectSURFFeaturesSURFPoints **|** MSERRegions **|**

Related Examples

Purpose   Disparity map between stereo images

Syntax   D = disparity(I1,I2)
                    D = disparity(I1,I2,ROI1,ROI2)
                    D = disparity(I1,I2,Name,Value)

Description   D = disparity(I1,I2) returns the disparity map for a pair of stereo
                    images, I1 and I2. The two input images, I1 and I2 must have the
                    same size and must be rectified, such that the corresponding points are
                    located on the same rows. The function returns the disparity map, D,
                    with the same size as I1 and I2. Each element of the output D, specifies
                    the disparity for the corresponding pixel in image I1.

                    D = disparity(I1,I2,ROI1,ROI2) returns the disparity map for
                    specified regions-of-interest, ROI1 and ROI2. Both inputs ROI1 and
                    ROI2 must have the format of [x y width height]. They also must be
                    fully contained in the input images I1 and I2. ROI1and ROI2 must
                    have the same size.

                    D = disparity(I1,I2,Name,Value) Additional control for the
                    disparity algorithm requires specification of parameters and
                    corresponding values. An additional option is specified by one or more
                    Name,Value pair arguments.

Tips   You can improve the speed of the function by setting the class of the
                    input images to uint8 and the number of columns for them, divisible
                    by 4.

**Input
Arguments**   **I1**

                    Input image referenced as I1, the left image. The input images must
                    be rectified such that the corresponding points are located on the same
                    rows.

                    Input images I1 and I2 must be real, finite, and nonsparse. They must
                    be the same class and must be uint8, uint16, int16, single, or double.

                    I2

Input image referenced as I2, the right image. The input images must be rectified such that the corresponding points are located on the same rows.

Input images I1 and I2 must be real, finite, and nonsparse. They must be the same class and must be uint8, uint16, int16, single, or double.

ROI1

Regions-of-interest within image I1 for computing the disparity map. ROI1 must have the format of [X Y WIDTH HEIGHT] and must be fully contained in image I1. ROI1 and ROI2 must have the same size.

ROI2

Regions-of-interest within image I2 for computing the disparity map. ROI2 must have the format of [X Y WIDTH HEIGHT] and must be fully contained in image I2. ROI1 and ROI2 must have the same size.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'ContrastThreshold'

A scalar value greater than 0 and less than or equal to 1. The contrast threshold defines a range of contrast values. When you set this parameter to 1, the function saturates the least number of pixels. You can reduce this value to saturate more pixels.

ContrastThreshold must be real, finite, and nonsparse.

> **Default:** 0.5

'BlockSize'

An odd integer between `0` and `255`. The function sets the block to a square, of size `BlockSize`-by-`BlockSize` pixels.

`BlockSize` must be real, finite, and nonsparse.

>   **Default:** `15`

'DisparityRange'

A two-element vector defining the range of disparity. The two-element vector must be in the format [*MinDisparity MaxDisparity*], where *MinDisparity* must be an integer. The difference between *MaxDisparity* and *MinDisparity* must be divisible by `16`.

`DisparityRange` must be real, finite, and nonsparse.

>   **Default:** `[0 64]`

'TextureThreshold'

A scalar value greater than or equal to `0`, and less than or equal to `1`. The texture threshold defines the minimum texture value for a pixel to be considered reliable. The lower the texture value, the less reliable the computed disparity is for the pixel. When you increase this parameter, more pixels with low texture value are marked as unreliable. You can set this parameter to `0` to disable it.

The texture of a pixel is defined as the sum of the saturated contrast computed over the `BlockSize`-by-`BlockSize` window around the pixel. The function considers the disparity computed for the pixel unreliable and marks it, when the texture falls below the value defined by:

> $Texture < X*$ `TextureThreshold` * `BlockSize`^{2}

X represents the maximum value supported by the class of the input images, `I1` and `I2`.

`TextureThreshold` must be real, finite, and nonsparse.

>   **Default:** `0.0002`

'UniquenessThreshold'

A non-negative integer defining the minimum value of uniqueness. When the function sets a pixel to a low uniqueness value, the disparity computed for it is less reliable. Increasing this parameter will result in marking more pixels unreliable because of the higher threshold. When you set this parameter to 0, the function disables the uniqueness thresholding.

The function defines uniqueness as a ratio of the optimal disparity estimation and the less optimal disparity estimation. For example:

Let K be the best estimated disparity, and let V be the corresponding SAD (Sum of Absolute Difference) value. Consider V as the smallest SAD value over the whole disparity range, and v, the smallest SAD value over the whole disparity range excluding K, K-1, and K+1. If $v < V$ * (1+0.01*UniquenessThreshold), then the function considers the disparity for the pixel unreliable and marks it.

UniquenessThreshold must be real, finite, and nonsparse.

Default: 15

'DistanceThreshold'

A non-negative integer defining the maximum distance for left-to-right image checking between two points.

When you decrease the value of the distance threshold, you increase the reliability of the disparity map.

When you set this parameter to an empty matrix [ ], the function disables the parameter.

Example: Let p_1 be a point in image I_1.

Step 1: The function searches for point p_1's best match in image I_2 (left-to-right check) and finds point p_2.
Step 2: The function then searches for p_2's best match in image I_1 (right-to-left check) and finds point p_3.

If the search returns a distance between p_1 and p_3 greater than DistanceThreshold, the disparity for point p_1, function marks the pixel as unreliable.

DistanceThreshold must be real, finite, and nonsparse.

Default: [] (disabled)

Output Arguments

D

The disparity map for a pair of stereo images. The function returns the disparity map with the same size as the input images. Each element of the output specifies the disparity for the corresponding pixel in the image references as I1.

The function computes the disparity map in three steps:

1 Compute a measure of contrast of the image by using the Sobel filter.

2 Compute the disparity for each of the pixels by using block matching and sum of absolute differences (SAD).

3 Optionally, mark the pixels which contain unreliable disparity values. The function sets the pixel to the value returned by -realmax('single').

Examples   **Find the Disparity Map for a Pair of Stereo Images**

Compute and visualize disparity map for a pair of stereo images.

1

Load the images and convert them to grayscale.

```
I1 = rgb2gray(imread('scene_left.png'));
I2 = rgb2gray(imread('scene_right.png'));
```

2

disparity

Show the stereo images.

```
figure; imshowpair(I1,I2,'ColorChannels','red-cyan');
title('Red-cyan composite view of the stereo images');
```

3

Compute the disparity map.

```
d = disparity(I1, I2, 'BlockSize', 35,'DisparityRange', [-6 10], 'Uniquer
```

4

For the purpose of visualizing the disparity, replace the
-realmax('single') marker with the minimum disparity value.

```
marker_idx = (d == -realmax('single'));
d(marker_idx) = min(d(~marker_idx));
```

5

Show the disparity map. Brighter pixels indicate objects which are
closer to the camera.

```
figure; imshow(mat2gray(d));
```

References [1] K. Konolige, *Small Vision Systems: Hardware and Implementation*,
Proceedings of the 8th International Symposium in Robotic Research,
pages 203-212, 1997.

[2] G. Bradski and A. Kaehler, *Learning OpenCV : Computer Vision
with the OpenCV Library*, O'Reilly, Sebastopol, CA, 2008.

See Also vision.ShapeInserter | size | line | lineToBorderPoints |
estimateFundamentalMatrix | epipolarLine

Purpose   Compute epipolar lines for stereo images

Syntax
```
LINES = epipolarLine(F,PTS)
LINES = epipolarLine(F',PTS)
```

Description   `LINES = epipolarLine(F,PTS)` returns an *M*-by-3 matrix, `LINES`. The matrix represents the computed epipolar lines in the second image corresponding to the points, `PTS`, in the first image. The input `F` represents the fundamental matrix that maps points in the first image to the epipolar lines in the second image.

`LINES = epipolarLine(F',PTS)` returns an *M*-by-3 matrix `LINES`. The matrix represents the computed epipolar lines of the first image corresponding to the points, `PTS`, in the second image.

Input Arguments

F

A 3-by-3 fundamental matrix. If *P*1 represents a point in the first image *I*1 that corresponds to *P*2, a point in the second image *I*2, then:

$$[P_2,1] * F * [P_1,1]' = 0$$

`F` must be double or single.

PTS

An *M*-by-2 matrix, where each row contains the x and y coordinates of a point in the image. *M* represents the number of points.

`PTS` must be a double, single, or integer value.

Output Arguments

LINES

An *M*-by-3 matrix, where each row must be in the format, [*A*,*B*,*C*]. This corresponds to the definition of the line:

$$A * x + B * y + C = 0.$$
M represents the number of lines.

epipolarLine

Examples
Use the Least Median of Squares method to find inliers and to compute the fundamental matrix. The points, matchedPoints1 and matchedPoints2, have been putatively matched:

```
load stereoPointPairs
[fLMedS, inliers] = estimateFundamentalMatrix(matchedPoints1,matchedP

% Show the inliers in the first image.
I1 = imread('viprectification_deskLeft.png');
figure;
subplot(121); imshow(I1); title('Inliers and Epipolar Lines in First
plot(matchedPoints1(inliers,1), matchedPoints1(inliers,2), 'go')

% Compute the epipolar lines in the first image.
epiLines = epipolarLine(fLMedS', matchedPoints2(inliers, :));

% Compute the intersection points of the lines and the image border.
pts = lineToBorderPoints(epiLines, size(I1));

% Show the epipolar lines in the first image
line(pts(:, [1,3])', pts(:, [2,4])');

% Show the inliers in the second image.
I2 = imread('viprectification_deskRight.png');
subplot(122); imshow(I2); title('Inliers and Epipole Lines in Second
plot(matchedPoints2(inliers,1), matchedPoints2(inliers,2), 'go')

% Compute and show the epipolar lines in the second image.
epiLines = epipolarLine(fLMedS, matchedPoints1(inliers, :));
pts = lineToBorderPoints(epiLines, size(I2));
line(pts(:, [1,3])', pts(:, [2,4])');

truesize;
```

See Also
vision.ShapeInserter | size | line | lineToBorderPoints | estimateFundamentalMatrix

Purpose	Estimate fundamental matrix from corresponding points in stereo images

Syntax	estimateFundamentalMatrix
	F =
	estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
	[F,inliersIndex] =
	estimateFundamentalMatrix(matchedPoints1,
	  matchedPoints2)
	[F,inliersIndex,
	  status] = estimateFundamentalMatrix(matchedPoints1,
	  matchedPoints2)
	[F,inliersIndex,
	  status] = estimateFundamentalMatrix(matchedPoints1,
	  matchedPoints2,Name,Value)

Description	estimateFundamentalMatrix estimates the fundamental matrix from corresponding points in stereo images. This function can be configured to use all corresponding points or to exclude outliers. You can exclude outliers by using a robust estimation technique such as random-sample consensus (RANSAC).

F =
estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
returns the 3-by-3 fundamental matrix, F, using the least median
of squares (LMedS) method. The input points can be *M*-by-2
matrices of *M* number of [x y] coordinates, or SURFPoints,
MSERRegions, or cornerPoints object.

[F,inliersIndex] =
estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
returns logical indices, inliersIndex, for the inliers used to compute
the fundamental matrix. The inliersIndex output is an *M*-by-1
vector. The function sets the elements of the vector to true when the
corresponding point was used to compute the fundamental matrix. The
elements are set to false if they are not used.

estimateFundamentalMatrix

[F,inliersIndex,status] =
estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
returns a status code.

[F,inliersIndex,status] =
estimateFundamentalMatrix(matchedPoints1,matchedPoints2,Name,Value)
sets parameters for finding outliers and computing the fundamental
matrix F, with each specified parameter set to the specified value
with one or more comma-separated parameters, specified as
name-value pairs.

Input Arguments

matchedPoints1 - Coordinates of corresponding points
M-by-2 matrix | SURFPoints object | MSERRegions object |
cornerPoints object

Coordinates of corresponding points in image one, specified as an
M-by-2 matrix of *M* number of [x y] coordinates, or as a SURFPoints,
MSERRegions, or cornerPoints object. The matchedPoints1 input
must contain points which are putatively matched by using a function
such as matchFeatures.

matchedPoints2 - Coordinates of corresponding points
M-by-2 matrix | SURFPoints object | MSERRegions object |
cornerPoints object

Coordinates of corresponding points in image one, specified as an
M-by-2 matrix of *M* number of [x y] coordinates, or as a SURFPoints,
MSERRegions, or cornerPoints object. The matchedPoints2 input
must contain points which are putatively matched by using a function
such as matchFeatures.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

'Method' - Method used to compute the fundamental matrix
Norm8Point | LMedS (default) | RANSAC | MSAC | LTS

Method used to compute the fundamental matrix, specified as the comma-separated pair consisting of 'Method' and one of five strings. You can set this parameter to one of the following methods:

Norm8Point   Normalized eight-point algorithm. To produce reliable results, the inputs, matchedPoints1 and matchedPoints2 must match precisely.

LMedS   Least Median of Squares. Select this method if you know that at least 50% of the points in matchedPoints1 and matchedPoints2 are inliers.

RANSAC   RANdom SAmple Consensus. Select this method if you would like to set the distance threshold for the inliers.

