Computer Vision System Toolbox™
User's Guide

R2011b

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™ User's Guide

Revision History

July 2004	First printing	New for Version 1.0 (Release 14)
October 2004	Second printing	Revised for Version 1.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 1.1 (Release 14SP2)
September 2005	Online only	Revised for Version 1.2 (Release 14SP3)
November 2005	Online only	Revised for Version 2.0 (Release 14SP3+)
March 2006	Online only	Revised for Version 2.1 (Release 2006a)
September 2006	Online only	Revised for Version 2.2 (Release 2006b)
March 2007	Online only	Revised for Version 2.3 (Release 2007a)
September 2007	Online only	Revised for Version 2.4 (Release 2007b)
March 2008	Online only	Revised for Version 2.5 (Release 2008a)
October 2008	Online only	Revised for Version 2.6 (Release 2008b)
March 2009	Online only	Revised for Version 2.7 (Release 2009a)
September 2009	Online only	Revised for Version 2.8 (Release 2009b)
March 2010	Online only	Revised for Version 3.0 (Release 2010a)
September 2010	Online only	Revised for Version 3.1 (Release 2010b)
April 2011	Online only	Revised for Version 4.0 (Release 2011a)
September 2011	Online only	Revised for Version 4.1 (Release 2011b)

Contents

1 Input, Output, and Conversions

2 Display and Graphics

Input, Output, and Conversions

Learn how to import and export videos, and perform color space and video image conversions.

File Opening, Loading and Saving

In this section...

Import from Video Files

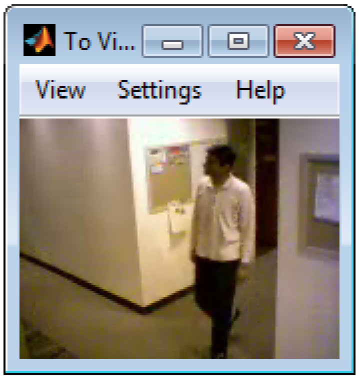
In this example, you use the From Multimedia File source block to import a video stream into a Simulink® model and the To Video Display sink block to view it. This procedure assumes you are working on a Windows platform.

You can open the example model by typing

```
ex_import_mmf
```

at the MATLAB® command line.

1. Run your model.
2. View your video in the To Video Display window that automatically appears when you start your simulation.



Note The video that is displayed in the To Video Display window runs at the frame rate that corresponds to the input sample time. To run the video as fast as Simulink processes the video frames, use the Video Viewer block.

You have now imported and displayed a multimedia file in the Simulink model. In the “Export to Video Files” on page 1-5 example you can manipulate your video stream and export it to a multimedia file.

For more information on the blocks used in this example, see the From Multimedia File and To Video Display block reference pages. To listen to audio associated with an AVI file, use the To Audio Device block in DSP System Toolbox™ software.

Setting Block Parameters for this Example

The block parameters in this example were modified from default values as follows:

Block	Parameter
From Multimedia File	Use the From Multimedia File block to import the multimedia file into the model: • If you do not have your own multimedia file, use the default `vipmen.avi` file, for the **File name** parameter. • If the multimedia file is on your MATLAB path, enter the filename for the **File name** parameter. • If the file is not on your MATLAB path, use the **Browse** button to locate the multimedia file. • Set the **Image signal** parameter to `Separate color signals`. By default, the **Number of times to play file** parameter is set to `inf`. The model continues to play the file until the simulation stops.
To Video Display	Use the To Video Display block to view the multimedia file. • **Image signal**: `Separate color signals` Set this parameter from the **Settings** menu of the display viewer.

Configuration Parameters

You can locate the **Configuration Parameters** by selecting **Configuration Parameters** from the **Simulation** menu. For this example, the parameters on the **Solver** pane, are set as follows:

- **Stop time** = `20`
- **Type** = `Fixed-step`
- **Solver** = `Discrete (no continuous states)`

Export to Video Files

The Computer Vision System Toolbox™ blocks enable you to export video data from your Simulink model. In this example, you use the To Multimedia File block to export a multimedia file from your model. This example also uses Gain blocks from the **Math Operations** Simulink library.

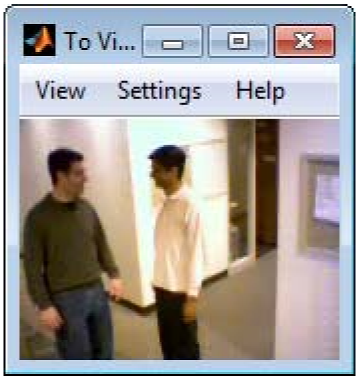
You can open the example model by typing

```
ex_export_mmf
```

at the MATLAB command line.

1 Run your model.

2 You can view your video in the To Video Display window.



By increasing the red, green, and blue color values, you increase the contrast of the video. The To Multimedia File block exports the video data from the Simulink model to a multimedia file that it creates in your current folder.

This example manipulated the video stream and exported it from a Simulink model to a multimedia file. For more information, see the To Multimedia File block reference page.

Setting Block Parameters for this Example

The block parameters in this example were modified from default values as follows:

Block	Parameter
Gain	The Gain blocks are used to increase the red, green, and blue values of the video stream. This increases the contrast of the video: • **Main** pane, **Gain** = `1.2` • **Signal Attributes** pane, **Output data type** = `Inherit:  Same as input`
To Multimedia File	The To Multimedia File block exports the video to a multimedia file: • **Output file name** = `my_output.avi` • **Write** = `Video only` • **Image signal** = `Separate color signals`

Configuration Parameters

You can locate the **Configuration Parameters** by selecting **Configuration Parameters** from the **Simulation** menu. For this example, the parameters on the **Solver** pane, are set as follows:

- **Stop time** = `20`
- **Type** = `Fixed-step`
- **Solver** = `Discrete (no continuous states)`

Batch Process Image Files

A common image processing task is to apply an image processing algorithm to a series of files. In this example, you import a sequence of images from a folder into the MATLAB workspace.

Note In this example, the image files are a set of 10 microscope images of rat prostate cancer cells. These files are only the first 10 of 100 images acquired.

1 Specify the folder containing the images, and use this information to create a list of the file names, as follows:

```
fileFolder = fullfile(matlabroot,'toolbox','images','imdemos');
dirOutput = dir(fullfile(fileFolder,'AT3_1m4_*.tif'));
fileNames = {dirOutput.name}'
```

2 View one of the images, using the following command sequence:

```
I = imread(fileNames{1});
imshow(I);
text(size(I,2),size(I,1)+15, ...
    'Image files courtesy of Alan Partin', ...
    'FontSize',7,'HorizontalAlignment','right');
text(size(I,2),size(I,1)+25, ....
    'Johns Hopkins University', ...
    'FontSize',7,'HorizontalAlignment','right');
```

3 Use a for loop to create a variable that stores the entire image sequence. You can use this variable to import the sequence into Simulink.

```
for i = 1:length(fileNames)
    my_video(:,:,i) = imread(fileNames{i});
end
```

4 Run your model. You can view the image sequence in the Video Viewer window.

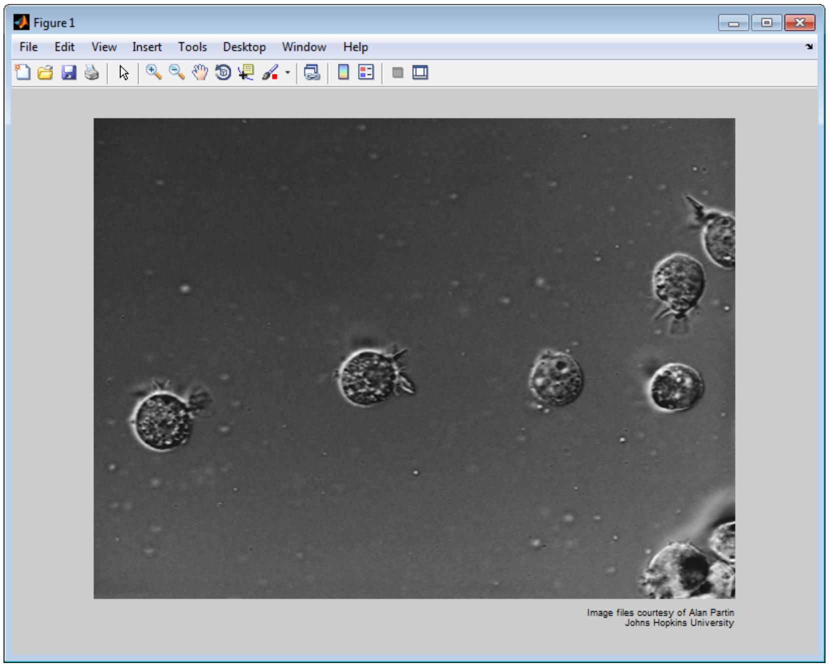


Figure 1
File Edit View Insert Tools Desktop Window Help
Image files courtesy of Alan Partin
Johns Hopkins University

Because the Video From Workspace block's **Sample time** parameter is set to `1` and the "Configuration Parameters" on page 1-10 **Stop time** is set to `10`, the Video Viewer block displays 10 images before the simulation stops.

For more information on the blocks used in this example, see the Video From Workspace and Video Viewer block reference pages. For additional information about batch processing, see the Batch Processing Image Files Using Distributed Computing demo in Image Processing Toolbox. You can run this demo by typing `ipexbatch` at the MATLAB command prompt.

Configuration Parameters

You can locate the **Configuration Parameters** by selecting **Configuration Parameters** from the **Simulation** menu. For this example, the parameters on the **Solver** pane, are set as follows:

- **Stop time** = `10`
- **Type** = `Fixed-step`

- **Solver** = `Discrete (no continuous states)`

Display a Sequence of Images

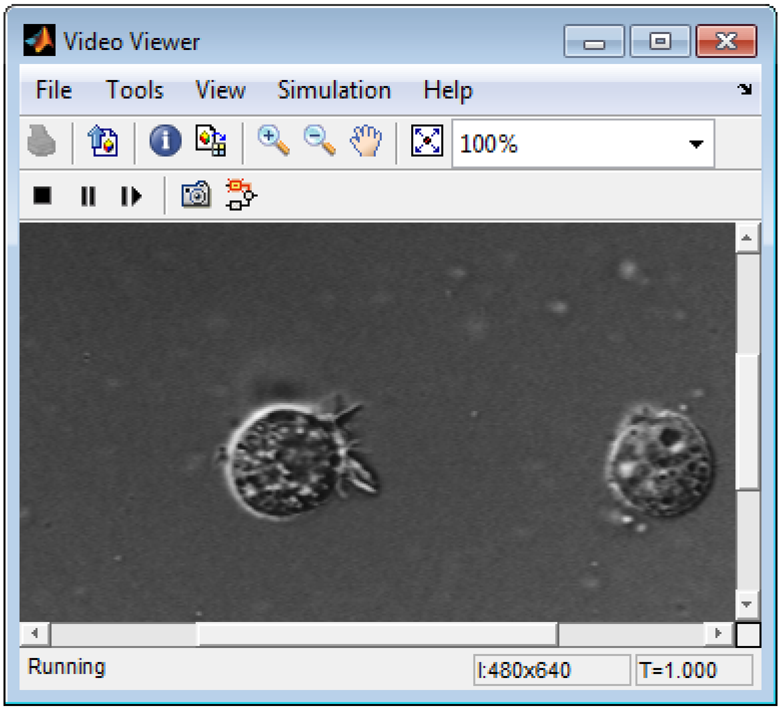
This example displays a sequence of images, which were saved in a folder, and then stored in a variable in the MATLAB workspace. At load time, this model executes the code from the “Batch Process Image Files” on page 1-6 example, which stores images in a workspace variable.

You can open the example model by typing

```
ex_display_sequence_of_images
```

at the MATLAB command line.

1. The Video From Workspace block reads the files from the MATLAB workspace. The **Signal** parameter is set to the name of the variable for the stored images. For this example, it is set to `my_video`.
2. The Video Viewer block displays the sequence of images.
3. Run your model. You can view the image sequence in the Video Viewer window.



4 Because the Video From Workspace block's **Sample time** parameter is set to 1 and the **Stop time** parameter in the configuration parameters, is set to 10, the Video Viewer block displays 10 images before the simulation stops.

Pre-loading Code

To find or modify the pre-loaded code, select the **Callbacks** tab in the **Model Properties** dialog located under **File** > **Model Properties**. For more details on how to set-up callbacks, see "Using Callback Functions".

Configuration Parameters

You can locate the **Configuration Parameters** by selecting **Configuration Parameters** from the **Simulation** menu. For this example, the parameters on the **Solver** pane, are set as follows:

- **Stop time** = 10

- **Type** = `Fixed-step`
- **Solver** = `Discrete (no continuous states)`

Partition Video Frames to Multiple Image Files

In this example, you use the To Multimedia File block, the Enabled Subsystem block, and a trigger signal, to save portions of one AVI file to three separate AVI files.

You can open the example model by typing

```
ex_vision_partition_video_frames_to_multiple_files
```

at the MATLAB command line.

1. Run your model.
2. The model saves the three output AVI files in your current folder.
3. View the resulting files by typing the following commands at the MATLAB command prompt:

   ```
   mplay output1.avi
   mplay output2.avi
   mplay output3.avi
   ```

4. Press the **Play** button on the MPlay GUI.

For more information on the blocks used in this example, see the From Multimedia File, Insert Text, Enabled Subsystem, and To Multimedia File block reference pages.