MSAC   M-estimator SAmple Consensus. Select the M-estimator SAmple Consensus method if you would like to set the distance threshold for the inliers. Generally, the MSAC method converges more quickly than the RANSAC method.

LTS   Least Trimmed Squares. Select the Least Trimmed Squares method if you know a minimum percentage of inliers in matchedPoints1 and matchedPoints2. Generally, the LTS method converges more quickly than the LMedS method.

To produce reliable results using the Norm8Point algorithm, the inputs, matchedPoints1 and matchedPoints2, must match precisely. The other methods can tolerate outliers and therefore only require putatively matched input points. You can obtain putatively matched points by using the matchFeatures function.

Default: LMedS

'OutputClass' - Fundamental matrix class

estimateFundamentalMatrix

`'double'` (default) | `'single'`

Fundamental matrix class, specified as the comma-separated pair consisting of '`OutputClass`' and either the string `'double'` or `'single'`. This specifies the class for the fundamental matrix and the function's internal computations.

> **Default:** `'double'`

'NumTrials' - Number of random trials for finding the outliers
500 (default) | integer

Number of random trials for finding the outliers, specified as the comma-separated pair consisting of '`NumTrials`' and an integer value. This parameter applies when you set the Method parameter to LMedS, RANSAC, MSAC, or LTS.

When you set the Method parameter to either LMedS or LTS, the function uses the actual number of trials as the parameter value.

When you set the Method parameter to either RANSAC or MSAC, the function uses the maximum number of trials as the parameter value. The actual number of trials depends on matchedPoints1, matchedPoints2, and the value of the Confidence parameter.

Select the number of random trials to optimize speed and accuracy.

> **Default:** 500

'DistanceType' - Algebraic or Sampson distance type
`'Sampson'` (default) | `'Algebraic'`

Algebraic or Sampson distance type, specified as the comma-separated pair consisting of '`DistanceType`' and either the Algebraic or Sampson string. The distance type determines whether a pair of points is an inlier or outlier. This parameter applies when you set the Method parameter to LMedS, RANSAC, MSAC, or LTS.

> **Note** For faster computations, set this parameter to `Algebraic`. For a geometric distance, set this parameter to `Sampson`.

Data Types
char

>   **Default:** `Sampson`

'DistanceThreshold' - Distance threshold for finding outliers
`0.01` (default)

Distance threshold for finding outliers, specified as the comma-separated pair consisting of 'DistanceThreshold' and a positive value. This parameter applies when you set the Method parameter to RANSAC or MSAC.

>   **Default:** `0.01`

'Confidence' - Desired confidence for finding maximum number of inliers
`99` (default) | scalar

Desired confidence for finding maximum number of inliers, specified as the comma-separated pair consisting of 'Confidence' and a percentage scalar value in the range (0 100). This parameter applies when you set the Method parameter to RANSAC or MSAC.

>   **Default:** `99`

'InlierPercentage' - Minimum percentage of inliers in input points
scalar | `50` (default)

Minimum percentage of inliers in input points, specified as the comma-separated pair consisting of 'InlierPercentage' and percentage scalar value in the range (0 100). Specify the minimum

percentage of inliers in `matchedPoints1` and `matchedPoints2`. This parameter applies when you set the `Method` parameter to `LTS`.

Default: `50`

'ReportRuntimeError' - Report runtime error
true (default) | false

Report runtime error, specified as the comma-separated pair consisting of 'ReportRuntimeError' and a logical value. Set this parameter to `true` to report run-time errors when the function cannot compute the fundamental matrix from `matchedPoints1` and `matchedPoints2`. When you set this parameter to `false`, you can check the `status` output to verify validity of the fundamental matrix.

Default: `true`

Output Arguments

F - Fundamental matrix
3-by-3 matrix

Fundamental matrix, returns as a 3-by-3 matrix that is computed from the points in the inputs `matchedPoints1` and `matchedPoints2`.

inliersIndex - Inliers index
M-by-1 logical vector

Inliers index, returned as an *M*-by-1 logical index vector. An element set to `true` means that the corresponding indexed matched points in `matchedPoints1` and `matchedPoints2` were used to compute the fundamental matrix. An element set to `false` means the indexed points were not used for the computation.

Data Types
logical

status - Status code
0 | 1 | 2 | 3 | 4

Status code, returned as one of the following possible values:

status	Value
0:	No error.
1:	Either `matchedPoints1` or `matchedPoints2` or both, are not *M*-by-2 matrices.
2:	`matchedPoints1` and `matchedPoints2` do not have the same number of points.
3:	`matchedPoints1` and `matchedPoints2` do not contain enough points. `Norm8Point`, `RANSAC`, and `MSAC` require at least 8 points, `LMedS` 16 points, and `LTS` requires `ceil(800/InlierPercentage)`.
4:	Not enough inliers found.

Data Types
`int32`

Examples

Compute Fundamental Matrix

Use the RANSAC method to compute the fundamental matrix.

The RANSAC method requires that the input points are already putatively matched. We can, for example, use the `matchFeatures` function for this. Using the RANSAC algorithm eliminates any outliers which may still be contained within putatively matched points.

1

```
load stereoPointPairs
fRANSAC = estimateFundamentalMatrix(matchedPoints1,matchedPoints2,'Met
```

Use the Least Median of Squares Method to Find Inliers

Find inliers and compute the fundamental matrix.

1

Begin by loading the putatively matched points.

estimateFundamentalMatrix

```
load stereoPointPairs
[fLMedS, inliers] = estimateFundamentalMatrix(matchedPoints1,matchedPoint
```

2

Load the stereo images.

```
I1 = imread('viprectification_deskLeft.png');
I2 = imread('viprectification_deskRight.png');
```

3

Show the putatively matched points.

```
figure;
showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2,'montage','Plc
title('Putative point matches');
```

4

Show the inlier points.

```
figure;
showMatchedFeatures(I1, I2, matchedPoints1(inliers,:),matchedPoints2(inli
title('Point matches after outliers were removed');
```

Use the Normalized Eight-Point Algorithm to Compute the Fundamental Matrix

Compute the fundamental matrix for input points which do not contain any outliers.

1

```
load stereoPointPairs
inlierPts1 = matchedPoints1(knownInliers, :);
inlierPts2 = matchedPoints2(knownInliers, :);
fNorm8Point = estimateFundamentalMatrix(inlierPts1, inlierPts2,'Method','
```

Algorithms

Computing the Fundamental Matrix

This function computes the fundamental matrix using the normalized eight-point algorithm [1]

When you choose the Norm8Point method, the function uses all points in matchedPoints1 and matchedPoints2 to compute the fundamental matrix.

When you choose any other method, the function uses the following algorithm to exclude outliers and compute the fundamental matrix from inliers:

1 Initialize the fundamental matrix, F, to a 3-by-3 matrix of zeros.

2 Set the loop counter n, to zero, and the number of loops N, to the number of random trials specified.

3 Loop through the following steps while $n < N$:

a Randomly select 8 pairs of points from matchedPoints1 and matchedPoints2.

b Use the selected 8 points to compute a fundamental matrix, f, by using the normalized 8-point algorithm.

c Compute the fitness of f for all points in matchedPoints1 and matchedPoints2.

d If the fitness of f is better than F, replace F with f.

For RANSAC and MSAC, update N.

e $n = n + 1$

Number of Random Samplings for RANSAC and MSAC Methods

The RANSAC and MSAC methods update the number of random trials N, for every iteration in the algorithm loop. The function resets N, according to the following:

$$N = \min(N, \; \frac{\log(1-p)}{\log(1-r)}).$$

Where, p represents the confidence parameter you specified, and r is calculated as follows:

$$\sum_{i}^{N} \text{sgn}(du_{i,v_i}),t) / N, \text{ where } \text{sgn}(a,b) = 1 \text{ if } a \le b \text{ and } 0 \text{ otherwise.}$$

Distance Types

The function provides two distance types, algebraic distance and Sampson distance, to measure the distance of a pair of points according to a fundamental matrix. The following equations are used for each type, with u representing matchedPoints1 and v representing matchedPoints2.

Algebraic distance:

$$d(u_i, v_i) = (v_i F u_i^T)^2$$

Sampson distance:

$$d(u_i, v_i) = (v_i F u_i^T)^2 \left[\frac{1}{(F u_i^T)_1^2 + (F u_i^T)_2^2} + \frac{1}{(v_i F)_1^2 + (v_i F)_2^2} \right.$$

where i represents the index of the corresponding points, and $\left(F u_i^T \right)_j^2$, the square of the j-th entry of the vector $F u_i^T$.

Fitness of Fundamental Matrix for Corresponding Points

The following table summarizes how each method determines the fitness of the computed fundamental matrix:

Method	Measure of Fitness
LMedS	$median(d(u_i, v_i); i = 1 : N)$, the number of input points. The smaller the value, the better the fitness.
RANSAC	$\sum\limits^{N} \text{sgn}(du_i, v_i), t) / N$, where $\text{sgn}(a, b) = 1$ if $a \le b$ and 0 otherwise, t represents the specified threshold. The greater the value, the better the fitness.
MSAC	$\sum\limits^{N} \min(d(u_i, v_i), t)$. The smaller the value, the better the fitness.
LTS	$\sum d(u_i, v_i)$, where Ω is the first lowest value of an (N x q) pair of points. Where q represents the inlier percentage you specified. The smaller the value, the better the fitness.

References

[1] Hartley, R., A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2003.

[2] Rousseeuw, P., A. Leroy, *Robust Regression and Outlier Detection*, John Wiley & Sons, 1987.

[3] Torr, P. H. S., and A. Zisserman, *MLESAC: A New Robust Estimator with Application to Estimating Image Geometry*, Computer Vision and Image Understanding, 2000.

See Also

epipolarline | extractFeatures | matchFeatures | estimateUncalibratedRectification | detectSURFFeatures | detectHarrisFeatures | detectMSERFeatures | detectMinEigenFeatures | detectFASTFeatures

estimateFundamentalMatrix

Related Examples
- "Stereo Image Rectification"

Purpose   Estimate geometric transform from matching point pairs

Syntax   tform = estimateGeometricTransform(matchedPoints1,
                    matchedPoints2,transformType)

                [tform,inlierpoints1,
                    inlierpoints2] = estimateGeometricTransform(matchedPoints1
                    ,matchedPoints2,transformType)
                [___ ,status] = estimateGeometricTransform(matchedPoints1,
                    matchedPoints2,transformType)

                [___] = estimateGeometricTransform(matchedPoints1,
                    matchedPoints2,transformType, Name,Value)

Description tform =
                estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType)
                returns a 2-D geometric transform object, tform. The tform
                object maps the inliers in matchedPoints1 to the inliers in
                matchedPoints2.

                The matchedPoints1 and matchedPoints2 inputs can be
                cornerPoints objects, SURFPoints objects, MSERRegions objects, or
                M-by-2 matrices of [x,y] coordinates. You can set the transformType
                input to either 'similarity', 'affine', or 'projective'.

                [tform,inlierpoints1,inlierpoints2] =
                estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType)
                returns the corresponding inlier points in inlierpoints1 and
                inlierpoints2.

                [___ ,status] =
                estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType)
                returns a status code of 0, 1, or 2. If you do not request the status
                code output, the function returns an error for conditions that
                cannot produce results.

estimateGeometricTransform

[__] =
estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType,
Name,Value) uses additional options specified by one or more
Name,Value pair arguments.

Input Arguments

matchedPoints1 - Matched points from image 1

cornerPoints object | SURFPoints object | MSERRegions object |
M-by-2 matrix of [x,y] coordinates

Matched points from image 1, specified as either a cornerPoints
object, SURFPoints object, MSERRegions object, or an *M*-by-2 matrix of
[x,y] coordinates. The function excludes outliers using the M-estimator
SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a
variant of the Random Sample Consensus (RANSAC) algorithm.

matchedPoints2 - Matched points from image 2

cornerPoints object | SURFPoints object | MSERRegions object |
M-by-2 matrix of [x,y] coordinates

Matched points from image 2, specified as either a cornerPoints
object, SURFPoints object, MSERRegions object, or an *M*-by-2 matrix of
[x,y] coordinates. The function excludes outliers using the M-estimator
SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a
variant of the Random Sample Consensus (RANSAC) algorithm.

transformType - Transform type

'similarity' | 'affine' | 'projective'

Transform type, specified as one of three character strings. You can set
the transform type to either 'similarity', 'affine', or 'projective'.

Data Types
char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can

specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: 'Confidence',99 sets the confidence value to 99.

'MaxNumTrials' - Maximum random trials
1000 (default) | positive integer

Maximum number of random trials for finding the inliers, specified as
the comma-separated pair consisting of 'MaxNumTrials' and a positive
integer scalar. Increasing this value improves the robustness of the
results at the expense of additional computations.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 |
uint16 | uint32 | uint64

'Confidence' - Confidence of finding maximum number of inliers
99 (default) | positive numeric scalar

Confidence of finding the maximum number of inliers, specified as the
comma-separated pair consisting of 'Confidence' and a percentage
numeric scalar in the range (0 100). Increasing this value improves the
robustness of the results at the expense of additional computations.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 |
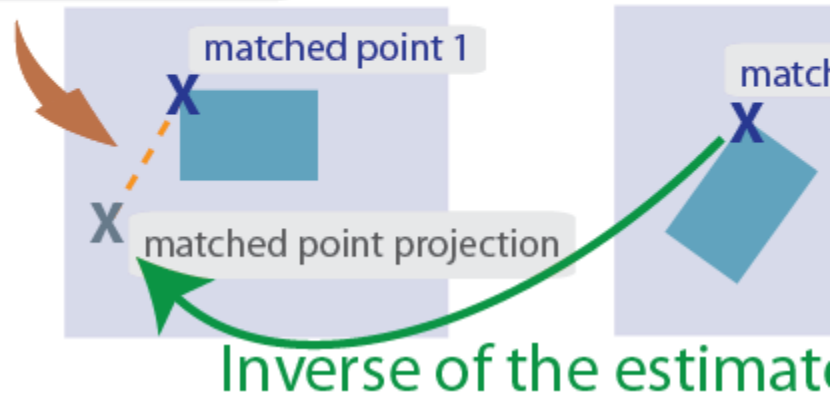uint16 | uint32 | uint64

'MaxDistance' - Maximum distance from point to projection
1.5 (default) | positive numeric scalar

Maximum distance in pixels, from a point to the projection of its
corresponding point, specified as the comma-separated pair consisting
of 'MaxDistance' and a positive numeric scalar. The corresponding
projection is based on the estimated transform.

estimateGeometricTransform

Distance in pixels between the point in image 1 and the projection of the corresponding point from image 2.

matched point 1

matched point projection

Inverse of the estimate

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Output Arguments

tform - Geometric transformation
affine2d object | projective2d object

Geometric transformation, returned as either an affine2d object or a projective2d object.

The returned geometric transformation matrix maps the inliers in matchedPoints1 to the inliers in matchedPoints1. When you set the transformType input to either 'similarity' or 'affine', the function returns an affine2d object. Otherwise, it returns a projective2d object.

status - Status code
0 | 1 | 2

Status code, returned as the value 0, 1, or 2.

status	Description
0	No error.
1	matchedPoints1 and matchedPoints2 inputs do not contain enough points.
2	Not enough inliers found.

If you do not request the status code output, the function will throw an error for the two conditions that cannot produce results.

Data Types
double

inlierpoints1 - Inlier points in image 1
inlier points

Inlier points in image 1, returned as the same type as the input matching points.

inlierpoints2 - Inlier points in image 2
inlier points

Inlier points in image 2, returned as the same type as the input matching points.

Examples　　**Recover a Transformed Image Using SURF Feature Points**

Read and display an image and a transformed image.

```
original  = imread('cameraman.tif'); imshow(original); title('Base ima
distorted = imresize(original, 0.7); distorted = imrotate(distorted, 3
figure; imshow(distorted); title('Transformed image');
```

Detect and extract features from both images.

```
ptsOriginal  = detectSURFFeatures(original);
```

```
ptsDistorted = detectSURFFeatures(distorted);
[featuresOriginal,  validPtsOriginal] = extractFeatures(original,  ptsOr
[featuresDistorted, validPtsDistorted] = extractFeatures(distorted, ptsDi
```

Match features.

```
index_pairs = matchFeatures(featuresOriginal, featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));
figure; showMatchedFeatures(original,distorted,matchedPtsOriginal,matched
title('Matched SURF points, including outliers');
```

Exclude the outliers, and compute the transformation matrix.

```
[tform,inlierPtsDistorted,inlierPtsOriginal] = estimateGeometricTransform
figure; showMatchedFeatures(original,distorted,inlierPtsOriginal,inlierPt
title('Matched inlier points');
```

Recover the original image from the distorted image.

```
geoTransformer = vision.GeometricTransformer;
Ir = step(geoTransformer, im2single(distorted), tform);
figure; imshow(Ir); title('Recovered image');
```

References

[1] Hartley, R., and A. Zisserman, "Multiple View Geometry in Computer Vision," *Cambridge University Press*, 2003.

[2] Torr, P. H. S., and A. Zisserman, "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry," *Computer Vision and Image Understanding*, 2000.

See Also   cp2tformcornerPoints **|** MSERRegions **|** SURFPoints **|**
| estimateFundamentalMatrix | detectSURFFeatures
| detectMinEigenFeatures | detectFASTFeatures |
detectMSERFeatures | extractFeatures | matchFeatures

Purpose   Uncalibrated stereo rectification

Syntax   ```
                   [T1,T2] =
                   estimateUncalibratedRectification(F,inlierPoints1,
                       inlierPoints2,imagesize)
                   ```

**Description**    ```
                   [T1,T2] =
                   estimateUncalibratedRectification(F,inlierPoints1,inlierPoints2,images
                   ```
                   returns projective transformations for rectifying stereo images. This
                   function does not require either intrinsic or extrinsic camera
                   parameters. The input points can be *M*-by-2 matrices of *M* number of [x
                   y] coordinates, or SURFPoints, MSERRegions, or cornerPoints object. F
                   is a 3-by-3 fundamental matrix for the stereo images.

Tips   • An epipole may be located in the first image or the second image.
                     Applying the output uncalibrated rectification of T1 (or T2) to image
                     1 (or image 2) may result in an undesired distortion. You can check
                     for an epipole within an image by applying the isEpipoleInImage
                     function.

Input   **F - Fundamental matrix for the stereo images**
Arguments   3-by-3 matrix

                   Fundamental matrix for the stereo images, specified as a 3-by-3
                   fundamental matrix. The fundamental matrix satisfies the following
                   criteria:

                   If P_1, a point in image 1, corresponds to P_2, a point in image 2, then:
                   $[P_2,1] * F * [P_1,1]' = 0$

                   F must be double or single.

                   inlierPoints1 - Coordinates of corresponding points
                   M-by-2 matrix | SURFPoints object | MSERRegions object |
                   cornerPoints object

Coordinates of corresponding points in image one, specified as an
M-by-2 matrix of M number of [x y] coordinates, or as a SURFPoints,
MSERRegions, or cornerPoints object.

inlierPoints2 - Coordinates of corresponding points

M-by-2 matrix | SURFPoints object | MSERRegions object |
cornerPoints object

Coordinates of corresponding points in image one, specified as an
M-by-2 matrix of M number of [x y] coordinates, or as a SURFPoints,
MSERRegions, or cornerPoints object.

imagesize - Input image size

single | double | integer

Second input image size, specified as a double, single, or integer value
and in the format returned by the size function. The size of input
image 2 corresponds to inlierPoints2.

**Output
Arguments**

T1 - Projective transformation one

Projective transformation, returned as a 3-by-3 matrix describing the
projective transformations for input image T1.

T2 - Projective transformation two

Projective transformation, returned as a 3-by-3 matrix describing the
projective transformations for input image T2.

Examples

Rectify Stereo Images

Rectify stereo images using the projective transformation.

1

Load the stereo images and feature points which are already matched.

```
I1 = imread('yellowstone_left.png');
I2 = imread('yellowstone_right.png');
load yellowstone_inlier_points;
```

2

Display point correspondences. Notice that the matching points are in different rows, indicating that the stereo pair is not rectified.

```
showMatchedFeatures(I1, I2, inlier_points1, inlier_points2, 'montage')
title('Original images and inlier feature points');
```

3

Compute the fundamental matrix from the corresponding points.

```
f = estimateFundamentalMatrix(inlier_points1, inlier_points2, 'Method
```

4

Compute the rectification transformations.

```
[t1, t2] = estimateUncalibratedRectification(f, inlier_points1, inlier
```

5

Set the size and position of the rectified images.

```
[w, h]  = deal(720, 620);  % Size of the rectified image
[x0, y0] = deal(-120, -30); % Upper-left corner of the rectified image
xLim = [0.5, w+0.5] + x0;
yLim = [0.5, h+0.5] + y0;
outputView = imref2d([h,w], xLim, yLim);
```

6

Rectify the stereo images using projective transformations t1 and t2.

```
I1Rect = imwarp(I1, projective2d(t1), 'OutputView', outputView);
I2Rect = imwarp(I2, projective2d(t2), 'OutputView', outputView);
```

7

Transform the points to visualize them together with the rectified images.

estimateUncalibratedRectification

```
pts1Rect = transformPointsForward(projective2d(t1), inlier_points1);
pts2Rect = transformPointsForward(projective2d(t2), inlier_points2);
```

8

Compensate for the shift in the coordinate system origin.

```
pts1Rect = bsxfun(@minus, pts1Rect, [xO, yO]);
pts2Rect = bsxfun(@minus, pts2Rect, [xO, yO]);
```

9

Notice that the inlier points now reside on the same rows. This also
means that the epipolar lines are parallel to the x-axis.

```
figure; showMatchedFeatures(I1Rect, I2Rect, pts1Rect, pts2Rect, 'montage'
title('Rectified images and the corresponding feature points');
```

See Also　　detectHarrisFeatures | detectMinEigenFeatures |
detectHarrisFeatures | vision.GeometricTransformer |
estimateFundamentalMatrix | extractFeatures | isEpipoleInImage
| matchFeatures | size

Tutorials　　• Image Rectification

Purpose   Extract interest point descriptors

Syntax   [features,validPoints] = extractFeatures(I,points)
                 [features,validPoints] =
                 extractFeatures(I,points,Name,Value)

Description [features,validPoints] = extractFeatures(I,points) returns
                 extracted feature vectors, also known as descriptors, and their
                 corresponding locations, from a binary or intensity image. The function
                 derives the descriptors from pixels surrounding an interest point. These
                 pixels describe and match features specified by a single-point location.
                 Each single- point specifies the center location of a neighborhood. The
                 method used for descriptor extraction depends on the class of the input
                 points. The table below shows the descriptor extraction method for
                 each class of points.

Class of Points	Descriptor extraction method
SURFPoints object	Speeded-Up Robust Features (SURF)
MSERRegions object	Speeded-Up Robust Features (SURF)
cornerPoints	Fast Retina Keypoint (FREAK)
M-by-2 matrix of [x y] coordinates	Simple square neighborhood around [x y] point locations

[features,validPoints] =
extractFeatures(I,points,Name,Value) uses additional options
specified by one or more Name,Value pair arguments.

**Input
Arguments**   **I - Input image**
                 binary | 2-D grayscale image

                 Input image, specified as either a binary or 2-D grayscale image.

Data Types
`logical | single | double | int16 | uint8 | uint16`

points - Center location point
`SURFPoints` object | `MSERRegions` object | `cornerPoints` object | *M*-by-2 matrix of [x y] coordinates

Center location point of a square neighborhood, specified as either a `SURFPoints`, `MSERRegions`, or `cornerPoints` object, or an *M*-by-2 matrix of *M* number of [x y] coordinates. The function extracts descriptors from the square neighborhoods that are fully contained within the image boundary. Only square neighborhoods fully contained in the image are used to determine the output valid points.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Method','Block'` specifies the `Block` method for descriptor extraction.

'Method' - Descriptor extraction method
FREAK | SURF | Block | Auto (default)

Descriptor extraction method, specified as the comma-separated pair consisting of 'Method' and the string 'FREAK', 'SURF', 'Block', or 'Auto'.

The table below describes how the function implements these methods.

Method	Feature vector (descriptor)
FREAK	Fast Retina Keypoint (FREAK)
SURF	Speeded-Up Robust Features (SURF). When SURF is used to extract descriptors, the Orientation property of returned validPoints of the SURFPoints object to the orientation of extracted features, in radians. This is useful for visualizing the SURF descriptor orientation.
Block	Simple square neighbhorhood. The Block method extracts only the neighborhoods fully contained within the image boundary. Therefore, the output, validPoints may contain fewer points than the input POINTS.
Auto	The function selects the method based on the class of the input points. For a cornerPoints input, the function implements the FREAK method.
	For a SURFPoints or MSERRegions input, the function implements the SURF method. When you use an MSERRegions object with the SURF method, the Centroid property of the object is used to extract SURF descriptors. The Axes property of the object is used to select the scale of the SURF descriptors such that the circle representing the feature has an area proportional to the MSER ellipse area. The scale is calculated as 1/4*sqrt((majorAxes/2).*(minorAxes/2)) and saturated to 1.6 as required by the SURFPoints object. The Orientation of the object is not used.

extractFeatures

Method	Feature vector (descriptor)
	For an *M*-by-2 matrix of [x y] coordinates input, the function implements the Block method.

Default: 'Auto'

'BlockSize' - Block size
odd integer scalar | 11 (default)

Block size, specified as the comma-separated pair consisting of
'BlockSize' and an odd integer scalar. This value defines the local
square neighborhood **BlockSize**-by-**BlockSize**, centered at each
interest point. This option only applies when the function implements
the Block method.

Default: 11

'SURFSize' - Length of feature vector
64 (default) | 128

Length of the SURF feature vector (descriptor), specified as the
comma-separated pair consisting of 'SURFSize' and either the

integer scalar 64 or 128. This option only applies when the function
implements the SURF method. The higher **SURFSize** of 128 provides
greater accuracy, but effects the feature matching speed.