Setting Block Parameters for this Example

The block parameters in this example were modified from default values as follows:

Block	Parameter
From Multimedia File	The From Multimedia File block imports an AVI file into the model. • Cleared **Inherit sample time from file** checkbox.
Insert Text	The example uses the Insert Text block to annotate the video stream with frame numbers. The block writes the frame number in green, in the upper-left corner of the output video stream. • **Text**: `'Frame %d'` • **Color**: `[0 1 0]` • **Location**: `[10 10]`
To Multimedia File	The To Multimedia File blocks send the video stream to three separate AVI files. These block parameters were modified as follows: • **Output file name**: `output1.avi`, `output2.avi`, and `output3.avi`, respectively • **Write**: `Video only`
Counter	The Counter block counts the number of video frames. The example uses this information to specify which frames are sent to which file. The block parameters are modified as follows: • **Count event**: `Free running` • **Initial count**: `1` • **Output**: `Count` • Cleared **Reset input** check box. • **Sample time**: `1/30` • **Count data type**: `uint16`

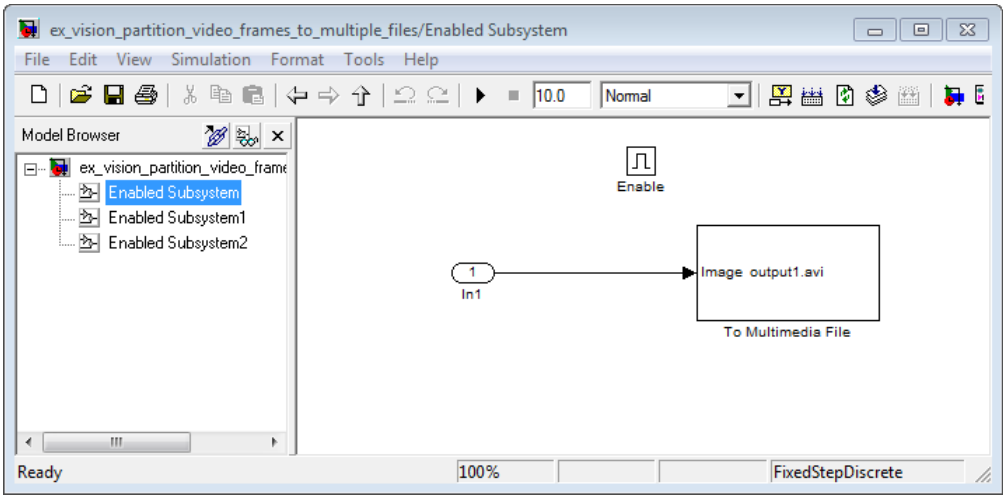
Block	Parameter
Compare to Constant	The Compare to Constant block sends frames 1 to 9 to the first AVI file. The block parameters are modified as follows: • **Operator**: `<` • **Constant value**: `10`
Compare to Constant1 **Compare to Constant2**	The Compare to Constant1 and Compare to Constant2 blocks send frames 10 to 19 to the second AVI file. The block parameters are modified as follows: • **Operator**: `>=` • **Constant value**: `10` The Compare to Constant2 block parameters are modified as follows: • **Operator**: `<` • **Constant value**: `20`
Compare to Constant3	The Compare to Constant3 block send frames 20 to 30 to the third AVI file. The block parameters are modified as follows: • **Operator**: `>=` • **Constant value**: `20`
Compare to Constant4	The Compare to Constant4 block stopa the simulation when the video reaches frame 30. The block parameters are modified as follows: • **Operator**: `==`

Block	Parameter
	• **Constant value**: `30` • **Output data type mode**: `boolean`

Using the Enabled Subsystem Block

Each To Multimedia File block gets inserted into one Enabled Subsystem block, and connected to it's input. You can do this, by double-clicking the Enabled Subsystem blocks, then click-and-drag a To Multimedia File block into it.

Each enabled subsystem should look similar to the subsystem shown in the following figure.



Configuration Parameters

You can locate the **Configuration Parameters** by selecting **Configuration Parameters** from the **Simulation** menu. For this example, the parameters on the **Solver** pane, are set as follows:

- **Solver** pane, **Type** = `Fixed-step`

- **Solver** pane, **Solver** = `Discrete (no continuous states)`

Combine Video and Audio Streams into a Single Video File

In this example, you use the From Multimedia File blocks to import video and audio streams into a Simulink model. You then write the audio and video to a single file using the To Multimedia File block.

You can open the example model by typing

```
ex_combine_video_and_audio_streams
```

on the MATLAB command line.

1. Run your model. The model creates a multimedia file called `output.avi` in your current folder.

2. Play the multimedia file using a media player. The original video file now has an audio component to it.

Setting Up the Video Input Block

The From Multimedia File block imports a video file into the model. During import, the **Inherit sample time from file** check box is deselected, which enables the **Desired sample time** parameter. The other default parameters are accepted.

The From Multimedia File block used for the input video file inherits its sample time from the `vipmen.avi` file. For video signals, the sample time equals the frame period. The frame period is defined as 1/(frame rate). Because the input video frame rate is 30 frames per second (fps), the block sets the frame period to `1/30` or `0.0333` seconds per frame.

Setting Up the Audio Input Block

The From Multimedia File1 block imports an audio file into the model.

The **Samples per audio frame** parameter is set to `735`. This output audio frame size is calculated by dividing the frequency of the audio signal (22050 samples per second) by the frame rate (approximately 30 frames per second).

You must adjust the audio signal frame period to match the frame period of the video signal. The video frame period is `0.0333` seconds per frame. Because the frame period is also defined as the frame size divided by frequency, you can calculate the frame period of the audio signal by dividing the frame size of the audio signal (735 samples per frame) by the frequency (22050 samples per second) to get `0.0333` seconds per frame.

frame period = (*frame size*)/(*frequency*)
frame period = (`735` *samples per frame*)/(22050 *samples per second*)
frame period = `0.0333` seconds per frame

Alternatively, you can verify that the frame period of the audio and video signals is the same using a Simulink Probe block.

Setting Up the Output Block

The To Multimedia File block is used to output the audio and video signals to a single multimedia file. The `Video and audio` option is selected for the **Write** parameter and `One multidimensional signal` for the **Image signal** parameter. The other default parameters are accepted.

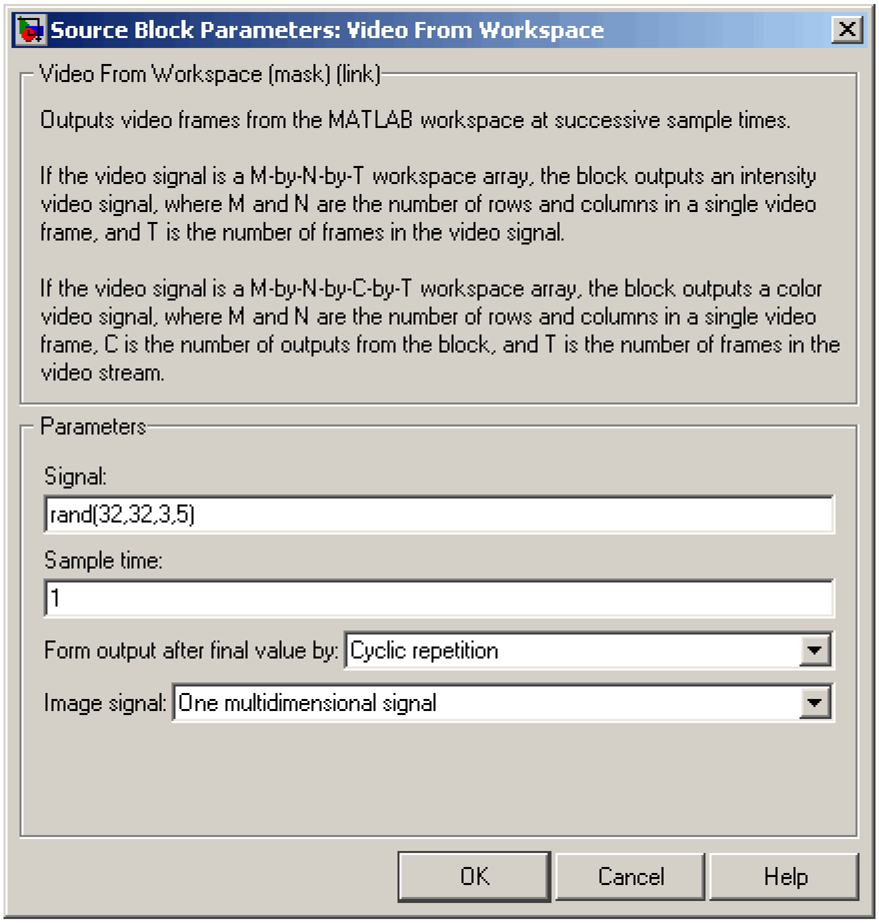
Configuration Parameters

You can locate the Configuration Parameters by selecting **Simulation > Configuration Parameters**. The parameters, on the **Solver** pane, are set as follows:

- **Stop time** = `10`
- **Type** = `Fixed-step`
- **Solver** = `Discrete (no continuous states)`

Import MATLAB Workspace Variables

You can import data from the MATLAB workspace using the Video From Workspace block, which is created specifically for this task.


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

Use the **Signal** parameter to specify the MATLAB workspace variable from which to read. For more information about how to use this block, see the Video From Workspace block reference page.

Import a Live Video Stream

Image Acquisition Toolbox provides functions for acquiring images and video directly into MATLAB and Simulink from PC-compatible imaging hardware. You can detect hardware automatically, configure hardware properties, preview an acquisition, and acquire images and video.

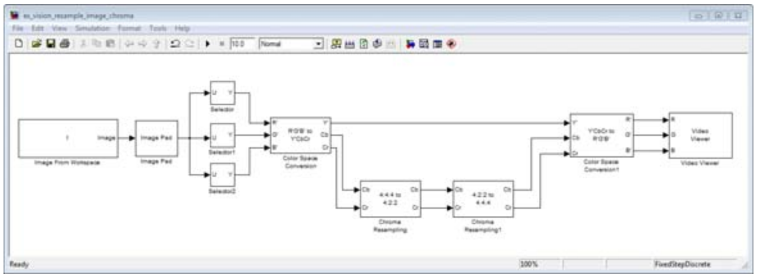
See the live video processing demos to view demos that use the Image Acquisition Toolbox together with Computer Vision System Toolbox blocks. To see the full list of Computer Vision System Toolbox demos, type `visiondemos` at the MATLAB command prompt.

Colorspace Formatting and Conversions

In this section...

Resample Image Chroma

In this example, you use the Chroma Resampling block to downsample the Cb and Cr components of an image. The Y'CbCr color space separates the luma (Y') component of an image from the chroma (Cb and Cr) components. Luma and chroma, which are calculated using gamma corrected R, G, and B (R', G', B') signals, are different quantities than the CIE chrominance and luminance. The human eye is more sensitive to changes in luma than to changes in chroma. Therefore, you can reduce the bandwidth required for transmission or storage of a signal by removing some of the color information. For this reason, this color space is often used for digital encoding and transmission applications.



You can open the example model by typing

```
ex_vision_resample_image_chroma
```

on the MATLAB command line.

1 Define an RGB image in the MATLAB workspace. To do so, at the MATLAB command prompt, type:

```
I= imread('autumn.tif');
```

This command reads in an RGB image from a TIF file. The image `I` is a 206-by-345-by-3 array of 8-bit unsigned integer values. Each plane of this array represents the red, green, or blue color values of the image.

2 To view the image this array represents, at the MATLAB command prompt, type:

```
imshow(I)
```

3 Configure Simulink to display signal dimensions next to each signal line. Select **Format > Port/Signal Displays > Signal Dimensions**.

4 Run your model. The recovered image appears in the Video Viewer window. The Chroma Resampling block has downsampled the Cb and Cr components of an image.

5 Examine the signal dimensions in your model. The Chroma Resampling block downsamples the Cb and Cr components of the image from 206-by-346 matrices to 206-by-173 matrices. These matrices require less bandwidth for transmission while still communicating the information necessary to recover the image after it is transmitted.

Setting Block Parameters for This Example

The block parameters in this example are modified from default values as follows:

Block	Parameter
Image from Workspace	Import your image from the MATLAB workspace. Set the **Value** parameter to `I`.
Image Pad	Change dimensions of the input `I` array from 206-by-345-by-3 to 206-by-346-by-3. You are changing these dimensions because the Chroma Resampling block requires that the dimensions of the input be divisible by 2. Set the block parameters as follows: • **Method** = `Symmetric` • **Add columns to** = `Right` • **Number of added columns** = `1` • **Add row to** = `No padding` The Image Pad block adds one column to the right of each plane of the array by repeating its border values. This padding minimizes the effect of the pixels outside the image on the processing of the image. **Note** When you process video streams, be aware that it is computationally expensive to pad every video frame. You should change the dimensions of the video stream before you process it with Computer Vision System Toolbox blocks.

Block	Parameter
Selector, Selector1, Selector2	Separate the individual color planes from the main signal. Such separation simplifies the color space conversion section of the model. Set the Selector block parameters as follows:**Selector1** • **Number of input dimensions** = `3` • **Index 1** = `Select all` • **Index 2** = `Select all` • **Index 3** = `Index vector (dialog)` and **Index** = `1` **Selector2** • **Number of input dimensions** = `3` • **Index 1** = `Select all` • **Index 2** = `Select all` • **Index 3** = `Index vector (dialog)` and **Index** = `2` **Selector2** • **Number of input dimensions** = `3` • **Index 1** = `Select all` • **Index 2** = `Select all` • **Index 3** = `Index vector (dialog)` and **Index** = `3`
Color Space Conversion	Convert the input values from the R'G'B' color space to the Y'CbCr color space. The prime symbol indicates a gamma corrected signal. Set the **Image signal** parameter to `Separate color signals`.

Block	Parameter
Chroma Resampling	Downsample the chroma components of the image from the 4:4:4 format to the 4:2:2 format. Use the default parameters. The dimensions of the output of the Chroma Resampling block are smaller than the dimensions of the input. Therefore, the output signal requires less bandwidth for transmission.
Chroma Resampling1	Upsample the chroma components of the image from the 4:2:2 format to the 4:4:4 format. Set the **Resampling** parameter to `4:2:2 to 4:4:4`.
Color Space Conversion1	Convert the input values from the Y'CbCr color space to the R'G'B' color space. Set the block parameters as follows: • **Conversion** = `Y'CbCr to R'G'B'` • **Image signal** = `Separate color signals`
Video Viewer	Display the recovered image. Select **File>Image signal** to set **Image signal** to `Separate color signals`.

Configuration Parameters

Open the Configuration dialog box by selecting **Configuration Parameters...** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

Convert Intensity to Binary Images

- "Thresholding Intensity Images Using Relational Operators" on page 1-24
- "Thresholding Intensity Images Using the Autothreshold Block" on page 1-29

Binary images contain Boolean pixel values that are either 0 or 1. Pixels with the value 0 are displayed as black; pixels with the value 1 are displayed as white. Intensity images contain pixel values that range between the minimum and maximum values supported by their data type. Binary images can contain only 0s and 1s, but they are not binary images unless their data type is Boolean.