Default: 64

Output Arguments

features - Feature vectors
M-by-*N* matrix

Feature vectors, returned as a binaryFeatures object or an *M*-by-*N*
matrix of *M* feature vectors also known as descriptors. Each descriptor
is of length *N*.

validPoints - Valid points

M-by-2 matrix

Valid points associated with each output feature vector (descriptor) in `features`, returned as an *M*-by-2 matrix of [x y] coordinates. The function extracts *M* number of descriptors from the valid interest points fully contained within the image.

Examples   **Extract Corner Features From an Image**

Extract corner features, and plot interest points with overlay of valid interest points.

Read the image.

```
I = imread('cameraman.tif');
```

Find and extract features.

```
corners = detectHarrisFeatures(I);
[features, valid_corners] = extractFeatures(I, corners);
```

Display image

```
figure; imshow(I); hold on
```

Plot valid corner points.

```
plot(valid_corners);
```

Extract SURF features

Extract SURF features and visualize the ten strongest SURF features. Include scale, and orientation which was determined during the descriptor extraction process.

Read image

```
I = imread('cameraman.tif');
```

extractFeatures

Find and extract features.

```
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I, points);
```

Display and plot ten strongest SURF features.

```
figure; imshow(I); hold on;
plot(valid_points.selectStrongest(10),'showOrientation',true);
```

Extract MSER features

Extract MSER features and visualize SURF features corresponding to
the MSER ellipse centers along with scale and orientation.

Read image.

```
I = imread('cameraman.tif');
```

Find features using MSER with SURF feature descriptor.

```
regions = detectMSERFeatures(I);
[features, valid_points] = extractFeatures(I, regions);
```

Display SURF features correspondign to the MSER ellipse centers.

```
figure; imshow(I); hold on;
plot(valid_points,'showOrientation',true);
```

References   [1] G. Bradski and A. Kaehler, *Learning OpenCV : Computer Vision
with the OpenCV Library*, O'Reilly, Sebastopol, CA, 2008.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool,
*SURF: Speeded Up Robust Features", Computer Vision and Image
Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008

See Also   vision.CornerDetector | matchFeatures | detectSURFFeatures |
detectMSERFeatures | detectHarrisFeatures | detectFASTFeatures

| detectMinEigenFeatures | MSERRegions | SURFPoints |
binaryFeatures

integralFilter

Purpose	Integral image filter
Syntax	`J = integralFilter(intI,H)`

Description `J = integralFilter(intI,H)` filters an image, given its integral image, `intI`, and filter object, H. The `integralKernel` function returns the filter object used for the input to the `integralFilter`. The `integralFilter` returns only the parts of correlation that are computed without padding. The resulting size of the filtered output J equals:

$$\text{size(J)} = \text{size(intI)} - \text{H.Size}$$

This function uses integral images for filtering an image with box filters. You can obtain the integral image, `intI`, by calling the `integralImage` function. The filter size does not affect the speed of the filtering operation. Thus, the `integralFilter` function is ideally suited to use for fast analysis of images at different scales, as demonstrated by the Viola-Jones algorithm [1].

Tips Because the `integralFilter` function uses correlation for filtering, the filter is not rotated before computing the result.

Input Arguments

intI

Integral image. You can obtain the integral image, `intI`, by calling the `integralImage` function. The class for this value can be `double` or `single`.

H

Filter object. You can obtain the filter object, H, by calling the `integralKernel` function.

Output Arguments

J

Filtered image. The filtered image, J, returns only the parts of correlation that are computed without padding. The resulting size of the filtered output equals:

$$\text{size}(J) = \text{size}(\text{int}I) - \text{H.Size}$$

Because the function uses correlation for filtering, the filter is not rotated before computing the result.

Examples   **Blur an Image Using a Filter**

Use an averaging filter to blur an image.

Read the input image and compute the integral image.

```
I = imread('pout.tif');
figure; imshow(I);
intImage = integralImage(I);
```

Create a 7-by-7 average filter object.

```
avgH = integralKernel([1 1 7 7], 1/49);
```

Blur the image using the integral filter.

```
J = integralFilter(intImage, avgH);
```

Cast the result back to the same class as the input image and display the results.

```
J = uint8(J); % cast the result back to same class as I
figure; imshow(J);
```

Find Vertical and Horizontal Edges in an Image

Construct Harr-like wavelet filters to find vertical and horizontal edges in an image.

Read the input image and compute the integral image.

```
I = imread('pout.tif');
intImage = integralImage(I);
```

integralFilter

Construct Harr-like wavelet filters. Use the dot notation to find the
vertical filter.

```
horiH = integralKernel([1 1 4 3; 1 4 4 3], [-1, 1]); % horizontal filter
vertH = horiH.';
```

Display the horizontal filter.

```
imtool(horiH.Coefficients, 'InitialMagnification','fit');
```

Compute the filter responses.

```
horiResponse = integralFilter(intImage, horiH);
vertResponse = integralFilter(intImage, vertH);
```

Display the results.

```
figure; imshow(horiResponse, []); title('Horizontal edge responses');
figure; imshow(vertResponse, []); title('Vertical edge responses');
```

References [1] Viola, Paul and Michael J. Jones, "Rapid Object Detection using
a Boosted Cascade of Simple Features", *Proceedings of the 2001
IEEE Computer Society Conference on Computer Vision and Pattern
Recognition*, 2001. Volume: 1, pp.511–518.

See Also cumsum | integralImageintegralKernel **|**

**Related
Examples**
• "Compute an Integral Image" on page 3-81

Purpose   Compute integral image

Syntax   J = integralImage(I)

Description   J = integralImage(I) computes an integral image of the input
                   intensity image, I. The function zero-pads the top and left side of the
                   output integral image, J. The resulting size of the output integral image
                   equals:

```
size(J) = size(I) +1
```

                   Such sizing facilitates easy computation of pixel sums along all image
                   boundaries. The integral image, J, is essentially a padded version of the
                   value cumsum(cumsum(I)).

                   An *integral image* lets you rapidly calculate summations over image
                   subregions. Use of integral images was popularized by the Viola-Jones
                   algorithm [1]. Integral images facilitate summation of pixels and can be
                   performed in constant time, regardless of the neighborhood size.

Input   **I**
Arguments
                   Intensity image. This value can be any numeric class.

Output   **J**
Arguments
                   Integral image. The function zero-pads the top and left side of the
                   integral image. The class of the output is double.

Examples   **Compute an Integral Image**

                   Compute the integral image and use it to compute the sum of pixels
                   over a rectangular region of an intensity image.

```
I = magic(5)
```

                   Define rectangular region as [startingRow, startingColumn, endingRow,
                   endingColumn].

```
[sR sC eR eC] = deal(1, 3, 2, 4);
```

Compute the sum over the region using the integral image.

```
J = integralImage(I);
regionSum = J(eR+1,eC+1) - J(eR+1,sC) - J(sR,eC+1) + J(sR,sC)
```

Algorithms   **How Integral Image Summation Works**

An integral image helps you rapidly calculate summations over image
subregions. Every pixel in an integral image is the summation of the
pixels above and to the left of it.
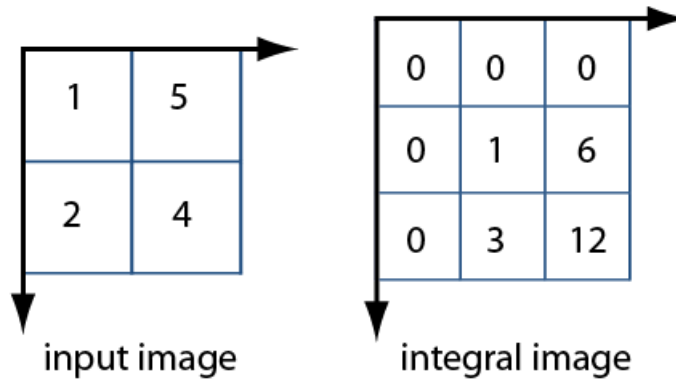
To calculate the summation of a subregion of an image, you can use the
corresponding region of its integral image. For example, in the input
image below, the summation of the shaded region becomes a simple
calculation using four reference values of the rectangular region in the
corresponding integral image. The calculation becomes, $46 - 22 - 20 + 10 = 14$. The calculation subtracts the regions above and to the left
of the shaded region. The area of overlap is added back to compensate
for the double subtraction.

input image   integral image

In this way, you can calculate summations in rectangular regions rapidly, irrespective of the filter size.

References   [1] Viola, Paul and Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. Volume: 1, pp.511–518.

See Also   cumsum | integralFilterintegralKernel |

Related Examples

- "Blur an Image Using a Filter" on page 3-79
- "Find Vertical and Horizontal Edges in an Image" on page 3-79

insertMarker

Purpose	Insert markers in image or video

Syntax	`RGB = insertMarker(I,position)`
	`RGB = insertMarker(I,position,marker)`
	`RGB = insertMarker(___,Name,Value)`

Description	`RGB = insertMarker(I,position)` returns a truecolor image with inserted plus (+) markers. The input image, `I`, can be either a truecolor or grayscale image. You draw the markers by overwriting pixel values. The input `position` can be either an *M*-by-2 matrix of *M* number of [x y] pairs or a `cornerPoints` object. `RGB = insertMarker(I,position,marker)` returns a truecolor image with the `marker` type of markers inserted. `RGB = insertMarker(___,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

I - Input image
truecolor image | 2-D grayscale image

Input image, specified in truecolor or 2-D grayscale.

Data Types
single | double | int16 | uint8 | uint16

position - Position of marker
matrix | vector

Position of marker, specified as either an *M*-by-2 matrix of *M* number of [*x y*] pairs or a `cornerPoints` object. The center positions for the markers are defined by either the [*xy*] pairs of the matrix or by the `position.Location` property of the `cornerPoints` object.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

marker - Type of marker

'plus' (default) | Character string

Type of marker, specified as a string. The string can be full text or corresponding symbol.

String	Symbol String
'circle'	'o'
'x-mark'	'x'
'plus'	'+'
'star'	'*'
'square'	's'

Data Types
char

> **Default:** 'plus'

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'Color','yellow' specifies yellow for the marker color.

'Size' - Size of marker

3 (default)

Size of marker in pixels, specified as the comma-separated pair consisting of 'Size' and a scalar value in the range [1, inf).

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 |
uint16 | uint32 | uint64

insertMarker

Default: 3

'Color' - Marker color

'green' (default) | character string | cell array of character strings | vector | matrix

Marker color, specified as the comma-separated pair consisting of 'Color' and either a string, cell array of strings, vector, or matrix. You can specify a different color for each marker or one color for all markers.

To specify a color for each marker, set Color to a cell array of color strings or an *M*-by-3 matrix of *M* number of RGB (red, green, and blue) color values.

To specify one color for all markers, set Color to either a color string or an [R G B] vector. The [R G B] vector contains the red, green, and blue values.

Supported color strings are: 'blue', 'green', 'red', 'cyan', 'magenta', 'yellow','black', and 'white'.

Data Types
cell | char | uint8 | uint16 | int16 | double | single

Default: 'green'

Output Arguments

RGB - Output image

truecolor image

Output image, returned as a truecolor image.

Examples

Draw Markers on an Image

Read the image.

```
I = imread('peppers.png');
```

Draw a plus (+).

```
RGB = insertMarker(I, [147 279]);
```

Draw four x-marks.

```
pos   = [120 248;195 246;195 312;120 312];
color = {'red', 'white', 'green', 'magenta'};
RGB = insertMarker(RGB, pos, 'x', 'color', color, 'size', 10);
```

Display the image.

```
imshow(RGB);
```

See Also insertObjectAnnotationcornerPoints **|** | insertShape |
insertText

Related Examples
- "Insert a Circle and Filled Shapes on an Image" on page 3-98
- "Insert Numbers and Strings on an Image" on page 3-104

insertObjectAnnotation

Purpose   Annotate true-color or grayscale image or video stream

Syntax
```
RGB = insertObjectAnnotation(I,Shape,Position,Label)
RGB = insertObjectAnnotation(I,Shape,Position,Label, Name,
    Value)

insertObjectAnnotation(I,'rectangle',Position,Label)
insertObjectAnnotation(I,'circle',Position,Label)
```

Description   `RGB = insertObjectAnnotation(I,Shape,Position,Label)` returns a true-color image annotated with `Shape` and `Label` . The input image, `I`, can be either a true-color or grayscale image. You can set the shape to `rectangle` or `circle`, and insert it at the location specified by the matrix, `Position`. The input, `Label`, can be a numeric vector of length M. It can also be a cell array of ASCII strings of length M, In this array, M is the number of shape positions. You can also specify a single label for all shapes as a numeric scalar or string.

`RGB = insertObjectAnnotation(I,Shape,Position,Label, Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

`insertObjectAnnotation(I,'rectangle',Position,Label)` inserts rectangles and corresponding labels at the location indicated by the position matrix. The `Position` input must be an M-by-4 matrix, where each row, M, specifies a rectangle as a four-element vector, [*x y width height*]. The elements *x* and *y* indicate the upper-left corner of the rectangle, and the *width* and *height* specify the size.

`insertObjectAnnotation(I,'circle',Position,Label)` inserts circles and corresponding labels at the location indicated by the position matrix. The `Position` input must be an M-by-3 matrix, where each row, M, specifies a circle as a three-element vector, [*x y r*]. The elements *x* and *y* indicate the center of the circle, and *r* specifies the radius.

Input Arguments

I - True-color or grayscale image
image

True-color or grayscale image, specified as an image or video stream.

Data Types
double | single | uint8 | uint16 | int16

Shape - Rectangle or circle annotation
'rectangle' | 'circle'

Rectangle or circle annotation, specified as a string indicating the annotation shape.

Data Types
char

Position - Location and size of the annotation shape
M-by-3 matrix | *M*-by-4 matrix

Location and size of the annotation shape, specified as an *M*-by-3 or *M*-by-4 matrix. When you specify a rectangle, the position input matrix must be an *M*-by-4 matrix. Each row, *M*, specifies a rectangle as a four-element vector, [*x y width height*]. The elements, *x* and *y*, indicate the upper-left corner of the rectangle, and the *width* and *height* specify the size.

When you specify a circle, the position input matrix must be an *M*-by-3 matrix, where each row, *M*, specifies a three-element vector [*x y r*]. The elements, *x* and *y*, indicate the center of the circle and *r* specifies the radius.

Example: position = [50 120 75 75]

A rectangle with top-left corner located at *x*=50, *y*=120, with a width and height of 75 pixels.

Example: position = [96 146 31]

A circle with center located at *x*=96, *y*=146 and a radius of 31 pixels.

Example: `position = [23 373 60 66;35 185 77 81;77 107 59 26]`

Location and size for three rectangles.

Label - A string label to associate with a shape

numeric scalar | numeric vector | ASCII string | cell array of ASCII strings

A string label to associate with a shape, specified as a numeric vector or a cell array. The input can be a numeric vector of length *M*. It can also be a cell array of ASCII strings of length *M*, where *M* is the number of shape positions. You can also specify a single numeric scalar or string label for all shapes.

Example: label = [5 10], where the function marks the first shape with the label, 5, and the second shape with the label, 10.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Color' - Color for shape and corresponding label text box

'yellow' (default) | color string | [R G B] vector | Cell array | *M*-by-3 matrix

Color for shape and for corresponding label text box. You can specify this value as the comma-separated pair consisting of `'Color'` and either a string, an [R G B] vector, a cell array, or an *M*-by-3 matrix. To specify one color for all shapes, set this parameter to either a string or an [R G B] vector. To specify a color for each of the *M* shapes, set this parameter to a cell array of *M* strings. Alternatively, you can specify an *M*-by-3 matrix of RGB values for each annotation. RGB values must be in the range of the input image data type.

Available color strings are: `'blue'`, `'green'`, `'red'`, `'cyan'`, `'magenta'`, `'black'`, and `'white'`.

Data Types
char | uint8 | uint16 | int16 | double | single | cell

'TextColor' - Color of text in text label
'black' (default) | color string | [R G B] vector | Cell array | *M*-by-3 matrix

Color of text in text label. You can specify this value as the comma-separated pair consisting of `'TextColor'` and either a string, an [R G B] vector, a cell array, or an *M*-by-3 matrix. To specify one color for all text, set this parameter to either a string or an [R G B] vector. To specify a color for each of the *M* text labels, set this parameter to a cell array of *M* strings. Alternatively, you can specify an *M*-by-3 matrix of RGB values for each annotation. RGB values must be in the range of the input image data type.

Available color strings are: `'black'`,`'blue'`, `'green'`, `'red'`, `'cyan'`, `'magenta'`, `'yellow'`, and `'white'`.

Data Types
char | uint8 | uint16 | int16 | double | single | cell

'TextBoxOpacity' - Opacity of text label box background
0.6 (default) | range of [0 1]

Opacity of text label box background. You can specify this value as the comma-separated pair consisting of `'TextBoxOpacity'` and a scalar defining the opacity of the background of the label text box. Specify this value in the range of 0 to 1.

Data Types
double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

'FontSize' - Label text font size
12 (default) | integer in the range of [8 72]

Label text font size, specified as the comma-separated pair consisting of `'FontSize'` and an integer corresponding to points in the range of [8 72].

Data Types
double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Output Arguments

RGB - True-color or grayscale image with annotation
Output image

True-color or grayscale image with annotation, returned as an image or video stream.

Data Types
double | single | uint8 | uint16 | int16

Examples

Object Annotation with Rectangles, Numbers, and Strings

Read and display an image.

```
I = imread('board.tif');
figure, imshow(I);
```

Create labels with floating point numbers that represent detection confidence for each shape.

```
label_str = cell(3,1);
conf_val = [85.212 98.76 78.342];
for ii=1:3
    label_str{ii} = ['Confidence: ' num2str(conf_val(ii),'%0.2f') '%'];
end
```

Set the position and size of the three rectangles in the format [*x y width height*].

```
position = [23 373 60 66;35 185 77 81;77 107 59 26];
```

Insert the annotation.

```
RGB = insertObjectAnnotation(I, 'rectangle', position, label_str,'Text
```

Display the annotated image.

```
figure, imshow(RGB), title('Annotated chips');
```

Object Annotation with Circles and Integer Numbers

Read and display an image.

```
I = imread('coins.png');
figure, imshow(I);
```

Set up the circle annotations with labels. The first two arguments are
the (*x*,*y*) coordinates for the center of the circle, the third argument is
the radius.

```
position = [96 146 31;236 173 26];
label = [5 10];
```

Insert the annotation.

```
RGB = insertObjectAnnotation(I, 'circle', position, label,'Color', {'c
```

Display the annotated image.

```
figure, imshow(RGB), title('Annotated coins');
```

See Also   insertMarker | insertText | insertShape

insertShape

Purpose	Insert shapes in image or video

Syntax

```
RGB = insertShape(I,shape,position)
RGB = insertShape( ___ ,Name,Value
```

Description

RGB = insertShape(I,shape,position) returns a truecolor image with shape inserted. The input image, I, can be either a truecolor or grayscale image. You draw the shapes by overwriting pixel values.

RGB = insertShape(___ ,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

Input Arguments

I - Input image

truecolor | 2-D grayscale image

Input image, specified in truecolor or 2-D grayscale.

Example:

Data Types

single | double | int16 | uint8 | uint16

shape - Type of shape

character string

Type of shape, specified as a string. The string can be, 'Rectangle', 'FilledRectangle', 'Line', 'Polygon', 'FilledPolygoon', 'Circle', or 'FilledCircle'.
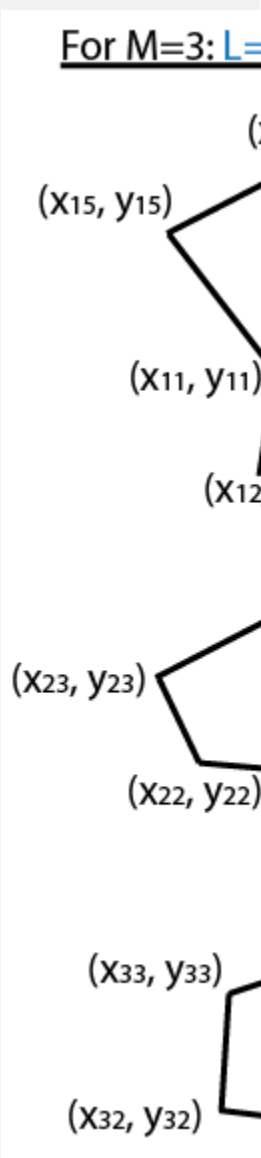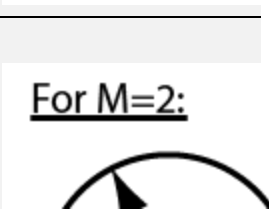
Data Types

char

position - Position of shape

matrix | vector | cell array

Position of shape, specified according to the type of shape, described in the table.

Shape	Position	Shape Drawn
'Rectangle' 'FilledRectangle'	*M*-by-4 matrix where each row specifies a rectangle as $\begin{bmatrix} x & y & width & height \end{bmatrix}$. $$\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$$	For M=2: (x_1, y_1) $height$ $width_1$ (x_2, y_2) $height$ $width_2$
'Line'	For one or more disconnected lines, an *M*-by-4 matrix, where each four-element vector $[x_1, y_1, x_2, y_2]$, describe a line with endpoints, $[x_1\ y_1]$ and $[x_2\ y_2]$. $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & x_{M2} \end{bmatrix}$$	For M=2: (x_{11}, y_{11}) (x_{12}, y_{12}) $(x_{22},$
	For one or more line segments, an *M*-by-2*L* matrix, where each row is a vector representing a polyline with *L* number of vertices. $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} \end{bmatrix}$$. The polyline always contains (*L*-1) number of segments because	For M=2: (x_{14}, y_{14}) L (x_{15}, y_{15}) (x_{11}, y_{11}) **3-95**

Shape	Position	Shape Drawn
'Polygon' 'FilledPolygon'	An M-by-$2L$ matrix, where each row represents a polygon with L number of vertices. Each row of the matrix corresponds to a polygon. For polygons with fewer segments, repeat the ending coordinates to fill the matrix. $$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} \end{bmatrix}$$ You can also specify the shapes as a cell array of M vectors: $\{[x_{11},y_{11},x_{12},y_{12},...,x_{1p},y_{1p}],$ $[x_{21},y_{21},x_{22},y_{22},.....,x_{2q},y_{2q}], ...$ $[x_{M1},y_{M1},x_{M2},y_{M2},.......,x_{Mr},y_{Mr}]\}$ p, q, and r specify the number of vertices.	For M=3: L=5 (x_{14}, y_{14}) (x_{15}, y_{15}) (x_{11}, y_{11}) (x_{12}, y_{12}) $(x_{24}, y_{24}) = ($ (x_{23}, y_{23}) (x_{22}, y_{22}) $(x_{21}, y_{21}$ $(x_{34}$ (x_{33}, y_{33}) (x_{32}, y_{32}) $(x_{31}, y$
'Circle' 'FilledCircle'	An M-by-3 matrix, where each row is a vector specifying a circle as $\begin{bmatrix} x & y & radius \end{bmatrix}$. The $\begin{bmatrix} x & y \end{bmatrix}$ coordinates represent the center of the circle.	For M=2:

Data Types
`single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Color','yellow'` specifies yellow for the shape color.

'Color' - Shape color
`'yellow'` (default) | character string | cell array of character strings | vector | matrix

Shape color, specified as the comma-separated pair consisting of `'Color'` and either a string, cell array of strings, vector, or matrix. You can specify a different color for each shape, or one color for all shapes.

To specify a color for each shape, set `Color` to a cell array of color strings or an *M*-by-3 matrix of *M* number of RGB (red, green, and blue) color values.

To specify one color for all shapes, set `Color` to either a color string or an [R G B] vector. The [R G B] vector contains the red, green, and blue values.

Supported color strings are: `'blue'`, `'green'`, `'red'`, `'cyan'`, `'magenta'`, `'yellow'`,`'black'`, and `'white'`.

Data Types
`cell | char | uint8 | uint16 | int16 | double | single`

> **Default:** `'yellow'`

'Opacity' - Opacity of filled shape
0.6 (default) | range of [0 1]

insertShape

Opacity of filled shape, specified as the comma-separated pair consisting of 'Opacity' and a scalar value in the range [0 1]. The Opacity property applies for the FilledRectangle, FilledPolygon, and FilledCircle shapes.

Data Types
`double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

'SmoothEdges' - Smooth shape edges
true (default) | false

Smooth shape edges, specified as the comma-separated pair consisting of 'SmoothEdges' and a logical value of true or false. A true value enables an anti-aliasing filter to smooth shape edges. This value applies only to nonrectangular shapes. Enabling anti-aliasing requires additional time to draw the shapes.

Data Types
`logical`

Output Arguments

RGB - Output image
truecolor | 2-D grayscale image

Output image, returned as a truecolor image.

Examples

Insert a Circle and Filled Shapes on an Image

Read the image.

```
I = imread('peppers.png');
```

Draw a circle.

```
RGB = insertShape(I, 'circle', [150 280 35]);
```

Draw a filled triangle and a filled hexagon.

```
pos_triangle = [183 297 302 250 316 297];
pos_hexagon = [340 163 305 186 303 257 334 294 362 255 361 191];
RGB = insertShape(RGB,'FilledPolygon',{pos_triangle, pos_hexagon},'Col
```

Display the image.

```
imshow(RGB);
```

See Also   insertObjectAnnotation | insertMarker | insertText

Related   • "Draw Markers on an Image" on page 3-86
Examples   • "Insert Numbers and Strings on an Image" on page 3-104

insertText

Purpose	Insert text in image or video

Syntax	`RGB = insertText(I,position,textstring)`
	`RGB = insertText(I,position,numericvalue)`
	`RGB = insertText( ___ ,Name,Value)`

Description	`RGB = insertText(I,position,textstring)` returns a truecolor image with text strings inserted. The input image, I, can be either a truecolor or grayscale image. The text strings are drawn by overwriting pixel values. `RGB = insertText(I,position,numericvalue)` returns a truecolor image with numeric values inserted. `RGB = insertText( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

I - Input image
truecolor | 2-D grayscale image

Input image, specified in truecolor or 2-D grayscale.

Data Types
single | double | int16 | uint8 | uint16

textstring - Text string
ASCII text string | cell array of ASCII text strings

Text string, specified as a single ASCII text string or a cell array of ASCII strings. The length of the cell array must equal the number of rows in the `position` matrix. If a single text string is provided, it is used for all positions.