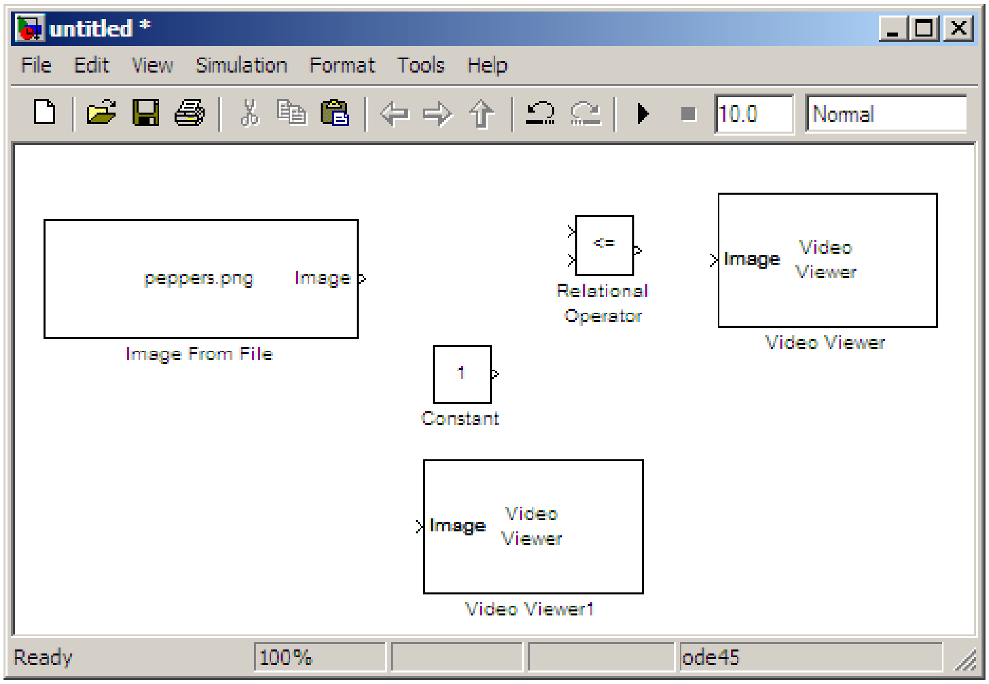
Thresholding Intensity Images Using Relational Operators

You can use the Relational Operator block to perform a thresholding operation that converts your intensity image to a binary image. This example shows you how to accomplish this task:

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Video Viewer	Computer Vision System Toolbox > Sinks	2
Relational Operator	Simulink > Logic and Bit Operations	1
Constant	Simulink > Sources	1

2 Position the blocks as shown in the following figure.

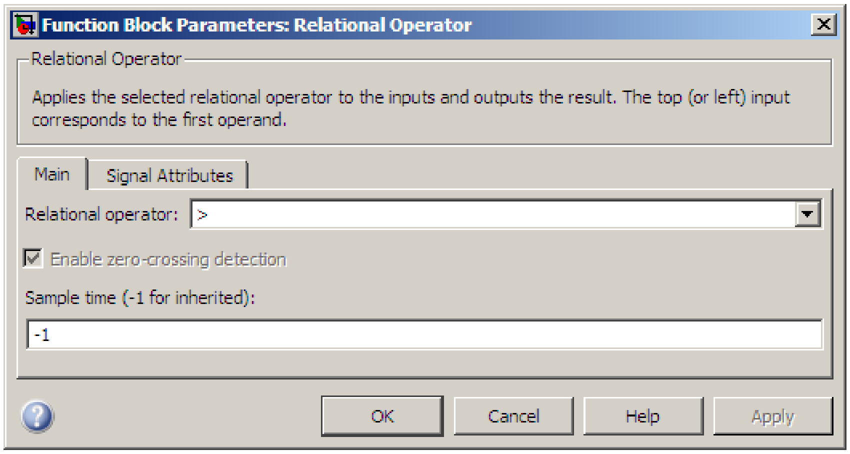


3 Use the Image from File block to import your image. In this example the image file is a 256-by-256 matrix of 8-bit unsigned integer values that range from 0 to 255. Set the **File name** parameter to `rice.png`

4 Use the Video Viewer1 block to view the original intensity image. Accept the default parameters.

5 Use the Constant block to define a threshold value for the Relational Operator block. Since the pixel values range from 0 to 255, set the **Constant value** parameter to `128`. This value is image dependent.

6 Use the Relational Operator block to perform a thresholding operation that converts your intensity image to a binary image. Set the **Relational Operator** parameter to `>`. If the input to the Relational Operator block is greater than 128, its output is a Boolean 1; otherwise, its output is a Boolean 0.


Function Block Parameters: Relational Operator

Relational Operator

Applies the selected relational operator to the inputs and outputs the result. The top (or left) input corresponds to the first operand.

Main | Signal Attributes

Relational operator: >

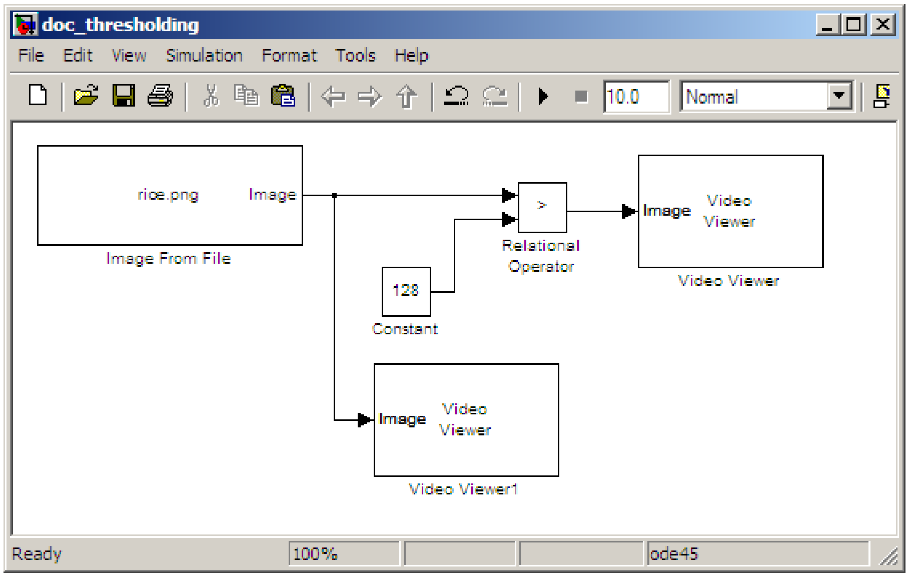
[✓] Enable zero-crossing detection

Sample time (-1 for inherited):

-1

7 Use the Video Viewer block to view the binary image. Accept the default parameters.

8 Connect the blocks as shown in the following figure.

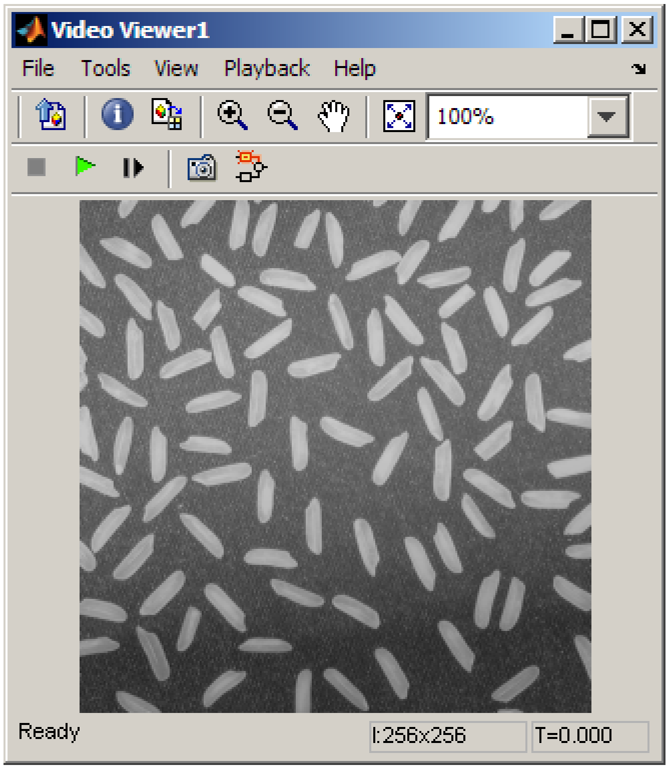


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Model Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

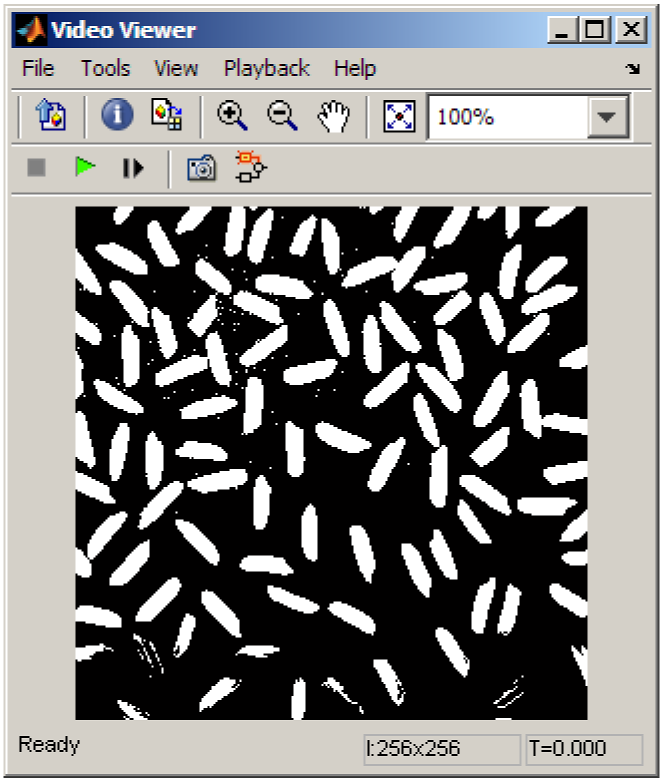
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

10 Run your model.

The original intensity image appears in the Video Viewer1 window.



The binary image appears in the Video Viewer window.



Note A single threshold value was unable to effectively threshold this image due to its uneven lighting. For information on how to address this problem, see "Correct Nonuniform Illumination" on page 7-9.

You have used the Relational Operator block to convert an intensity image to a binary image. For more information about this block, see the Relational Operator block reference page in the Simulink documentation. For additional information, see "Converting Between Image Types" in the Image Processing Toolbox documentation.

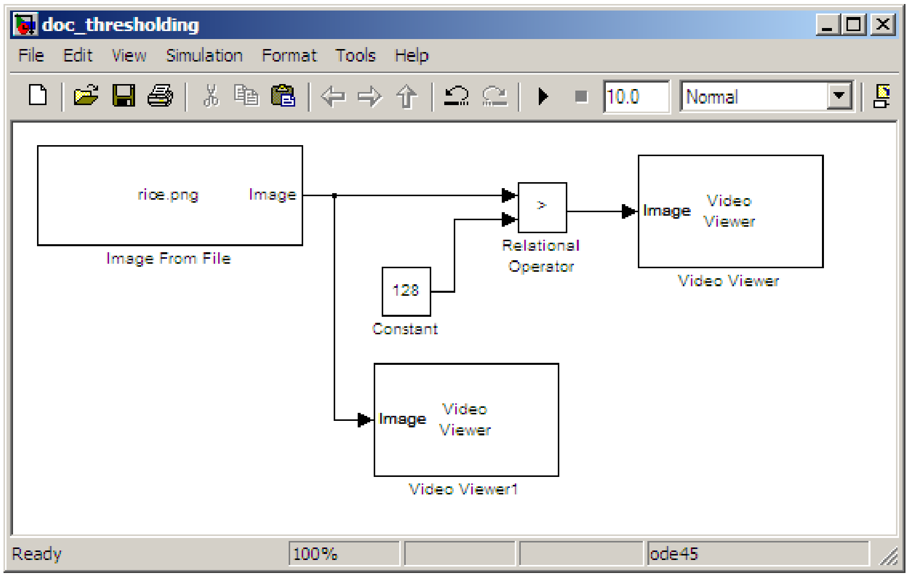
Thresholding Intensity Images Using the Autothreshold Block

In the previous topic, you used the Relational Operator block to convert an intensity image into a binary image. In this topic, you use the Autothreshold block to accomplish the same task. Use the Autothreshold block when lighting conditions vary and the threshold needs to change for each video frame.

1 If the model you created in "Thresholding Intensity Images Using Relational Operators" on page 1-24 is not open on your desktop, you can open an equivalent model by typing

```
ex_thresholding
```

at the MATLAB command prompt.

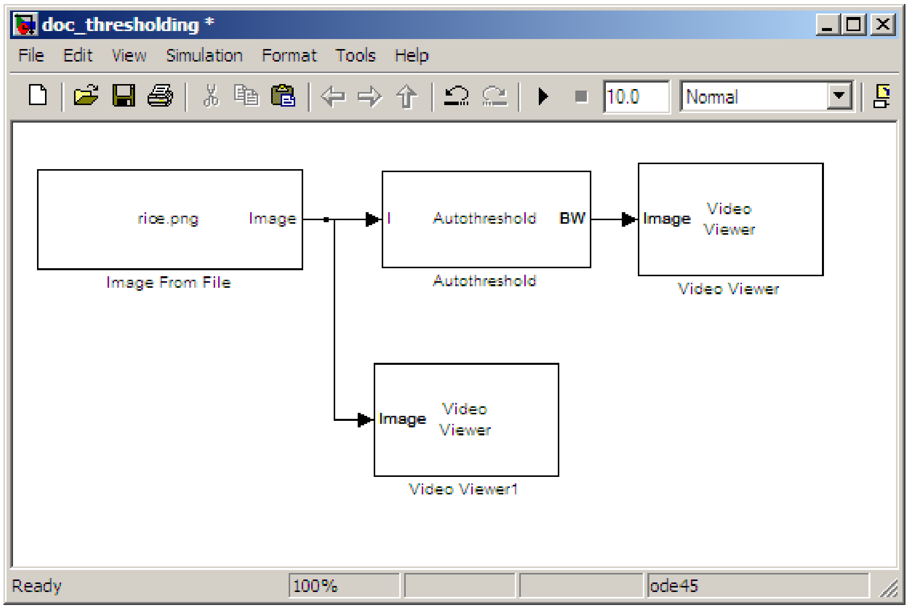


2 Use the Image from File block to import your image. In this example the image file is a 256-by-256 matrix of 8-bit unsigned integer values that range from 0 to 255. Set the **File name** parameter to `rice.png`

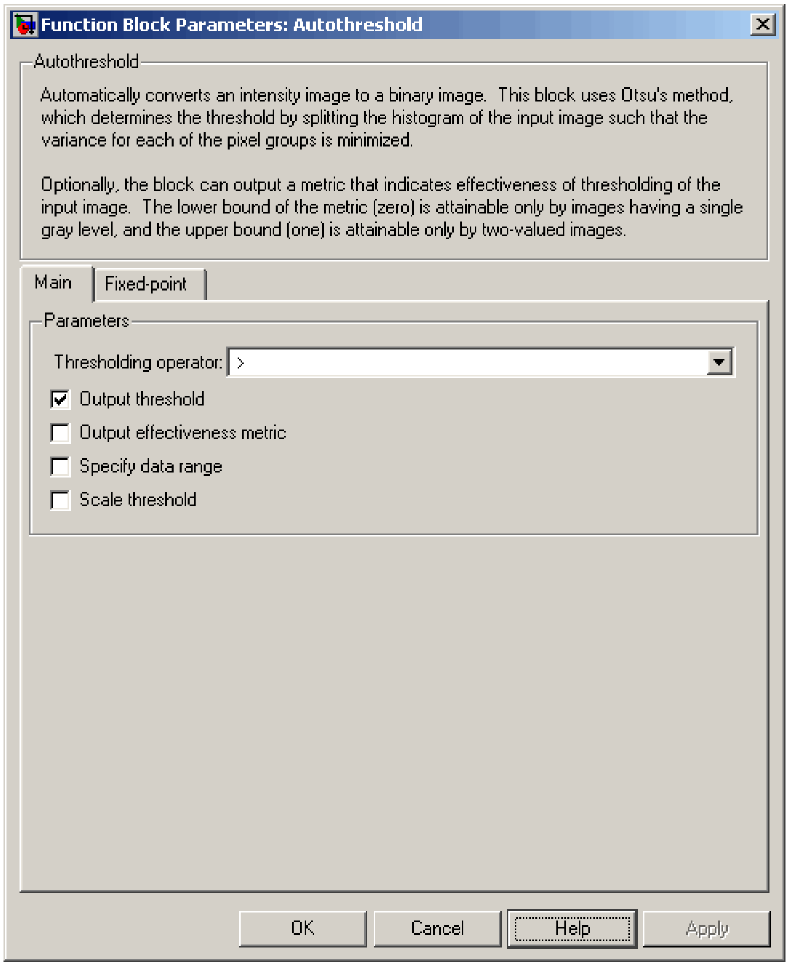
3 Delete the Constant and the Relational Operator blocks in this model.