Data Types
char

numericvalue - Numeric value text

scalar | vector

Numeric value text, specified as a scalar or a vector. If a scalar value is provided, it is used for all positions. The vector length must equal the number of rows in the position matrix. Numeric values are converted to a string using format '%0.5g', as used by sprintf.

Data Types
char

position - Position of text
vector | matrix

Position of inserted text, specified as an *M*-by-2 matrix of [*x y*] coordinates. Each row represents the [*x y*] coordinate for the AnchorPoint of the text bounding box.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: 'AnchorPoint','LeftTop' specifies the upper-left anchor point of the text bounding box.

'FontSize' - Font size
12 (default)

Font size, specified as the comma-separated pair consisting of 'FontSize' and an integer value in the range [8 72].

Data Types
double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Default: 12

'TextColor' - Text color

'black' (default) | character string | cell array of character strings | vector | matrix

Text color, specified as the comma-separated pair consisting of 'TextColor' and either a string, cell array of strings, vector, or matrix. You can specify a different color for each string or one color for all strings.

To specify a color for each text string, set TextColor to a cell array of *M* number of color strings or an *M*-by-3 matrix of *M* number of RGB (red, green, and blue) string color values.

To specify one color for all text strings, set TextColor to either a color string or an [R G B] vector. The [R G B] vector contains the red, green, and blue values.

RGB values must be in the range of the image data type. Supported color strings are: 'blue', 'green', 'red', 'cyan', 'magenta', 'yellow','black', and 'white'.

Data Types
cell | char | uint8 | uint16 | int16 | double | single

Default: 'black'

'BoxColor' - Text box color

'yellow' (default) | character string | cell array of character strings | vector | matrix

Text box color, specified as the comma-separated pair consisting of 'BoxColor' and either a string, cell array of strings, vector, or matrix. You can specify a different color for each text box, or one color for all of the boxes.

To specify a color for each text box, set BoxColor to a cell array of *M* number of color strings or an *M*-by-3 matrix of *M* number of RGB (red, green, and blue) string color values.

To specify one color for all of the text boxes, set `BoxColor` to either a color string or an [R G B] vector. The [R G B] vector contains the red, green, and blue values.

RGB values must be in the range of the image data type. Supported color strings are: `'blue'`, `'green'`, `'red'`, `'cyan'`, `'magenta'`, `'yellow'`,`'black'`, and `'white'`.

Data Types
`cell | char | uint8 | uint16 | int16 | double | single`

Default: `'yellow'`

'BoxOpacity' - Filled shape opacity
0.6 (default) | range of [0 1]

Filled shape opacity of text box, specified as the comma-separated pair consisting of '`BoxOpacity`' and a scalar value in the range [0 1]. A value of `0` corresponds to a fully transparent text box, and a value of `1` corresponds to a fully opaque text box. When you set the `BoxOpacity` to `0`, no text box appears.

Data Types
`double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

'AnchorPoint' - Text box reference point
'LeftTop' (default)

Text box reference point, specified as the comma-separated pair consisting of '`AnchorPoint`' and a string value. The anchor point defines a relative location on the text box. You can position the text box by placing the anchor point for it at the [x y] coordinate defined by the corresponding `position` for the text. For example, if you want the center of the text box to be at the [x y] coordinate you specifed with the `position` input, then set `AnchorPoint` to `Center`.

Supported positions are `LeftTop`, `LeftCenter`, `LeftBottom`, `CenterTop`, `Center`, `CenterBottom`, `RightTop`, `RightCenter`, and `RightBottom`.

insertText

Data Types
char

Output Arguments

RGB - Output image
truecolor | 2-D grayscale image

Output image, returned as a truecolor image.

Examples

Insert Numeric Values on an Image

Read the image.

```
I = imread('peppers.png');
```

Define position for the text.

```
position =  [1 50; 100 50]; % [x y]
value = [555 pi];
```

Insert text.

```
RGB = insertText(I, position, value, 'AnchorPoint', 'LeftBottom');
```

Display the image.

```
figure, imshow(RGB), title('Numeric values');
```

Insert Numbers and Strings on an Image

Read the image.

```
I = imread('board.tif');
```

Create texts that contain fractions.

```
text_str = cell(3,1);
conf_val = [85.212 98.76 78.342];
for ii=1:3
   text_str{ii} = ['Confidence: ' num2str(conf_val(ii),'%0.2f') '%'];
end
```

Define positions and colors.

```
position = [23 373; 35 185; 77 107]; % [x y]
box_color = {'red','green','yellow'};
```

Insert text.

```
RGB = insertText(I, position, text_str, 'FontSize', 18, 'BoxColor'
```

Display the image.

```
figure, imshow(RGB), title('Board');
```

See Also   insertObjectAnnotation | insertMarker | insertShape

Related Examples
- "Draw Markers on an Image" on page 3-86
- "Insert a Circle and Filled Shapes on an Image" on page 3-98

isEpipoleInImage

Purpose	Determine whether image contains epipole
Syntax	ISIN = isEpipoleInImage(F,imageSize) ISIN = isEpipoleInImage(F',imageSize) [ISIN,EPIPOLE] = isEpipoleInImage(...)

Description

ISIN = isEpipoleInImage(F,imageSize) determines whether the first stereo image associated with the fundamental matrix F contains an epipole. imageSize is the size of the first image, and is in the format returned by the function size.

ISIN = isEpipoleInImage(F',imageSize) determines whether the second stereo image associated with the fundamental matrix F' contains an epipole.

[ISIN,EPIPOLE] = isEpipoleInImage(...) also returns the epipole.

Input Arguments

F

A 3-by-3 fundamental matrix computed from stereo images. If P_1 represents a point in the first image I_1 that corresponds to P_2, a point in the second image I_2, then:

$[P_2,1]$ * F * $[P_1,1]$' = 0

F must be double or single.

imageSize

The size of the image, and in the format returned by the function size.

Output Arguments

ISIN

Logical value set to true when the image contains an epipole, and set to false when the image does not contain an epipole.

EPIPOLE

A 1-by-2 vector indicating the location of the epipole.

Examples   Using stereo point pairs, determine whether image contains epipole and find where that epipole is located:

```
load stereoPointPairs
f = estimateFundamentalMatrix(matchedPoints1,matchedPoints2,'NumTrials
imageSize = [200 300];
[isIn,epipole] = isEpipoleInImage(f,imageSize)
```

Determine whether image contains epipole:

```
f = [0 0 1; 0 0 0; -1 0 0];
imageSize = [200, 300];
[isIn,epipole] = isEpipoleInImage(f',imageSize)
```

See Also   estimateFundamentalMatrix | estimateUncalibratedRectification

isfilterseparable

Purpose	Determine whether filter coefficients are separable
Syntax	S = isfilterseparable(H) [S, HCOL, HROW] = isfilterseparable(H)
Description	S = isfilterseparable(H) takes in the filter kernel *H* and returns 1 (true) when the filter is separable, and 0 (false) otherwise. [S, HCOL, HROW] = isfilterseparable(H) uses the filter kernel, H, to return its vertical coefficients HCOL and horizontal coefficients HROW when the filter is separable. Otherwise, HCOL and HROW are empty.

Definitions

Separable two dimensional filters

Separable two-dimensional filters reflect the outer product of two vectors. Separable filters help reduce the number of calculations required.

A two-dimensional convolution calculation requires a number of multiplications equal to the *width* × *height* for each output pixel. The general case equation for a two-dimensional convolution is:

$$Y(m,n) = \sum_{k}\sum_{l} H(k,l)U(m-k,n-l)$$

If the filter *H* is separable then,

$$H(k,l) = H_{row}(k)H_{col}(l)$$

Shifting the filter instead of the image, the two-dimensional equation becomes:

$$Y(m,n) = \sum_{k} H_{row}(k)\sum_{l} H_{col}(l)U(m-k,n-l)$$

This calculation requires only (width + height) number of multiplications for each pixel.

Input Arguments

H

H numeric or logical, 2-D, and nonsparse.

Output Arguments

HCOL

HCOL is the same data type as input H when H is either single or double floating point. Otherwise, HCOL becomes double floating point. If S is true, HCOL is a vector of vertical filter coefficients. Otherwise, HCOL is empty.

HROW

HROW is the same data type as input H when H is either single or double floating point. Otherwise, HROW becomes double floating point. If S is true, HROW is a vector of horizontal filter coefficients. Otherwise, HROW is empty.

S

Logical variable that is set to true, when the filter is separable, and false, when it is not.

Examples

Determine if the Gaussian filter created using the fspecial function is separable.

```
% Create a gaussian filter
two_dimensional_filter = fspecial('gauss');
% Test with isfilterseparable
[isseparable, hcol, hrow] = ...
isfilterseparable(two_dimensional_filter)
```

When you run this example, notice that hcol*hrow equals the two_dimensional_filter. This result is expected for a Gaussian filter.

Algorithms

The isfilterseparable function uses the singular value decomposition svd function to determine the rank of the matrix.

isfilterseparable

See Also   2-D FIR Filter | svd | rank

Related Links
- MATLAB Central — Separable Convolution

Purpose	Intersection points of lines in image and image border
Syntax	POINTS = lineToBorderPoints(LINES,IMAGESIZE)
Description	POINTS = lineToBorderPoints(LINES,IMAGESIZE) computes the intersection points between one or more lines in an image with the image border.

Input Arguments

LINES

Line matrix. An *M*-by-3 matrix, where each row must be in the format, [*A*,*B*,*C*]. This matrix corresponds to the definition of the line:

$A * x + B * y + C = 0.$

M represents the number of lines.

LINES must be double or single.

IMAGESIZE

Image size. This input must be in the format returned by the size function.

IMAGESIZE must be double, single, or integer.

Output Arguments

POINTS

Output intersection points. An *M*-by-4 matrix. The function returns the matrix in the format of $[x_1, y_1, x_2, y_2]$. In this matrix, $[x_1 \ y_1]$ and $[x_2 \ y_2]$ are the two intersection points. When a line in the image and the image border do not intersect, the function returns [-1,-1,-1,-1].

Examples

Find the intersection points of a line in an image and the image border.

```
% Load and display an image.
  I = imread('rice.png');
  figure; imshow(I); hold on;

% Define a line: 2 * x + y - 300 = 0
```

lineToBorderPoints

```
aLine = [2,1,-300];

% Compute the intersection points of the line and the image border.
points = lineToBorderPoints(aLine, size(I));
line(points([1,3]), points([2,4]));
```

See Also   vision.ShapeInserter | size | line | epipolarLine

Purpose   Find matching features

Syntax

```
indexPairs = matchFeatures(features1,features2)
[indexPairs,matchmetric] =
matchFeatures(features1,features2)
[indexPairs,matchmetric] =
matchFeatures(features1,features2,
    Name,Value)
```

Description   indexPairs = matchFeatures(features1,features2) returns a
P-by-2 matrix, indexPairs, containing *P* pairs of indices. These indices
contain features most likely to correspond between the two input feature
sets. The function requires two inputs, features1 and features2.
The input features can be matrices or a binaryFeatures object.

[indexPairs,matchmetric] =
matchFeatures(features1,features2) also returns the metric values
that correspond to the associated features indexed by indexPairs in a
P-by-1 matrix matchmetric.

[indexPairs,matchmetric] =
matchFeatures(features1,features2,Name,Value) uses additional
options specified by one or more Name,Value pair arguments.

Input Arguments

features1 - Features
M_1-by-*N* matrix | binaryFeatures object

Features set, specified as an M_1-by-*N* matrix, where *N* is the length
of each feature vector or a binaryFeatures object. This object for
binary descriptors is produced using the Fast Retina Keypoint (FREAK)
descriptor.

features2 - Features
M_2-by-*N* matrix | binaryFeatures object

Features set, specified as an M_2-by-N matrix, where N is the length of each feature vector or a `binaryFeatures` object. This object for binary descriptors is produced using the Fast Retina Keypoint (FREAK) descriptor.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Metric','SSD'` specifies the sum of squared differences for the feature matching metric.

'Method' - Matching method
`Threshold | NearestNeighborSymmetric | NearestNeighborRatio` (default)

Matching method, specified as the comma-separated pair consisting of 'Method' and a string. All three methods use the match threshold. Two feature vectors match when the distance between them is less than the threshold set by the `MatchThreshold` parameter. Set the method to one of the following options:

Threshold	Uses only the match threshold. This method can return more than one match for each feature. [3]
NearestNeighborSymmetric	Returns only unique matches in addition to using the match threshold. A feature vector only matches to its nearest neighbor in the other feature set. [3]
NearestNeighborRatio	Eliminates ambiguous matches in addition to using the match threshold. A feature vector is matched to its nearest neighbor in the other feature set, when the nearest

neighbor satisfies a ratio test. The ratio test compares the distances from the feature vector to its first and second nearest neighbors in the other feature set. [2][3]

Default: NearestNeighborRatio

'MatchThreshold' - Threshold

percent value in the range (0, 100) | 10.0 for binary feature vectors; 1.0 non-binary feature vectors (default)

Threshold, specified as a scalar percent value in the range (0,100) for selecting the strongest matches. Matches having a metric more than this percent value from a perfect match are rejected. Increase this value to return more matches.

> **Default:** 10.0 for binary feature vectors; 1.0 non-binary feature vectors

'MaxRatio' - Ratio threshold

ratio in the range (0,1] | 0.6 (default)

Ratio threshold, specified as a scalar value in the range (0,1] for rejecting ambiguous matches. Increase this value to return more matches. You can only use this parameter when you set the Method parameter to NearestNeighborhoodRatio.

> **Default:** 0.6

'Metric' - Feature matching metric

SAD | SSD (default) | normxcorr

Feature matching metric, specified as a string. Possible values for the metric are:

`SAD:`	Sum of absolute differences
`SSD:`	Sum of squared differences
`normxcorr:`	Normalized cross-correlation[1]

When the input feature set, `features1` and `features2` are `binaryFeatures` objects, and the function uses the Hamming distance to compute the similarity metric. This parameter only applies when the input feature set, `features1` and `features2`, are not `binaryFeatures` objects.

Default: `'SSD'`

'Prenormalized' - Prenormalize indicator
true | false (default)

Prenormalize indicator, specified as a logical scalar. Set this value to `true` when the input feature set, `features1` and `features2` are already normalized to unit vectors before matching. When you set this value to `false`, the function normalizes `features1` and `features2`.

When you set this value to `true`, and features are not normalized in advance, the function produces wrong results.

Note This parameter applies only when the input feature set, `features1` and `features2`, are not `binaryFeatures` objects.

Default: false

Output Arguments

indexPairs - Indices to corresponding features
P-by-2 matrix

Indices of corresponding features most likely to exist between the two input feature sets, returned as a *P*-by-2 matrix of *P* number of indices. Each index pair corresponds to a matched feature between `features1` and `features2` inputs. The first element indexes the feature in

features1 that matches the feature in features2 indexed by the second element.
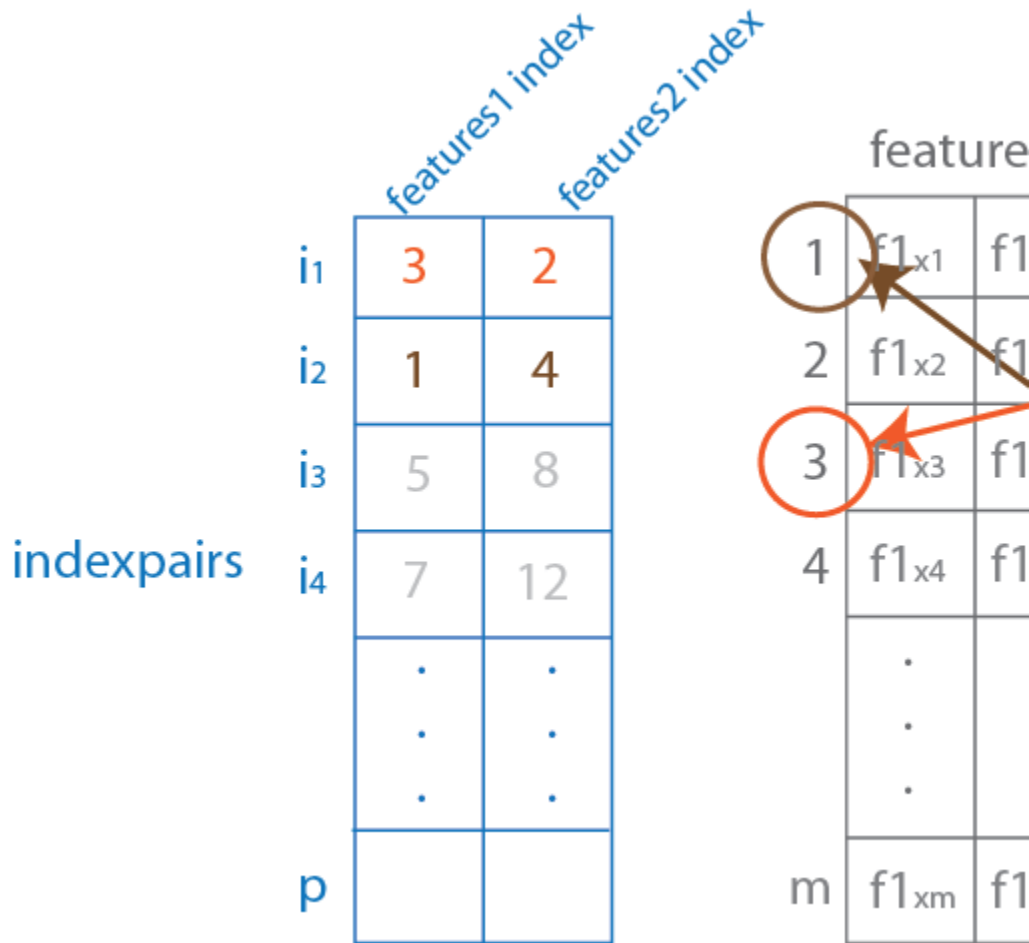
matchmetric - Metric values
```
[0, 2*sqrt(size(features1, 2))] | [0,4] | [-1,1] | [0,
features1.NumBits]
```

Metric values that have been matched, returned as a value within a range calculated based on the metric selected. The match metric value corresponds to the features indexed in the `indexPairs` output matrix. Values vary depending on which feature matching metric you select. When you select either SAD or SSD metrics, the feature vectors are normalized to unit vectors before computation.

The following tables summarize the output match metric ranges and perfect match values.

Metric	Range	Perfect Match Value
SAD	[0, 2*sqrt(size(features1, 2))].	0
SSD	[0,4]	0
normxcorr	[-1,1]	1

The `Hamming` metric cannot be selected, it is invoked automatically when `features1` and `features2` inputs are `binaryFeatures`.

Metric	Range	Perfect Match Value
Hamming	[0, features1.NumBits]	0

Examples **Find Corresponding Interest Points Between a Pair of Images**

Find the corresponding interest points using local neighborhoods and the Harris algorithm.

Read the stereo images.

```
I1 = rgb2gray(imread('viprectification_deskLeft.png'));
I2 = rgb2gray(imread('viprectification_deskRight.png'));
```

Find the corners.

```
points1 = detectHarrisFeatures(I1);
```

```
points2 = detectHarrisFeatures(I2);
```

Extract the neighborhood features.

```
[features1, valid_points1] = extractFeatures(I1, points1);
[features2, valid_points2] = extractFeatures(I2, points2);
```

Match the features.

```
indexPairs = matchFeatures(features1, features2);
```

Retrieve the locations of corresponding points for each image.

```
matched_points1 = valid_points1(indexPairs(:, 1), :);
matched_points2 = valid_points2(indexPairs(:, 2), :);
```

Visualize corresponding points. You can see the effect of translation between the two images despite several erroneous matches.

```
figure; showMatchedFeatures(I1, I2, matched_points1, matched_points2);
```

Use SURF Features to Find Corresponding Points

Find the corresponding points between two images that are rotated and scaled with respect to each other.

Read the two images.

```
I1 = imread('cameraman.tif');
I2 = imresize(imrotate(I1,-20), 1.2);
```

Find the SURF features.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract the features.

```
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
```

matchFeatures

Retrieve the locations of matched points. The SURF feature vectors are already normalized.

```
indexPairs = matchFeatures(f1, f2, 'Prenormalized' true) ;
matched_pts1 = vpts1(indexPairs(:, 1));
matched_pts2 = vpts2(indexPairs(:, 2));
```

Display the matching points. The data still includes several outliers, but you can see the effects of rotation and scaling on the display of matched features.

```
figure; showMatchedFeatures(I1,I2,matched_pts1,matched_pts2);
legend('matched points 1','matched points 2');
```

References

[1] .2005 [43] Lewis J.P.: Fast Normalized Cross Correlation. Available from:http://www.idiom.com/~zilla/Papers/nvisionInterface/nip.html

[2] Lowe, David G. "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, Volume 60, Number 2, Pages 91–110.

[3] Mikolajczyk, K. and C. Shmid, "A Performance Evaluation of Local Descriptors," *Journal IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 27, Issue 10, October 2005, pp1615–1630.

See Also   extractFeaturesbinaryFeatures **|** | detectHarrisFeatures
| detectMSERFeatures | estimateFundamentalMatrix |
estimateGeometricTransform

Purpose   View video from MATLAB workspace, multimedia file, or Simulink model

Syntax
```
mplay
mplay('filename.avi')
mplay('filename.avi',FPS)
mplay(A)
mplay(A,FPS)
mplay({line_handles})
mplay({'block',PORT})
```

Description   mplay opens an MPlay GUI that allows you to view video from files, the MATLAB workspace, or Simulink signals. The MPlay GUI does not play audio.

mplay('filename.avi') connects the MPlay GUI to the specified AVI file.

mplay('filename.avi',FPS) plays the specified frame rate in frames per second, (FPS). The FPS value equals that of the frame rate specified in the file.

mplay(A) connects the MPlay GUI to the variable in the MATLAB workspace, A.

mplay(A,FPS) plays the specified frame rate in frames per second, FPS. The FPS value defaults to 20.

mplay({line_handles}) connects the MPlay GUI to one or three Simulink signal lines to display, where all signals must originate from the same block.

mplay({'block',PORT}) connects the MPlay GUI to the output signal of the specified block, 'block', on output port port. All ports on the specified block are selected if PORT is omitted.

Input Arguments

A

A is a variable in the MATLAB workspace, which must have one of the following formats:

- MATLAB movie structure

- Intensity video. This input is an M-by-N-by-T or M-by-N-by-1-by-T array, where the size of each frame is M-by-N and there are T image frames.

- RGB video array. This input is an M-by-N-by-3-by-T array, where the size of each RGB image is M-by-N-by-3 and there are T image frames.

For performance considerations, the video input A data type converts to uint8 as follows:

Supported Data Types	Converted to uint8
double	✓
single	✓
int8	✓
uint8	
int16	✓
uint16	
int32	✓
uint32	✓
Boolean	✓
Fixed point	✓

block

block is a full path to a specified Simulink block. To get the full block path name of the currently selected Simulink block, issue the command mplay({gcb,1}) on the MATLAB command line.

filename.avi

Filename.avi is a specified AVI file.

FPS

FPS stands for frames per second. You can specify the frame rate in frames per second.

line_handles

line_handles are Simulink signal lines. To get the handles to the Simulink signals, line_handles, issue the command mplay({gsl}) on the MATLAB command line.

port

port refers to a Simulink block output port number.

Examples

```
% Create a video
  fig=figure;  % create a video
  set(gca,'xlim',[-80 80],'ylim',[-80 80],'NextPlot', ...
  'replace','Visible','off');
  x = -pi:.1:pi;
  radius = 0:length(x);
  video = []; % initialize video variable

  for i=length(x):-1:1
   patch(sin(x)*radius(i),cos(x)*radius(i), ...
   [abs(cos(x(i))) 0 0]);
   F = getframe(gca);
   video = cat(4,video,F.cdata); % video is MxNx3xT
  end

% Close the figure
  close(fig);

% Display video player. Click play button on display
  mplay(video);
```

Play an AVI file

Play an AVI video file.

mplay

Play an AVI file.

```
mplay('vipwarnsigns.avi');
```

Animate a sequence of images.

```
load cellsequence
mplay(cellsequence,10);
```

Play a video from a Simulink signal.

```
mplay({'vipmplaytut/Rotate'});
```

Alternatives   Access this GUI by selecting **Tools > MPlay Video Viewer**.

See Also   Video Viewer | To Video Display | "MPlay"

Tutorials   • Computer Vision System Toolbox demos

How To   • "View Video with MPlay"

  • "Preview Video in MATLAB using MPlay Function"

Purpose   Display corresponding feature points

Syntax   showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2)
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,
     method)

showMatchedFeatures(___ ,'PlotOptions', {MarkerStyle1,
     MarkerStyle2, LineStyle})

H = showMatchedFeatures(___)

Description   showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2)
displays a falsecolor overlay of images I1 and I2 with a color-coded plot
of corresponding points connected by a line. matchedPoints1 and
matchedPoints2 contain the coordinates of corresponding points in I1
and I2. The input points can be *M*-by-2 matrices of *M* number of [x y]
coordinates, or SURFPoints, MSERRegions, or cornerPoints object.

showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,method)
displays images I1 and I2 using the visualization style specified by the
method parameter.

showMatchedFeatures(___ ,'PlotOptions', {MarkerStyle1,
MarkerStyle2, LineStyle}) lets you specify custom plot options
in a cell array containing three string values. The MarkerStyle1,
MarkerStyle2, and LineStyle string values correspond to the marker
specification in I1, marker specification in I2, and line style and color.
The LineSpec syntax of the plot function defines each of the specifiers.

H = showMatchedFeatures(___) returns the handle to the image
object returned by showMatchedFeatures.

showMatchedFeatures

Input Arguments

I1 - Input image
numeric array

Input image one, specified as a numeric array.