4 Add an Autothreshold block from the Conversions library of the Computer Vision System Toolbox into your model.

5 Connect the blocks as shown in the following figure.



6 Use the Autothreshold block to perform a thresholding operation that converts your intensity image to a binary image. Select the **Output threshold** check box.


Function Block Parameters: Autothreshold

Autothreshold

Automatically converts an intensity image to a binary image. This block uses Otsu's method, which determines the threshold by splitting the histogram of the input image such that the variance for each of the pixel groups is minimized.

Optionally, the block can output a metric that indicates effectiveness of thresholding of the input image. The lower bound of the metric (zero) is attainable only by images having a single gray level, and the upper bound (one) is attainable only by two-valued images.

Main | Fixed-point

Parameters

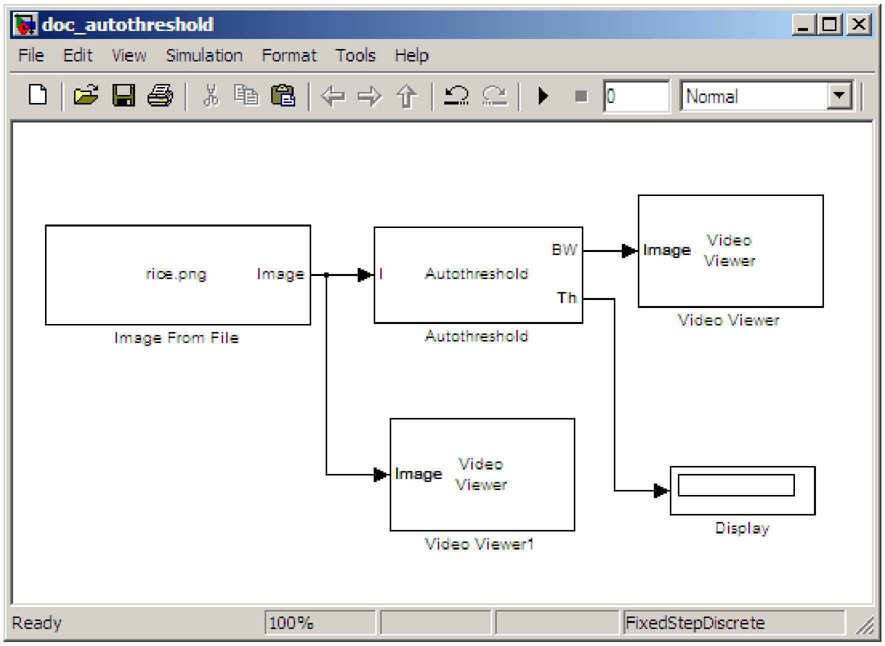
Thresholding operator: >

- [x] Output threshold
- [ ] Output effectiveness metric
- [ ] Specify data range
- [ ] Scale threshold

The block outputs the calculated threshold value at the **Th** port.

7 Add a Display block from the Sinks library of the DSP System Toolbox library. Connect the Display block to the **Th** output port of the Authothreshold block.

Your model should look similar to the following figure:



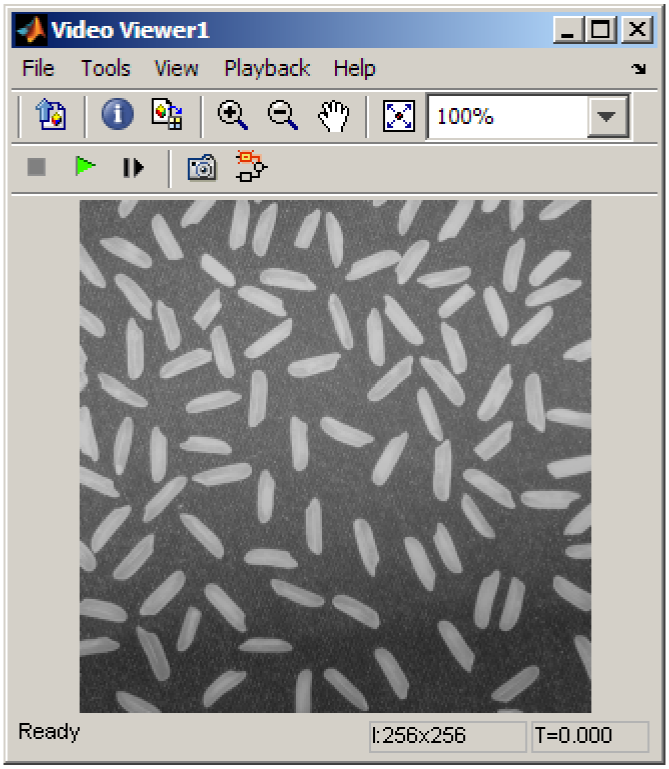
8 Double-click the Image From File block. On the **Data Types** pane, set the **Output data type** parameter to `double`.

9 If you have not already done so, set the configuration parameters. Open the Configuration dialog box by selecting **Model Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

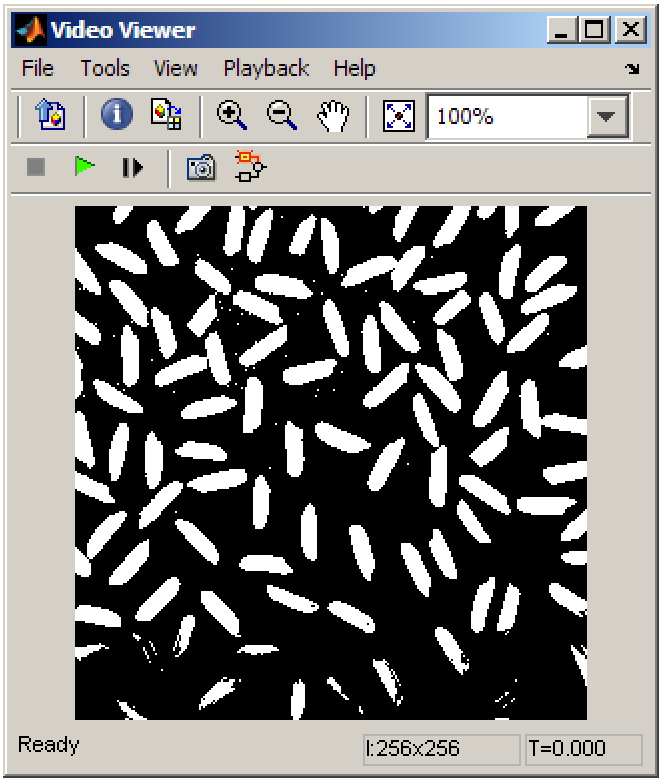
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

10 Run the model.

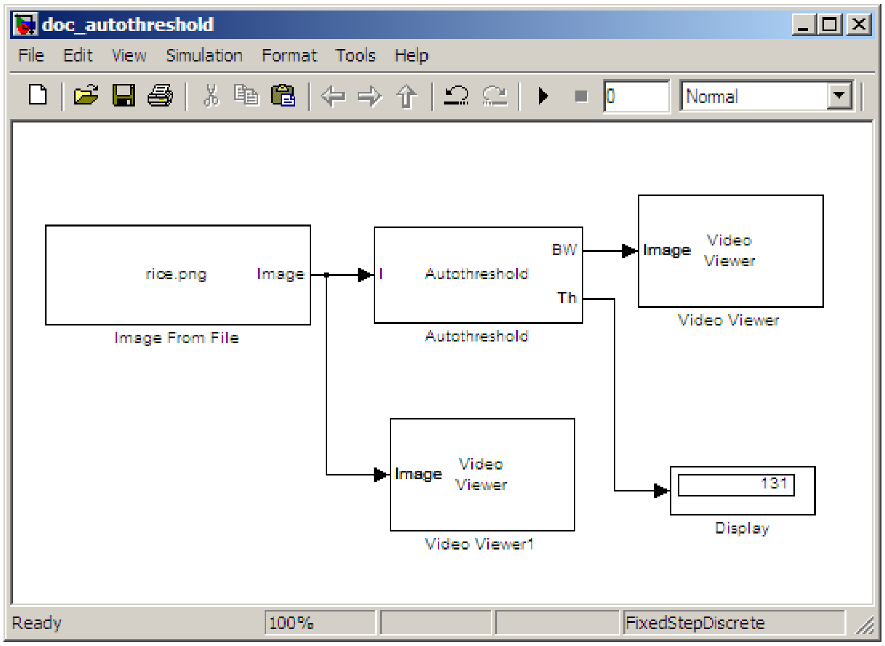
The original intensity image appears in the Video Viewer1 window.



The binary image appears in the Video Viewer window.



In the model window, the Display block shows the threshold value, calculated by the Autothreshold block, that separated the rice grains from the background.



You have used the Autothreshold block to convert an intensity image to a binary image. For more information about this block, see the Autothreshold block reference page in the *Computer Vision System Toolbox Reference*. To open a demo model that uses this block, type `vipstaples` at the MATLAB command prompt.

Convert R'G'B' to Intensity Images

The Color Space Conversion block enables you to convert color information from the R'G'B' color space to the Y'CbCr color space and from the Y'CbCr color space to the R'G'B' color space as specified by Recommendation ITU-R BT.601-5. This block can also be used to convert from the R'G'B' color space to intensity. The prime notation indicates that the signals are gamma corrected.

Some image processing algorithms are customized for intensity images. If you want to use one of these algorithms, you must first convert your image to intensity. In this topic, you learn how to use the Color Space Conversion block to accomplish this task. You can use this procedure to convert any R'G'B' image to an intensity image:

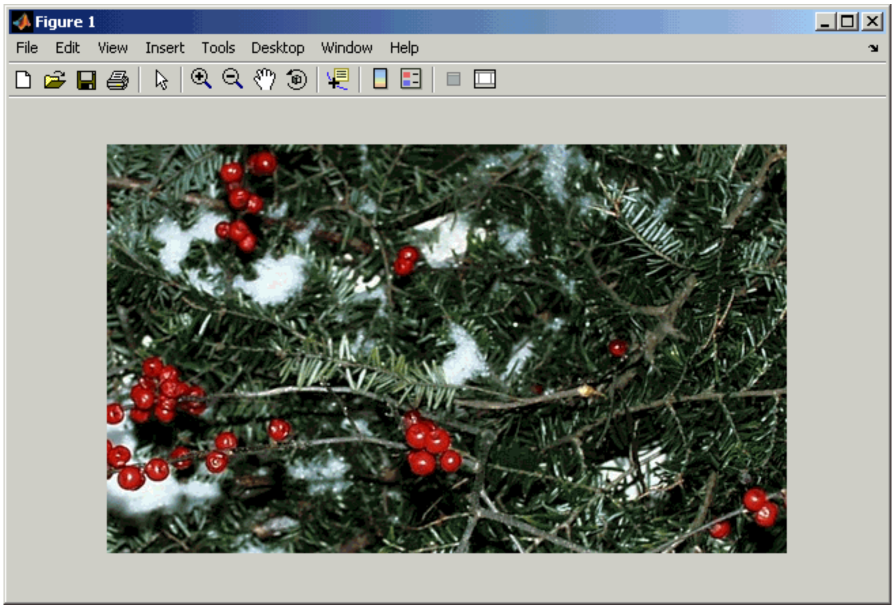
1 Define an R'G'B' image in the MATLAB workspace. To read in an R'G'B' image from a JPG file, at the MATLAB command prompt, type

```
I= imread('greens.jpg');
```

`I` is a 300-by-500-by-3 array of 8-bit unsigned integer values. Each plane of this array represents the red, green, or blue color values of the image.

2 To view the image this matrix represents, at the MATLAB command prompt, type

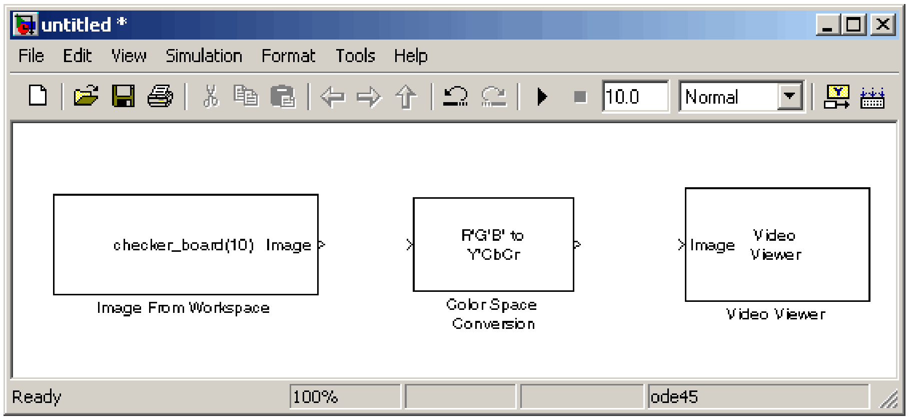
```
imshow(I)
```



3 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From Workspace	Computer Vision System Toolbox > Sources	1
Color Space Conversion	Computer Vision System Toolbox > Conversions	1
Video Viewer	Computer Vision System Toolbox > Sinks	1

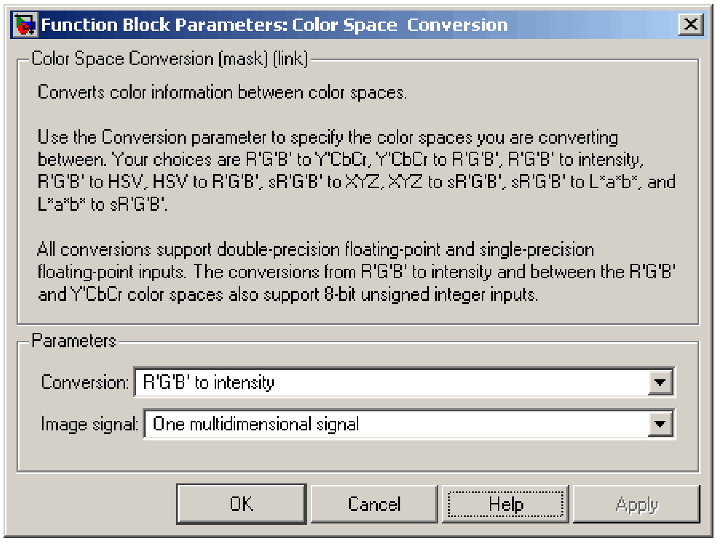
4 Position the blocks as shown in the following figure.



Once you have assembled the blocks needed to convert a R'G'B' image to an intensity image, you are ready to set your block parameters. To do this, double-click the blocks, modify the block parameter values, and click **OK**.

5 Use the Image from Workspace block to import your image from the MATLAB workspace. Set the**Value** parameter to `I`.

6 Use the Color Space Conversion block to convert the input values from the R'G'B' color space to intensity. Set the **Conversion** parameter to `R'G'B' to intensity`.