I2 - Input image
numeric array

Input image two, specified as a numeric array.

matchedPoints1 - Coordinates of corresponding points
M-by-2 matrix | SURFPoints object | MSERRegions object | cornerPoints object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [x y] coordinates, or as a SURFPoints, MSERRegions, or cornerPoints object.

matchedPoints2 - Coordinates of corresponding points
M-by-2 matrix | SURFPoints object | MSERRegions object | cornerPoints object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [x y] coordinates, or as a SURFPoints, MSERRegions, or cornerPoints object.

method - Display method
falsecolor (default) | blend | montage

Display style method, specified as one of the following:

falsecolor:	Overlay the images by creating a composite red-cyan image showing I1 as red and I2 as cyan.
blend:	Overlay I1 and I2 using alpha blending.
montage:	Place I1 and I2 next to each other in the same image.

Default: falsecolor

PlotOptions - Line style and color

{'ro','g+','y-'} (default) | cell array

Line style and color options, specified as a cell array containing three string values, {*MarkerStyle1*, *MarkerStyle2*, *LineStyle*}, corresponding to a marker specification in I1, marker specification in I2, and line style and color. The LineSpec syntax of the plot function defines each of the specifiers.

Default: {'ro','g+','y-'}

Output Arguments

H - Handle to image object

handle

Handle to image object, returned as the handle to the image object returned by showMatchedFeatures.

Examples

Find Corresponding Points Between Two Images Using Harris Features

Read images.

```
I1 = rgb2gray(imread('parkinglot_left.png'));
I2 = rgb2gray(imread('parkinglot_right.png'));
```

SURF features.

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
```

Extract features.

```
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
```

Match features.

```
index_pairs = matchFeatures(f1, f2) ;
matched_pts1 = vpts1(index_pairs(1:20, 1));
matched_pts2 = vpts2(index_pairs(1:20, 2));
```

Visualize putative matches.

```
figure; showMatchedFeatures(I1,I2,matched_pts1,matched_pts2,'montage');
```

```
title('Putative point matches');
legend('matchedPts1','matchedPts2');
```

Display Corresponding Points Between Two Rotated and Scaled Images

Use SURF features to find corresponding points between two images rotated and scaled with respect to each other.

Read images.

```
I1 = imread('cameraman.tif');
I2 = imresize(imrotate(I1,-20), 1.2);
```

SURF features.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract features.

```
[f1, vpts1] = extractFeatures(I1, points1);
```

```
[f2, vpts2] = extractFeatures(I2, points2);
```

Match features.

```
index_pairs = matchFeatures(f1, f2) ;
matched_pts1 = vpts1(index_pairs(:, 1));
matched_pts2 = vpts2(index_pairs(:, 2));
```

Visualize putative matches.

```
figure; showMatchedFeatures(I1,I2,matched_pts1,matched_pts2);
title('Putative point matches');
legend('matchedPts1','matchedPts2');
```

See Also   matchFeaturesSURFPoints **|** MSERRegions **|** cornerPoints **|** |
estimateGeometricTransform | legend | imshowpair

trainCascadeObjectDetector

Purpose	Train cascade object detector model

Syntax

```
trainCascadeObjectDetector(outputxmlfilename,
    positiveinstances,negativeimages)
trainCascadeObjectDetector(outputxmlfilename,'resume')

trainCascadeObjectDetector( ___ , Name,Value)
```

Description

trainCascadeObjectDetector(outputxmlfilename,positiveinstances,negativeim
writes a trained cascade detector XML file with the name,
outputxmlfilename. The name specified by the
outputxmlfilename input must have an XML extension. For a
more detailed explanation on how this function works, refer to
"Train a Cascade Object Detector".

trainCascadeObjectDetector(outputxmlfilename,'resume')
resumes an interrupted training session. The outputxmlfilename
input must match the output file name from the interrupted session.
All arguments saved from the earlier session are reused automatically.

trainCascadeObjectDetector(___ , Name,Value) uses additional
options specified by one or more Name,Value pair arguments.

Input Arguments

positiveinstances - Positive samples
array of structs

Positive samples, specified as an array of structs containing string
image file names, and an *M*-by-4 matrix of bounding boxes specifying
object locations in the images.

The struct fields are defined as follows:

imageFilename	A string that specifies the image name. The image can be true color, grayscale, or indexed, in any of the formats supported by `imread`.
objectBoundingBoxes	An *M*-by-4 matrix of *M* bounding boxes. Each bounding box is in the format, [*x y width height*] and specifies an object location in the corresponding image.

The function automatically determines the number of positive samples to use at each of the cascade stages. This value is based on the number of stages and the true positive rate. The true positive rate specifies how many positive samples can be misclassified

Data Types
struct

negativeimages - Negative images
cell array | string

Negative images, specified as either a path to a folder containing images or as a cell array of image file names. Because the images are used to generate negative samples, they must not contain any objects of interest. Instead, they should contain backgrounds associated with the object.

Data Types
char | cell

outputxmlfilename - Trained cascade detector file name
string

Trained cascade detector file name, specified as a string with an XML extension.

Data Types
char

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'FeatureType','Haar'` specifies Haar for the type of features to use.

'ObjectTrainingSize' - Object size for training
`'Auto'` (default) | 2-element vector

Training object size, specified as the comma-separated pair. This pair contains '`ObjectTrainingSize`' and either a 2-element vector [*height*, *width*] or as the string `'Auto'`. Before training, the function resizes the positive and negative samples to `ObjectTrainingSize` in pixels. If you select `'Auto'`, the function determines the size automatically based on the median width-to-height ratio of the positive instances. For optimal detection accuracy, specify an object training size close to the expected size of the object in the image. However, for faster training and detection, set the object training size to be smaller than the expected size of the object in the image.

Data Types
`char | single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

> **Default:** `'Auto'`

'NegativeSamplesFactor' - Negative sample factor
2 (default) | real-valued scalar

Negative sample factor, specified as the comma-separated pair consisting of '`NegativeSamplesFactor`' and a real-valued scalar. The number of negative samples to use at each stage is equal to

NegativeSamplesFactor x [*the number of positive samples used at each stage*].

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

> **Default:** 2

'NumCascadeStages' - Number of cascade stages
20 (default) | positive integer

Number of cascade stages to train, specified as the comma-separated pair consisting of 'NumCascadeStages' and a positive integer. Increasing the number of stages may result in a more accurate detector but also increases training time. More stages may require more training images., because at each stage, some number of positive and negative samples may be eliminated. This value depends on the FalseAlarmRate and the TruePositiveRate. More stages may also allow you to increase the FalseAlarmRate. See the "Train a Cascade Object Detector" tutorial for more details.

Data Types
single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

> **Default:** 20

'FalseAlarmRate' - Acceptable false alarm rate
0.5 (default) | value in the range (0 1]

Acceptable false alarm rate at each stage, specified as the comma-separated pair consisting of 'FalseAlarmRate' and a value in the range (0 1]. The false alarm rate is the fraction of negative training samples incorrectly classified as positive samples.

The overall false alarm rate is calculated using the `FalseAlarmRate` per stage and the number of cascade stages, `NumCascadeStages`:

$$FalseAlarmRate^{NumCascadeStages}$$

Lower values for `FalseAlarmRate` increase complexity of each stage. Increased complexity can achieve fewer false detections but may result in longer training and detection times. Higher values for `FalseAlarmRate` may require a greater number of cascade stages to achieve reasonable detection accuracy.

Data Types
`single | double`

Default: `0.5`

'TruePositiveRate' - Minimum true positive rate
`0.995` (default)

Minimum true positive rate required at each stage, specified as the comma-separated pair consisting of 'TruePositiveRate' and a value in the range (0 1]. The true positive rate is the fraction of correctly classified positive training samples.

The overall resulting target positive rate is calculated using the `TruePositiveRate` per stage and the number of cascade stages, `NumCascadeStages`:

$$TruePositiveRate^{NumCascadeStages}$$

Higher values for `TruePositiveRate` increase complexity of each stage. Increased complexity can achieve a greater number of correct detections but may result in longer training and detection times.

Data Types
`single | double`

Default: `0.995`

'FeatureType' - Feature type
'HOG' (default) | 'LBP' | 'Haar'

Feature type, specified as the comma-separated pair consisting of 'FeatureType' and one of three strings. The possible features types are:

'Haar'[1]	Haar-like features
'LBP'[2]	Local Binary Patterns
'HOG'[3]	Histogram of Oriented Gradients

The function allocates a large amount of memory, this is especially the case for the Haar features. To avoid running out of memory, use this function on a 64-bit operating system with a sufficient amount of RAM.

Data Types
char

> **Default:** 'HOG'

Examples

Train a Stop Sign Detector

Load the positive samples data from a .mat file. The file names and bounding boxes are contained in an array of structures named 'data'.

```
load('stopSigns.mat');
```

Add the image directory to the MATLAB path.

```
imDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondemos','stopS
addpath(imDir);
```

Specify the folder for negative images.

```
negativeFolder = fullfile(matlabroot, 'toolbox', 'vision','visiondemos
```

Train a cascade object detector called 'stopSignDetector.xml' using HOG features.

> **Note** The following command may take several minutes to run.

```
trainCascadeObjectDetector('stopSignDetector.xml', data, negativeFolder,
```

Use the newly trained classifier to detect a stop sign in an image.

```
detector = vision.CascadeObjectDetector('stopSignDetector.xml');
```

Read the test image.

```
img = imread('stopSignTest.jpg');
```

Detect a stop sign.

```
bbox = step(detector, img);
```

Insert bounding boxes and return marked image.

```
detectedImg = insertObjectAnnotation(img, 'rectangle', bbox, 'stop sign')
```

Display the detected stop sign.

```
figure; imshow(detectedImg);
```

Remove the image directory from the path.

```
rmpath(imDir);
```

How Does The `trainCascadeObjectDetector` Function Work?

"Train a Cascade Object Detector"

References

[1] Viola, P., and M. J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features". *Proceedings of the 2001 IEEE Computer Society Conference*. Volume 1, 15 April 2001, pp. I-511–I-518.

[2] Ojala, T., M. Pietikainen, and T. Maenpaa, "Multiresolution Gray-scale and Rotation Invariant Texture Classification With Local Binary Patterns". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Volume 24, No. 7 July 2002, pp. 971–987.

[3] Dalal, N., and B. Triggs, "Histograms of Oriented Gradients for Human Detection". *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.Volume 1, (2005), pp. 886–893.

See Also　　"Train a Cascade Object Detector" **|** insertObjectAnnotation **|** imrectvision.CascadeObjectDetector **|**

External Web Sites　　• Cascade Training GUI

vision.getCoordinateSystem

Purpose   Get coordinate system for Computer Vision System Toolbox.

Syntax   `vision.getCoordinateSystem(coordinateSystem)`

Description   `vision.getCoordinateSystem(coordinateSystem)` returns the
coordinate system string `coordinateSystem` RC or XY.

When you set the coordinate system to 'RC', the function sets the
Computer Vision System Toolbox to the zero-based, [r c] coordinate
system prior to R2011b release.

When you set the coordinate system to 'XY', the function sets the
Computer Vision System Toolbox to the one-based, [x y] coordinate
system.

This function facilitates transition of Computer Vision System
Toolbox MATLAB code prior to R2011b, to newer releases. Refer to
the "Conventions Changed for Indexing, Spatial Coordinates, and
Representation of Geometric Transforms" R2011b release notes for
further details.

Note When you set the coordinate system to `RC`, only your current
MATLAB session persists using the `RC` coordinate system. The
coordinate system resets to `XY` when you restart MATLAB.

Example   Get the Computer Vision System Toolbox coordinate system.

`vision.getCoordinateSystem`

Purpose   Set coordinate system for Computer Vision System Toolbox.

Syntax   vision.setCoordinateSystem(coordinateSystem)

Description vision.setCoordinateSystem(coordinateSystem) sets the coordinate
system based on the string coordinateSystem.

When you set the coordinate system to 'RC', the function sets the
Computer Vision System Toolbox to the zero-based, [r c] coordinate
system prior to R2011b release.

When you set the coordinate system to 'XY', the function sets the
Computer Vision System Toolbox to the one-based, [x y] coordinate
system.

This function facilitates transition of Computer Vision System
Toolbox MATLAB code prior to R2011b, to newer releases. Refer to
the "Conventions Changed for Indexing, Spatial Coordinates, and
Representation of Geometric Transforms" R2011b release notes for
further details.

Note When you set the coordinate system to RC, only your current
MATLAB session persists using the RC coordinate system. The
coordinate system resets to XY when you restart MATLAB.

Example   Set the Computer Vision System Toolbox to the coordinate system prior
to R2011b.

vision.setCoordinateSystem('RC');

Set the Computer Vision System Toolbox to the new coordinate system.

vision.setCoordinateSystem('XY');

visionlib

Purpose   Open top-level Computer Vision System Toolbox Simulink library

Syntax   `visionlib`

Description   `visionlib` opens the top-level Computer Vision System Toolbox block
library model.

Examples   View and gain access to the Computer Vision System Toolbox blocks:

`visionlib`

Alternatives To view and gain access to the Computer Vision System Toolbox blocks
using the Simulink library browser:

- Type `simulink` at the MATLAB command line, and then expand the
  Computer Vision System Toolbox node in the library browser.

- Click the Simulink icon from the MATLAB desktop or from a
  model.

Index

Symbols and Numerics

A

B

C

D

E

F

G

H