Color Space Conversion (mask) (link)

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'.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

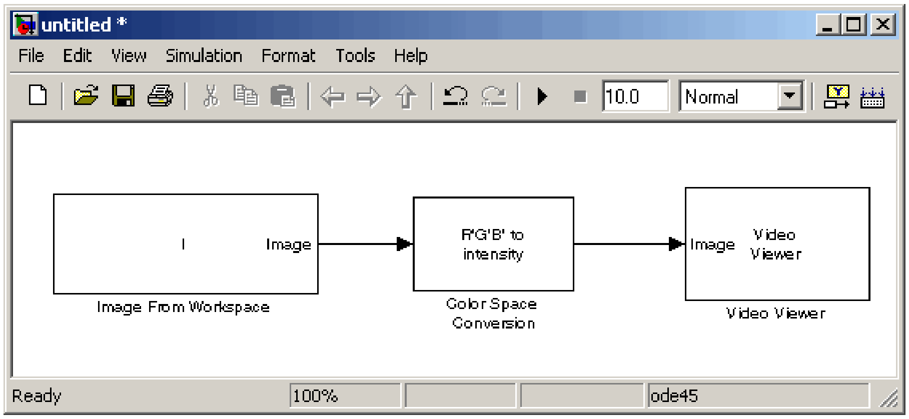
Parameters

Conversion: R'G'B' to intensity

Image signal: One multidimensional signal

7 View the modified image using the Video Viewer block. Accept the default parameters.

8 Connect the blocks so that your model is similar to the following figure.

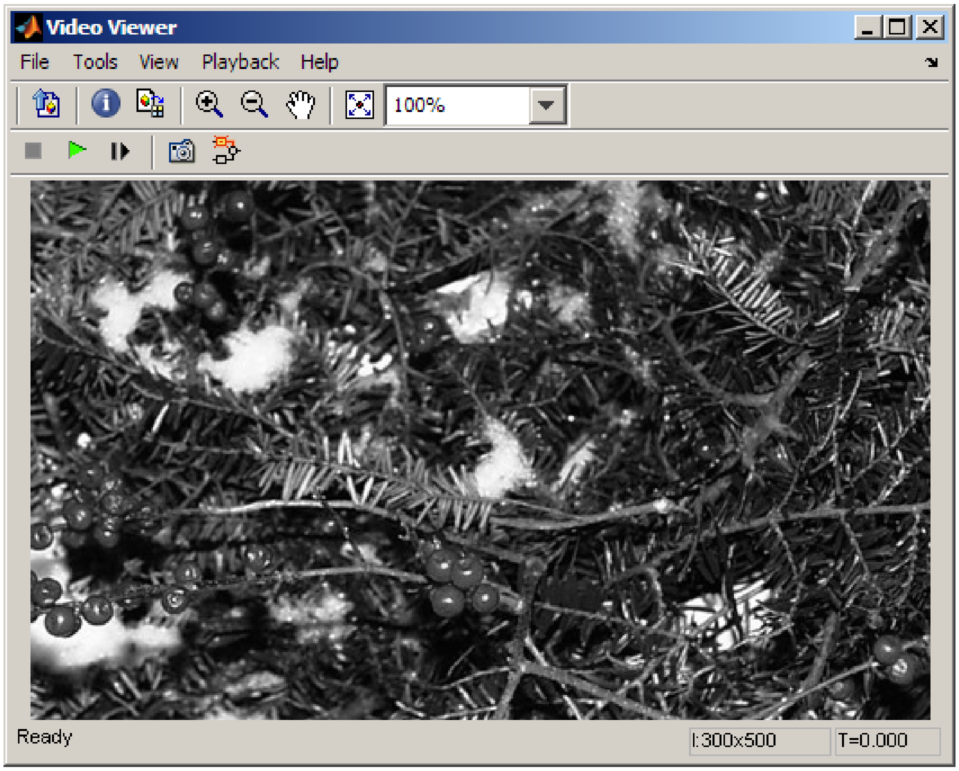


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Model Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

10 Run your model.

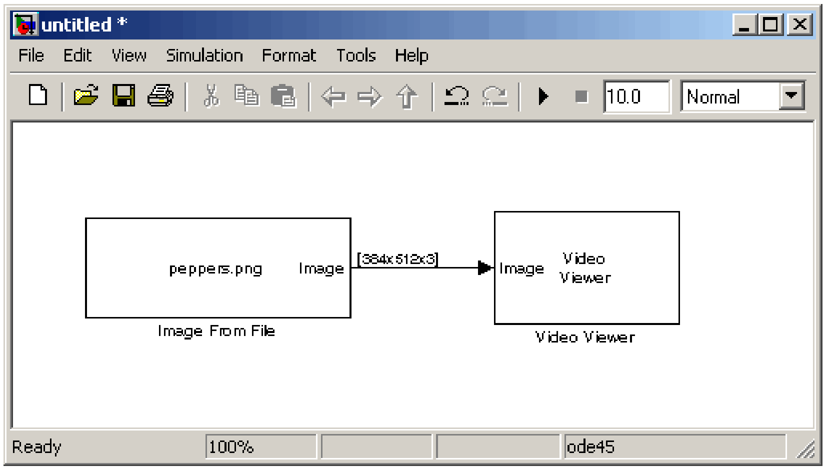
The image displayed in the Video Viewer window is the intensity version of the `greens.jpg` image.

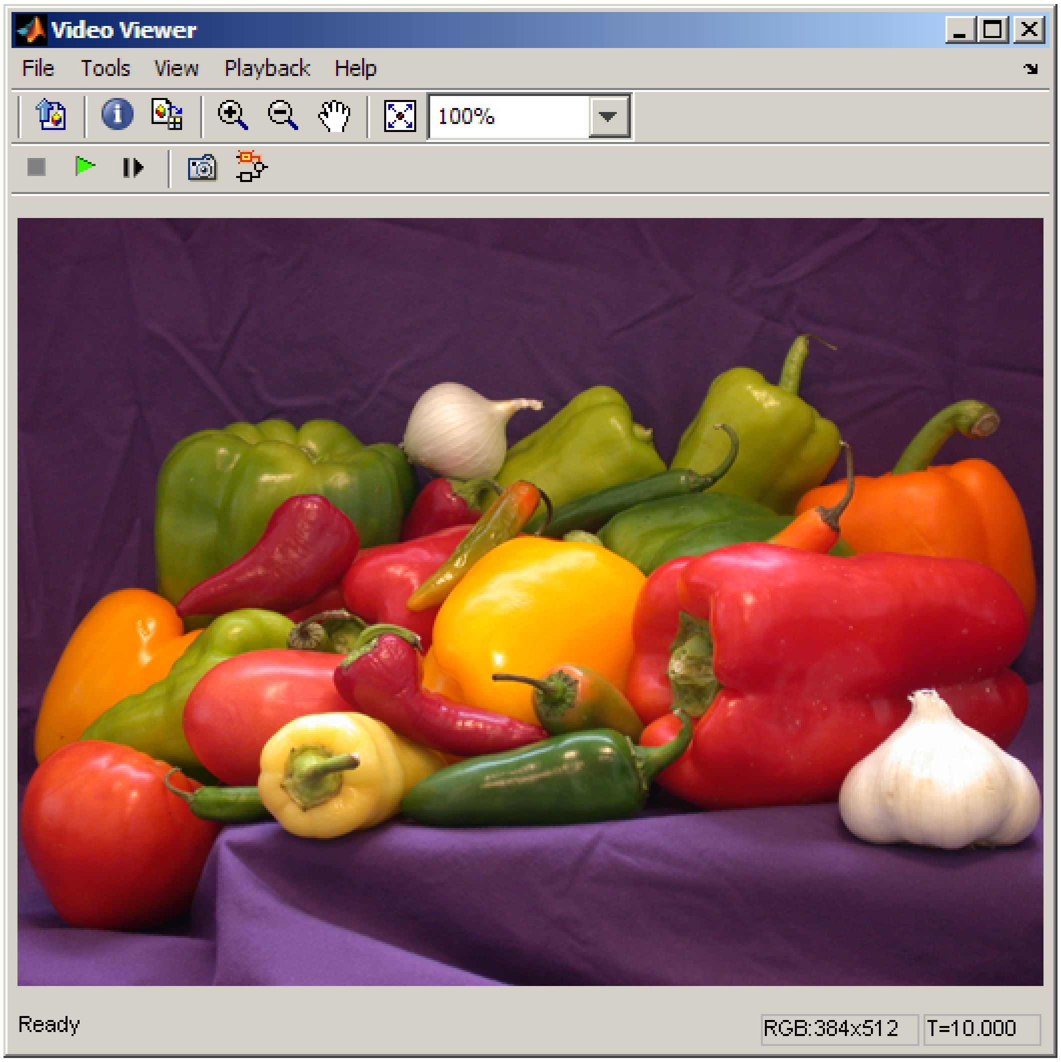


In this topic, you used the Color Space Conversion block to convert color information from the R'G'B' color space to intensity. For more information on this block, see the Color Space Conversion block reference page.

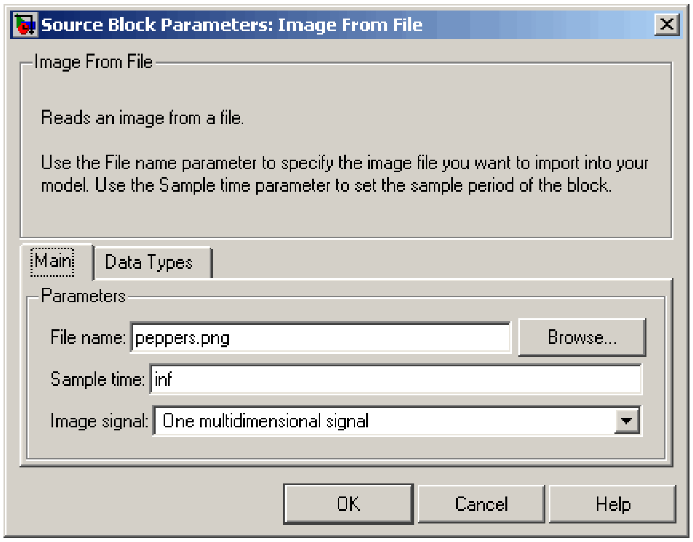
Process Multidimensional Color Video Signals

The Computer Vision System Toolbox software enables you to work with color images and video signals as multidimensional arrays. For example, the following model passes a color image from a source block to a sink block using a 384-by-512-by-3 array.





You can choose to process the image as a multidimensional array by setting the **Image signal** parameter to `One multidimensional signal` in the Image From File block dialog box.



Reads an image from a file.

Use the File name parameter to specify the image file you want to import into your model. Use the Sample time parameter to set the sample period of the block.

File name: peppers.png

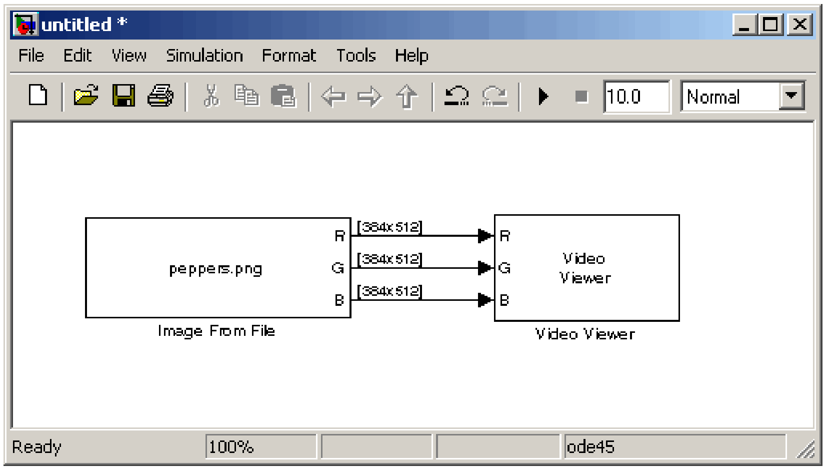
Sample time: inf

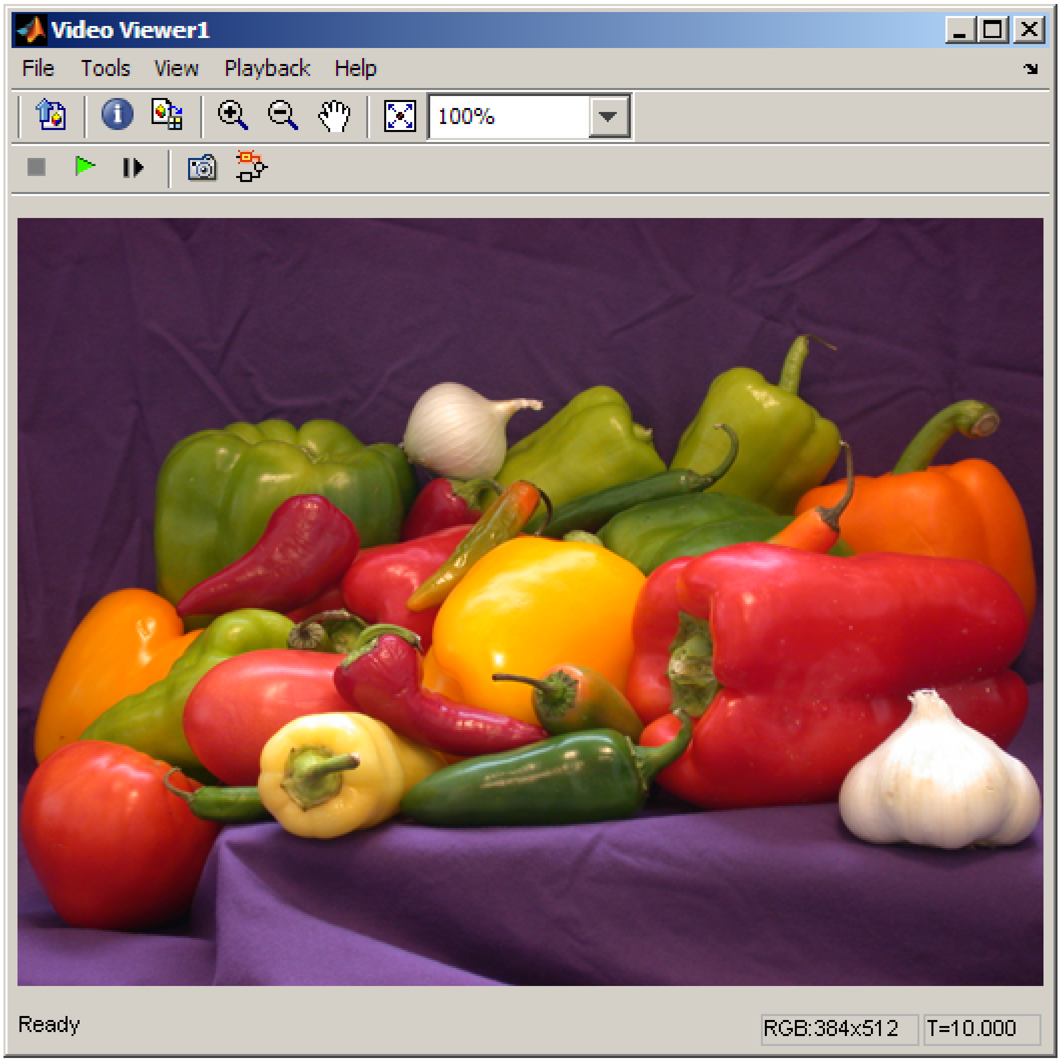
Image signal: One multidimensional signal

The blocks that support multidimensional arrays meet at least one of the following criteria:

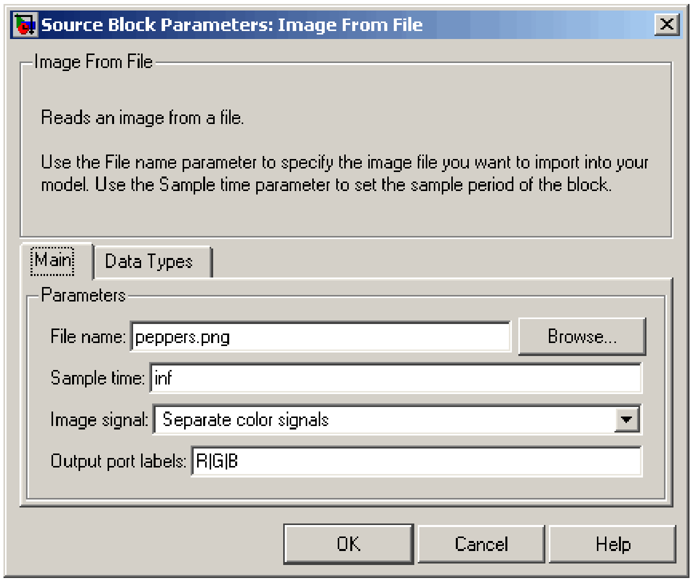
- They have the **Image signal** parameter on their block mask.
- They have a note in their block reference pages that says, “This block supports intensity and color images on its ports.”
- Their input and output ports are labeled “Image”.

You can also choose to work with the individual color planes of images or video signals. For example, the following model passes a color image from a source block to a sink block using three separate color planes.


untitled *
File Edit View Simulation Format Tools Help
10.0
Normal
peppers.png
R
G
B
Image From File
[384x512]
[384x512]
[384x512]
R
G
B
Video Viewer
Video Viewer
Ready
100%
ode45



To process the individual color planes of an image or video signal, set the **Image signal** parameter to `Separate color signals` in both the Image From File and Video Viewer block dialog boxes.



Source Block Parameters: Image From File

Image From File

Reads an image from a file.

Use the File name parameter to specify the image file you want to import into your model. Use the Sample time parameter to set the sample period of the block.

Main | Data Types

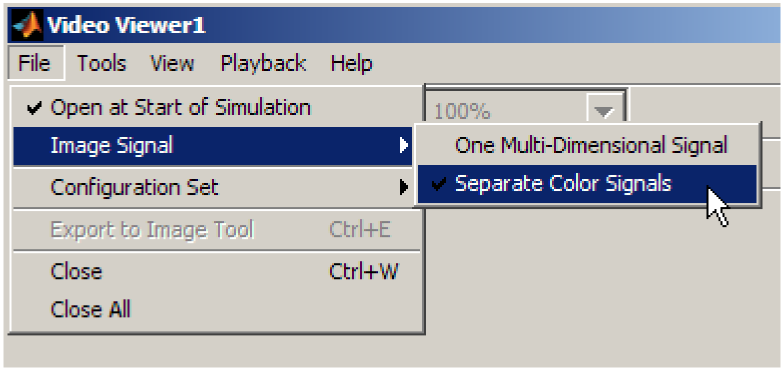
Parameters

File name: peppers.png

Sample time: inf

Image signal: Separate color signals

Output port labels: R|G|B



Note The ability to output separate color signals is a legacy option. It is recommend that you use multidimensional signals to represent color data.

If you are working with a block that only outputs multidimensional arrays, you can use the Selector block to separate the color planes. For an example of this process, see “Measure an Angle Between Lines” on page 3-16. If you are working with a block that only accepts multidimensional arrays, you can use the Matrix Concatenation block to create a multidimensional array. For an example of this process, see “Find the Histogram of an Image” on page 7-2.

Data Formats

In this section...

Video Formats

Video data is a series of images over time. Video in binary or intensity format is a series of single images. Video in RGB format is a series of matrices grouped into sets of three, where each matrix represents an R, G, or B plane.

Defining Intensity and Color

Video data is a series of images over time. Video in binary or intensity format is a series of single images. Video in RGB format is a series of matrices grouped into sets of three, where each matrix represents an R, G, or B plane.

The values in a binary, intensity, or RGB image can be different data types. The data type of the image values determines which values correspond to black and white as well as the absence or saturation of color. The following table summarizes the interpretation of the upper and lower bound of each data type. To view the data types of the signals at each port, from the **Format** menu, point to **Port/Signal Displays**, and select **Port Data Types**.

Data Type	Black or Absence of Color	White or Saturation of Color
Fixed point	Minimum data type value	Maximum data type value
Floating point	0	1

Note The Computer Vision System Toolbox software considers any data type other than double-precision floating point and single-precision floating point to be fixed point.

For example, for an intensity image whose image values are 8-bit unsigned integers, 0 is black and 255 is white. For an intensity image whose image values are double-precision floating point, 0 is black and 1 is white. For an intensity image whose image values are 16-bit signed integers, -32768 is black and 32767 is white.

For an RGB image whose image values are 8-bit unsigned integers, 0 0 0 is black, 255 255 255 is white, 255 0 0 is red, 0 255 0 is green, and 0 0 255 is blue. For an RGB image whose image values are double-precision floating point, 0 0 0 is black, 1 1 1 is white, 1 0 0 is red, 0 1 0 is green, and 0 0 1 is blue. For an RGB image whose image values are 16-bit signed integers, -32768 -32768 -32768 is black, 32767 32767 32767 is white, 32767 -32768 -32768 is red, -32768 32767 -32768 is green, and -32768 -32768 32767 is blue.

Video Data Stored in Column-Major Format

The MATLAB technical computing software and Computer Vision System Toolbox blocks use column-major data organization. The blocks' data buffers store data elements from the first column first, then data elements from the second column second, and so on through the last column.

If you have imported an image or a video stream into the MATLAB workspace using a function from the MATLAB environment or the Image Processing Toolbox, the Computer Vision System Toolbox blocks will display this image or video stream correctly. If you have written your own function or code to import images into the MATLAB environment, you must take the column-major convention into account.

Image Formats

In the Computer Vision System Toolbox software, images are real-valued ordered sets of color or intensity data. The blocks interpret input matrices as images, where each element of the matrix corresponds to a single pixel in the displayed image. Images can be binary, intensity (grayscale), or RGB. This section explains how to represent these types of images.

Binary Images

Binary images are represented by a Boolean matrix of 0s and 1s, which correspond to black and white pixels, respectively.

For more information, see “Binary Images” in the Image Processing Toolbox™ documentation.

Intensity Images

Intensity images are represented by a matrix of intensity values. While intensity images are not stored with colormaps, you can use a gray colormap to display them.

For more information, see “Grayscale Images” in the Image Processing Toolbox documentation.

RGB Images

RGB images are also known as a true-color images. With Computer Vision System Toolbox blocks, these images are represented by an array, where the first plane represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. In the Computer Vision System Toolbox software, you can pass RGB images between blocks as three separate color planes or as one multidimensional array.

For more information, see “Truecolor Images” in the Image Processing Toolbox documentation.

Display and Graphics

Display

In this section...

View Streaming Video in MATLAB using Video Player and Deployable Video Player System Objects

Video Player System Object

Use the video player System object when you require a simple video display in MATLAB.

For more information about the video player object, see the `vision.VideoPlayer` reference page.

Deployable Video Player System Object

Use the deployable video player object as a basic display viewer designed for optimal performance. This block supports code generation for the Windows® platform.

For more information about the Deployable Video Player block, see the `vision.DeployableVideoPlayer` object reference page.

Preview Video in MATLAB using MPlay Function

The MPlay function enables you to view videos represented as variables in the MATLAB workspace.

You can open several instances of the MPlay function simultaneously to view multiple video data sources at once. You can also dock these MPlay GUIs in the MATLAB desktop. Use the figure arrangement buttons in the upper-right corner of the Sinks window to control the placement of the docked GUIs.

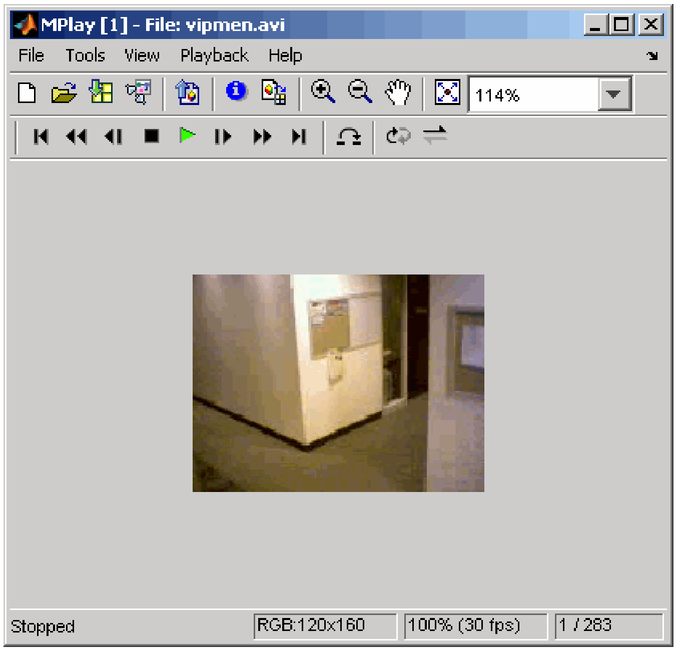
The MPlay GUI enables you to view videos directly from files without having to load all the video data into memory at once. The following procedure shows you how to use the MPlay GUI to load and view a video one frame at a time:

1 On the MPlay GUI, click *open file* element,

2 Use the Connect to File dialog box to navigate to the multimedia file you want to view in the MPlay window.

   For example, navigate to `$matlabroot\toolbox\vision\visiondemos\vipmen.avi`.

   Click **Open**. The first frame of the video appears in the MPlay window.



Stopped RGB:120x160 100% (30 fps) 1 / 283

Note The MPlay GUI supports AVI files that the mmreader supports.

3 Experiment with the MPlay GUI by using it to play and interact with the video stream.

View Video in Simulink using the Video Viewer and To Video Display Blocks

Video Viewer Block

Use the Video Viewer block when you require a wired-in video display with simulation controls in your Simulink model. The Video Viewer block provides simulation control buttons directly from the GUI. The block integrates play, pause, and step features while running the model and also provides video analysis tools such as pixel region viewer.

For more information about the Video Viewer block, see the Video Viewer block reference page.

To Video Display Block

Use the To Video Display block in your Simulink model as a simple display viewer designed for optimal performance. This block supports code generation for the Windows platform.

For more information about the To Video Display block, see the To Video Display block reference page.

View Video in Simulink using MPlay Function as a Floating Scope

The MPlay GUI enables you to view video signals in Simulink models without adding blocks to your model.

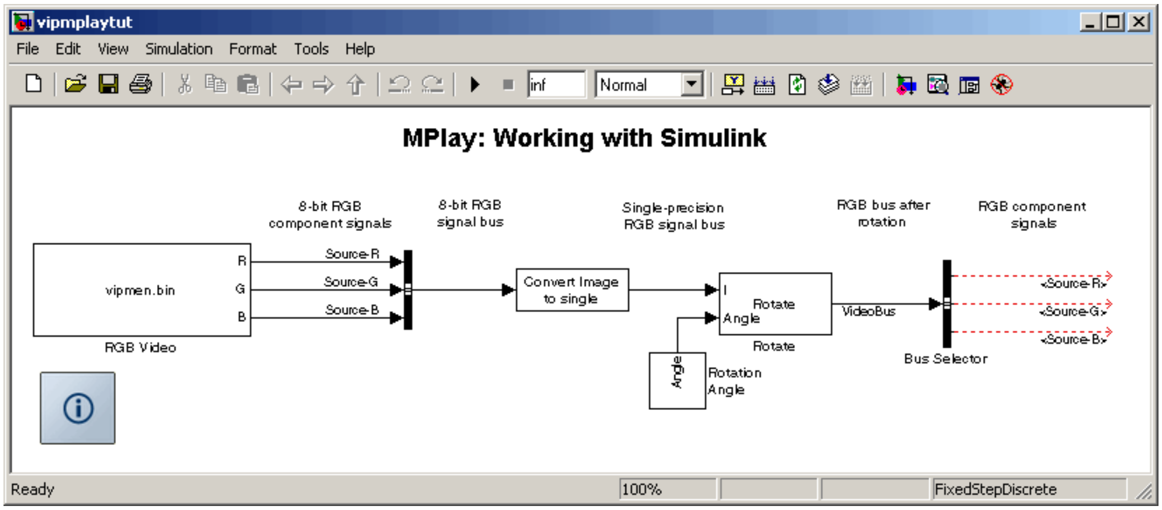
You can open several instances of the MPlay GUI simultaneously to view multiple video data sources at once. You can also dock these MPlay GUIs in the MATLAB desktop. Use the figure arrangement buttons in the upper-right corner of the Sinks window to control the placement of the docked GUIs.

Set Simulink simulation mode to `Normal` to use `mplay` . MPlay does not work when you use "Accelerating Simulink Models" on page 8-8.

The following procedure shows you how to use MPlay to view a Simulink signal:

1. Open a Simulink model. At the MATLAB command prompt, type

   ```
   vipmplaytut
   ```



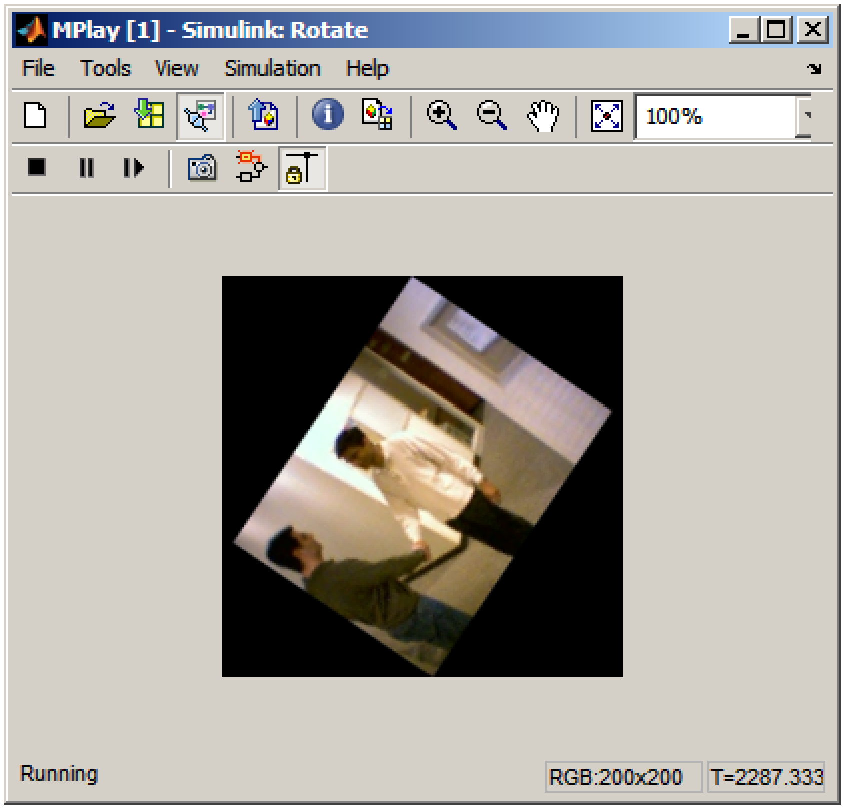
2. Open an MPlay GUI by typing `mplay` on the MATLAB command line.

3. Run the model.

4. Select the signal line you want to view. For example, select the bus signal coming out of the Rotate block.

5. On the MPlay GUI, click *Connect to Simulink Signal* GUI element,

   The video appears in the MPlay window.



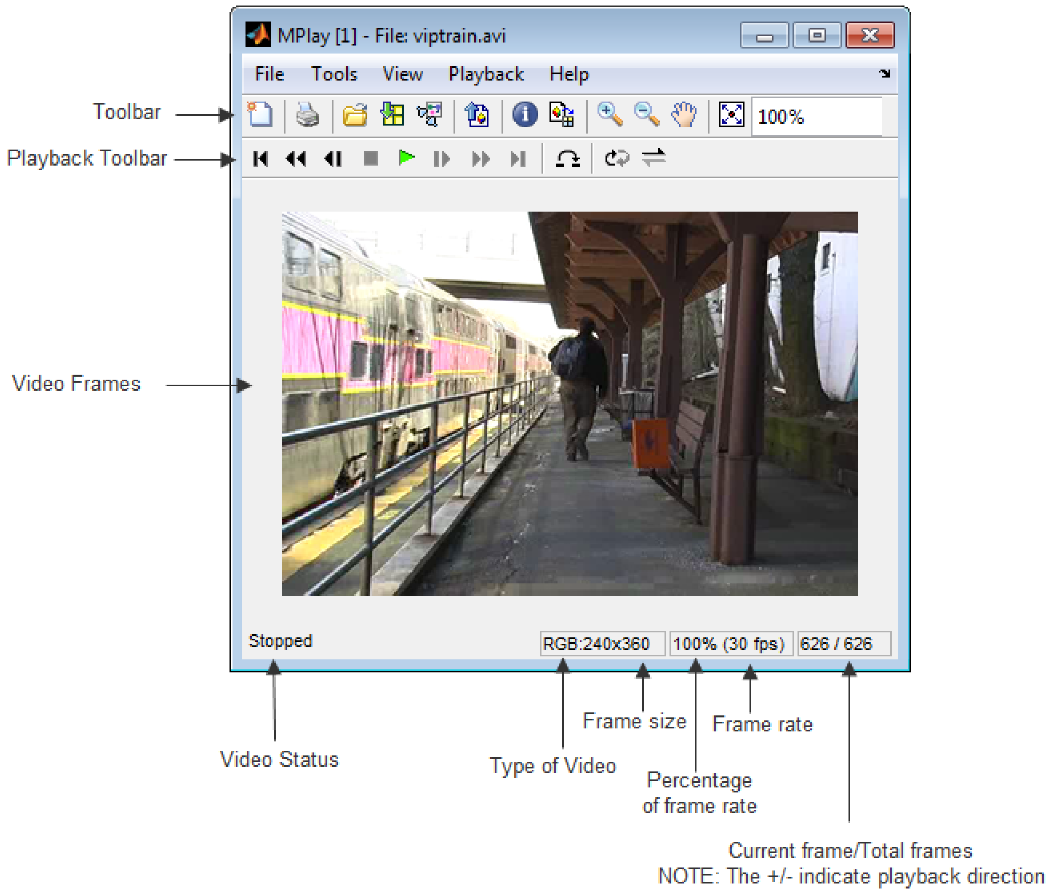
6 Change to floating-scope mode by clicking the *persistent connect* GUI element, button.

7 Experiment with selecting different signals and viewing them in the MPlay window. You can also use multiple MPlay GUIs to display different Simulink signals.

Note During code generation, the Simulink® Coder™ does not generate code for the MPlay GUI.

MPlay

The following figure shows the MPlay GUI containing an image sequence.



The following sections provide descriptions of the MPlay GUI toolbar buttons and equivalent menu options.

Toolbar Buttons

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	File > New MPlay	**Ctrl+N**	Open a new MPlay GUI.
	File > Print	**Ctrl+P**	Print the current scope window. Printing is only available when the scope display is not changing. You can enable printing by placing the scope in snapshot mode, or by pausing or stopping model simulation. To print the current scope window to a figure rather than sending it to your printer, select **File > Print to figure**.
	File > Open	**Ctrl+O**	Connect to a video file.
	File > Import from Workspace	**Ctrl+I**	Connect to a variable from the base MATLAB workspace.
	File > Connect to Simulink Signal		Connect to a Simulink signal.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	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. The Image Tool only knows the frame is an intensity image if the colormap of the frame is grayscale (`gray(256)`). Otherwise, the Image Tool assumes that the frame is an indexed image and disables the **Adjust Contrast** button.
	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.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
⛶	**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 display. This option is available if you do not select the **Maintain Fit to Window** button.

Playback Toolbar — Workspace and File Sources

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
⏮	**Playback > Go to First**	**F**, **Home**	Go to the first frame of the video.
⏪	**Playback > Rewind**	Up arrow	Jump back ten frames.
◀❚	**Playback > Step Back**	Left arrow, **Page Up**	Step back one frame.
■	**Playback > Stop**	**S**	Stop the video.
▶	**Playback > Play**	**P**, Space bar	Play the video.
❚❚	**Playback > Pause**	**P**, Space bar	Pause the video. This button appears only when the video is playing.
❚▶	**Playback > Step Forward**	Right arrow, **Page Down**	Step forward one frame.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
⏩	**Playback > Fast Forward**	Down arrow	Jump forward ten frames.
⏭	**Playback > Go to Last**	**L**, **End**	Go to the last frame of the video.
↷	**Playback > Jump to**	**J**	Jump to a specific frame.
🔁	**Playback > Playback Modes > Repeat**	**R**	Repeated video playback.
⇄	**Playback > Playback Modes > Forward play**	**A**	Play the video forward.
⇄	**Playback > Playback Modes > Backwardplay**	**A**	Play the video backward.
⇄	**Playback > Playback Modes > AutoReverse play**	**A**	Play the video forward and backward.

Playback Toolbar — Simulink Sources

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
■	**Simulation > Stop**	S	Stop the video. This button also controls the Simulink model.
▶	**Simulation > Start**	**P**, Space bar	Play the video. This button also controls the Simulink model.
⏸	**Simulation > Pause**	**P**, Space bar	Pause the video. This button also controls the Simulink model and appears only when the video is playing.
❚▶	**Simulation > Step Forward**	Right arrow, Page Down	Step forward one frame. This button also controls the Simulink model.
📷	**Simulation > Simulink Snapshot**	N/A	Click this button to freeze the display in the MPlay window.
⧉	**View > Highlight Simulink Signal**	**Ctrl+L**	In the model window, highlight the Simulink signal the MPlay GUI is displaying.
🔒	**Simulation > Floating Signal Connection (not selected)**	N/A	Indicates persistent Simulink connection. In this mode, the MPlay GUI always associates with the Simulink signal you selected before you clicked the **Connect to Simulink Signal** button.
🔓	**Simulation > Floating Signal**	N/A	Indicates floating Simulink connection. In this mode, you can click different signals in

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	Connection (selected)		the model, and the MPlay GUI displays them. You can use only one MPlay GUI in floating-scope mode at a time.

Configuration

The MPlay Configuration dialog box enables you to change the behavior and appearance of the GUI as well as the behavior of the playback shortcut keys.

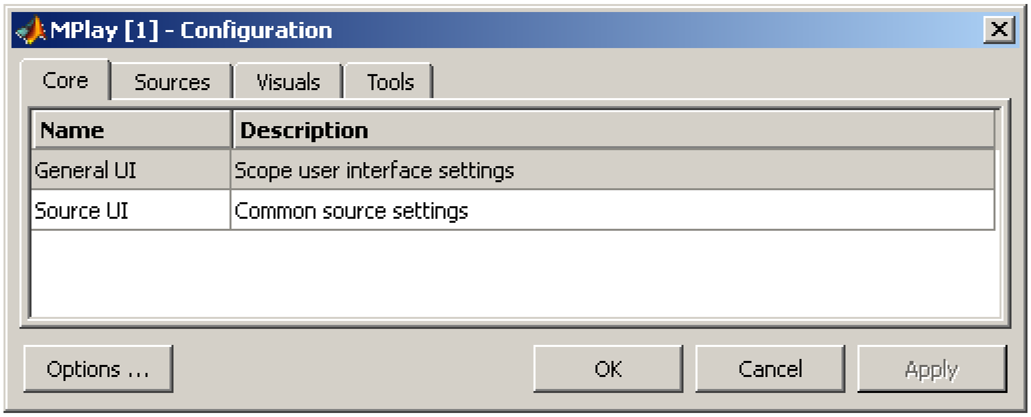
- To open the Configuration dialog box, select **File > Configuration Set > Edit**.
- To save the configuration settings for future use, select **File > Configuration Set > Save as**.

Note By default, the MPlay GUI uses the configuration settings from the file `mplay.cfg`. Create a backup copy of the file to store your configuration settings.

- To load a preexisting configuration set, select **File > Configuration Set > Load**.

Configuration Core Pane

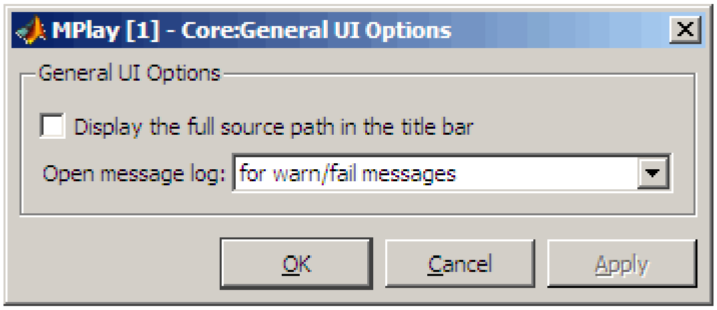
The Core pane controls the graphic user interface (GUI) general and source settings.


MPlay [1] - Configuration

Name	Description
General UI	Scope user interface settings
Source UI	Common source settings

General UI

Click **General UI**, and then select the **Options** button to open the General UI Options dialog box.


MPlay [1] - Core:General UI Options

General UI Options

☐ Display the full source path in the title bar

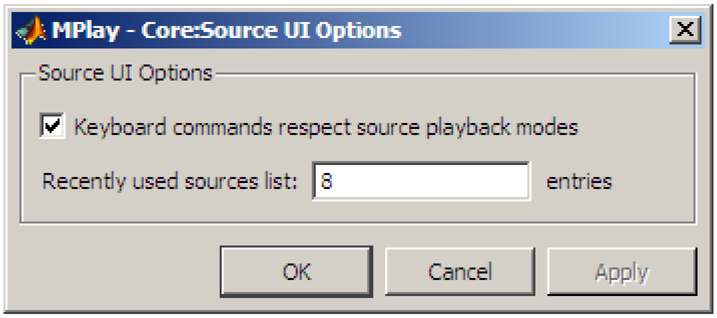
Open message log: for warn/fail messages

If you select the **Display the full source path in the title bar** check box, the full Simulink path appears in the title bar. Otherwise, the title bar displays a shortened name.

Use the **Message log opens** 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`.

Source UI

Click Source UI, and then click the **Options** button to open the Source UI Options dialog box.



☑ Keyboard commands respect source playback modes

Recently used sources list: 8 entries

If you select the **Keyboard commands respect playback modes** check box, the keyboard shortcut keys behave in response to the playback mode you selected.

Using the Keyboard commands respect playback modes

Open and play a video using MPlay.

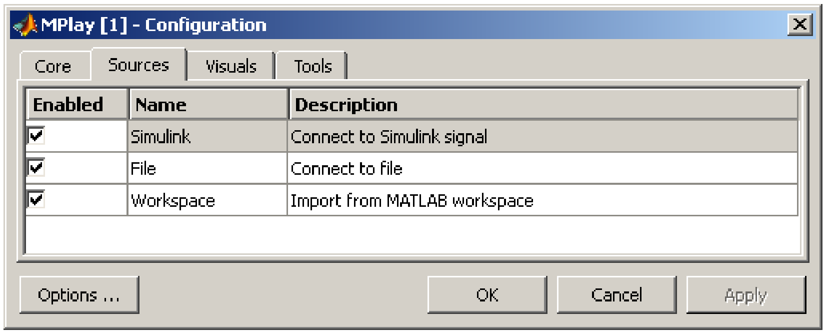
1. Select the **Keyboard commands respect playback modes** check box.
2. Select the **Backward playback** button.
   - Using the right keyboard arrow key moves the video backward, and using the left keyboard arrow key moves the video forward.
   - With MPlay set to play backwards, the keyboard "forward" performs "forward with the direction the video is playing".

To disconnect the keyboard behavior from the MPlay playback settings, clear the check box.

Use the **Recently used sources list** parameter to control the number of sources you see in the **File** menu.

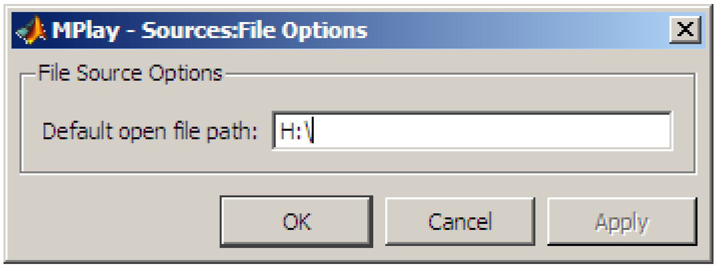
Configuration Sources Pane

The Sources pane contains the GUI options that relate to connecting to different sources. Select the **Enabled** check box next to each source type to specify to which type of source you want to connect the GUI.



Enabled	Name	Description
✓	Simulink	Connect to Simulink signal
✓	File	Connect to file
✓	Workspace	Import from MATLAB workspace

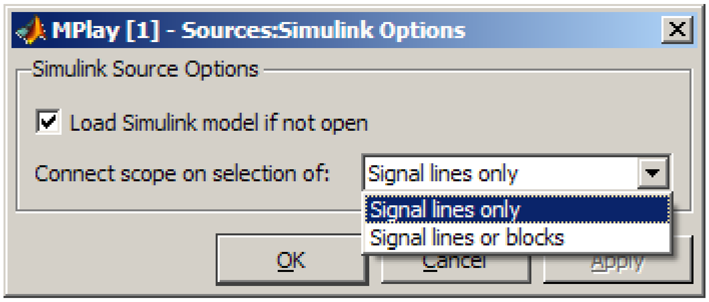
- Click **File**, and then click the **Options** button to open the **Sources:File Options** dialog box.



Default open file path: H:\

Use the **Default open file path** parameter to control the folder that is displayed in the **Connect to File** dialog box. The **Connect to File** dialog box becomes available when you select **File > Open**.

- Click **Simulink**, and then click the **Options** button to open the **Sources:Simulink Options** dialog box.



Load Simulink model if not open [✓]

Connect scope on selection of: Signal lines only

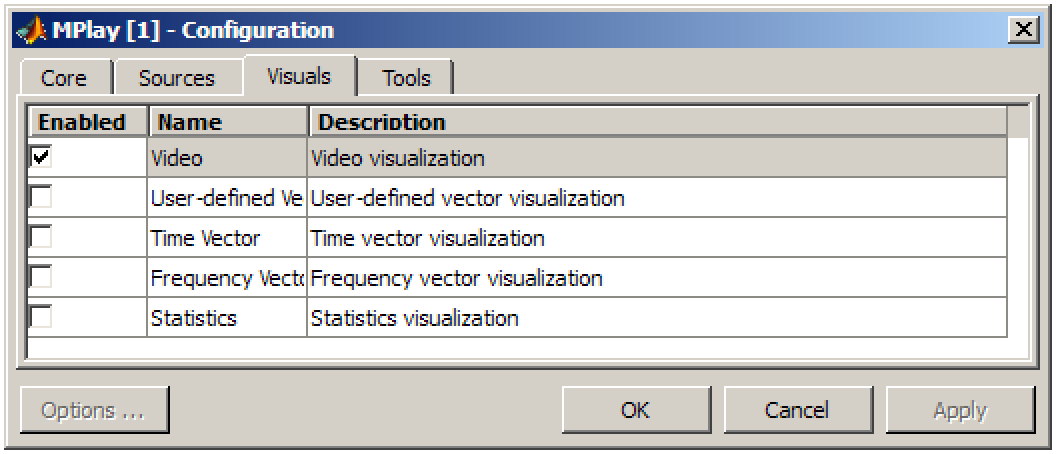
- Signal lines only
- Signal lines or blocks

You can have the Simulink model associated with an MPlay GUI to open with MPlay. To do so, select the **Load Simulink model if not open** check box.

Select **Signal lines only** to sync the video display only when you select a signal line. If you select a block the video display will not be affected. Select **Signal lines or blocks** to sync the video display to the signal line or block you select. The default is **Signal lines only**.

Configuration Visuals Pane

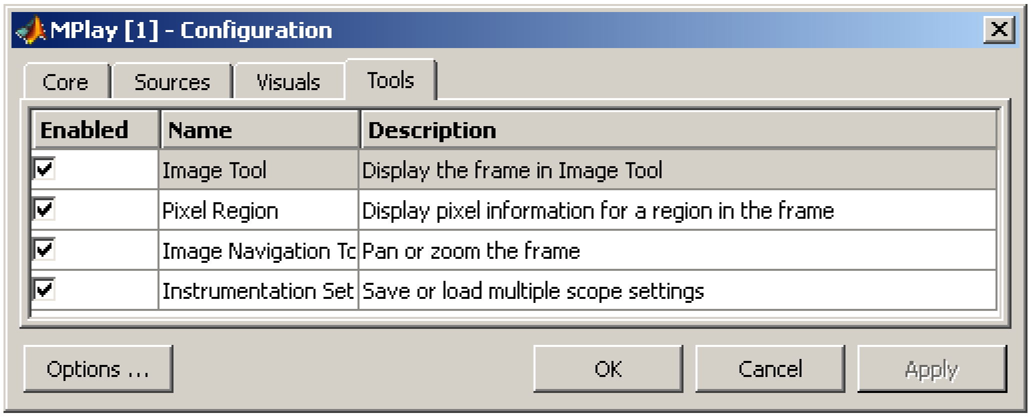
The Visuals pane contains the name of the visual type and its description.



Enabled	Name	Description
✓	Video	Video visualization
	User-defined Ve	User-defined vector visualization
	Time Vector	Time vector visualization
	Frequency Vecto	Frequency vector visualization
	Statistics	Statistics visualization

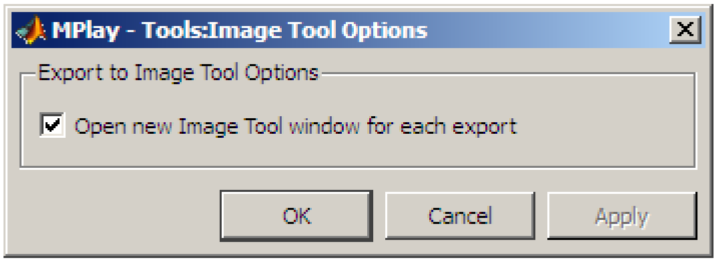
Configuration Tools Pane

The Tools pane contains the tools that are available on the MPlay GUI. Select the **Enabled** check box next to the tool name to specify which tools to include on the GUI.



Enabled	Name	Description
✓	Image Tool	Display the frame in Image Tool
✓	Pixel Region	Display pixel information for a region in the frame
✓	Image Navigation Tc	Pan or zoom the frame
✓	Instrumentation Set	Save or load multiple scope settings

Click **Image Tool**, and then click the **Options** button to open the Image Tool Options dialog box.


MPlay - Tools:Image Tool Options
Export to Image Tool Options
OK
Cancel
Apply

✓ Open new Image Tool window for each export

Select the **Open new Image Tool window for export** check box if you want to open a new Image Tool for each exported frame.

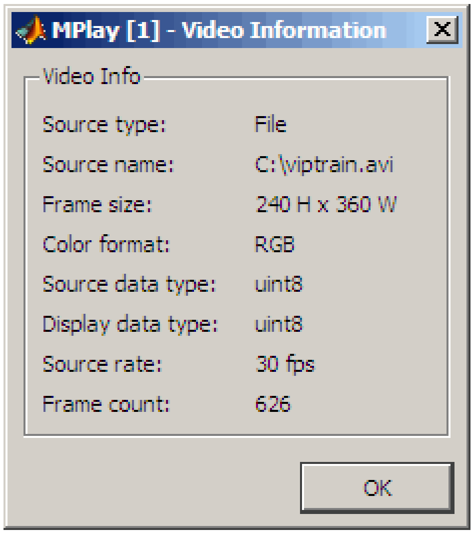
Pixel Region
Select the **Pixel Region** check box to display and enable the pixel region GUI button. For more information on working with pixel regions, see Getting Information about the Pixels in an Image.

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, select **Tools > Video Information** or click the information button ⓘ .

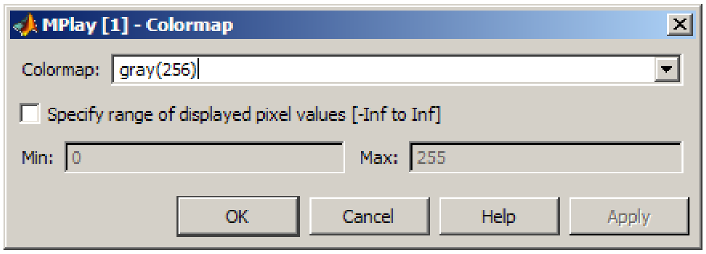


Video Info

Source type:	File
Source name:	C:\viptrain.avi
Frame size:	240 H x 360 W
Color format:	RGB
Source data type:	uint8
Display data type:	uint8
Source rate:	30 fps
Frame count:	626

Color Map 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**.


MPlay [1] - Colormap

Colormap: gray(256)

Specify range of displayed pixel values [-Inf to Inf]

Min: 0 Max: 255

Use the **Colormap** parameter to specify the colormap to apply to the intensity video.

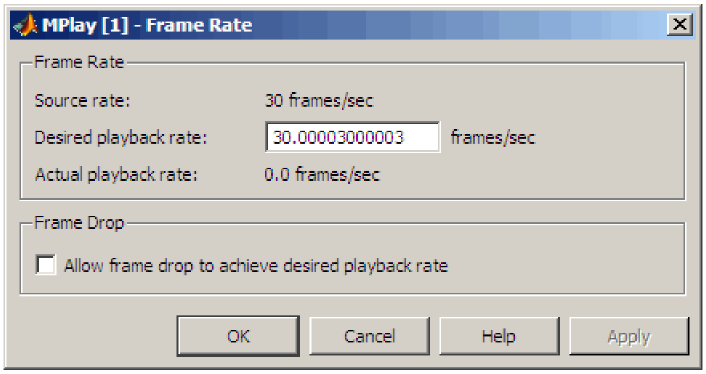
Sometimes, the pixel values do not use the entire data type range. In such cases, you can select the **Specify range of displayed pixel values** check box. You can then enter the range for your data. The dialog box automatically displays the range based on the data type of the pixel values.

Frame Rate

The Frame Rate dialog box displays the frame rate of the source. It also lets you change the rate at which the MPlay GUI plays the video and displays the actual playback rate.

Note This dialog box becomes available when you use the MPlay GUI to view a video signal.

The *playback rate* is the number of frames the GUI processes per second. You can use the **Desired playback rate** parameter to decrease or increase the playback rate. To open this dialog box, select **Playback > Frame Rate** or press **T**.



Frame Rate

Source rate: 30 frames/sec

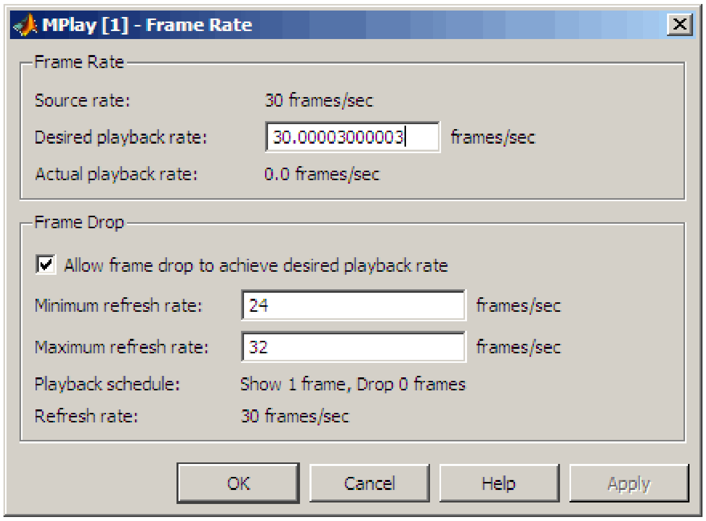
Desired playback rate: 30.00003000003 frames/sec

Actual playback rate: 0.0 frames/sec

Frame Drop

☐ Allow frame drop to achieve desired playback rate

To increase the playback rate when system hardware cannot keep pace with the desired rate, select the **Allow frame drop to achieve desired playback rate** check box. This parameter enables the MPlay GUI to achieve the playback rate by dropping video frames. Dropped video frames sometimes cause lower quality playback.



Frame Rate

Source rate: 30 frames/sec

Desired playback rate: 30.00003000003 frames/sec

Actual playback rate: 0.0 frames/sec

Frame Drop

☑ Allow frame drop to achieve desired playback rate

Minimum refresh rate: 24 frames/sec

Maximum refresh rate: 32 frames/sec

Playback schedule: Show 1 frame, Drop 0 frames

Refresh rate: 30 frames/sec

You can refine further the quality of playback versus hardware burden, by controlling the number of frames to drop per frame or frames displayed. For example, suppose you set the **Desired playback rate** to 80 frames/sec. One way to achieve the desired playback rate is to set the **Playback schedule** to `Show 1 frame, Drop 3 frames`. Change this playback schedule, by setting the refresh rates (which is how often the GUI updates the screen), to:

Maximum refresh rate: `21` frames/sec
Minimum refresh rate: `20` frames/sec

MPlay can achieve the desired playback rate (in this case, 80 frames/sec) by using these parameter settings.

In general, the relationship between the **Frame Drop** parameters is:

$$Desired\_rate = refresh\_rate * \frac{show\_frames + drop\_frames}{show\_frames}$$

In this case, the *refresh_rate* includes a more accurate calculation based on both the minimum and maximum refresh rates.

Use the **Minimum refresh rate** and **Maximum refresh rate** parameters to adjust the playback schedule of video display. Use these parameters in the following way:

- Increase the **Minimum refresh rate** parameter to achieve smoother playback.
- Decrease the **Maximum refresh rate** parameter to reduce the demand on system hardware.

Saving the Settings of Multiple MPlay GUIs

The MPlay GUI enables you to save and load the settings of multiple GUI instances. You only have to configure the MPlay GUIs associated with your model once.

To save the GUI settings:

- Select **File > Instrumentation Sets > Save Set**

To open the preconfigured MPlay GUIs:

- Select **File > Instrumentation Sets > Load Set**

You can save instrument sets for instances of MPlay connected to a source. If you attempt to save an instrument set for an MPlay instance that is not connected to a source, the Message Log displays a warning.

Message Log

The Message Log dialog box 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` message indicates a new configuration file loaded.
- The `Extension` message indicates a registered component. For example, a `Simulink` message, indicating a registered component, available for configuration.

Status Bar

Along the bottom of the MPlay viewer is the status bar. It displays information, such as video status, Type of video playing (`I` or `RGB`), Frame size, Percentage of frame rate, Frame rate, and `Current frame:  Total frames`.

Note A minus sign `(-)` for `Current frame` indicates reverse video playback.

Graphics

In this section...

Abandoned Object Detection

This example tracks objects at a train station and determines which ones remain stationary. Abandoned objects in public areas concern authorities since they might pose a security risk. Algorithms, such as the one used in this example, can be used to assist security officers monitoring live surveillance video by directing their attention to a potential area of interest.

This example illustrates how to use the BlobAnalysis System object to identify objects and track them. The example implements this algorithm using the following steps:

- Extract a region of interest (ROI), thus eliminating video areas that are unlikely to contain abandoned objects.
- Perform video segmentation using background subtraction.
- Calculate object statistics using the blob analysis System object.
- Track objects based on their area and centroid statistics.
- Visualize the results.

Initialize Required Variables and System Objects

Use these next sections of code to initialize the required variables and System objects.

Rectangular ROI [x y width height], where [x y] is the uppef left corner of the ROI

```
roi = [100 80 360 240];
% Maximum number of objects to track
maxNumObj = 200;
```

```
% Number of frames that an object must remain stationary before an alarm
% raised
alarmCount = 45;
% Maximum number of frames that an abandoned object can be hidden before
% is no longer tracked
maxConsecutiveMiss = 4;
areaChangeFraction = 13;     % Maximum allowable change in object area ir
centroidChangeFraction = 18; % Maximum allowable change in object centroi
% Minimum ratio between the number of frames in which an object is detect
% and the total number of frames, for that object to be tracked.
minPersistenceRatio = 0.7;
% Offsets for drawing bounding boxes in original input video
PtsOffset = int32(repmat([roi(1), roi(2), 0, 0],[maxNumObj 1]));
```

Create a VideoFileReader System object to read video from a file.

```
hVideoSrc = vision.VideoFileReader;
hVideoSrc.Filename = 'viptrain.avi';
hVideoSrc.VideoOutputDataType = 'single';
```

Create a ColorSpaceConverter System object to convert the RGB image to Y'CbCr format.

```
hColorConv = vision.ColorSpaceConverter('Conversion', 'RGB to YCbCr');
```

Create an Autothresholder System object to convert an intensity image to a binary image.

```
hAutothreshold = vision.Autothresholder('ThresholdScaleFactor', 1.3);
```

Create a MorphologicalClose System object to fill in small gaps in the detected objects.

```
hClosing = vision.MorphologicalClose('Neighborhood', strel('square',5));
```

Create a BlobAnalysis System object to find the area, centroid, and bounding box of the objects in the video.

```
hBlob = vision.BlobAnalysis('MaximumCount', maxNumObj, 'ExcludeBorderBlob
hBlob.MinimumBlobAreaSource = 'Property';
hBlob.MinimumBlobArea = 100;
hBlob.MaximumBlobAreaSource = 'Property';
hBlob.MaximumBlobArea = 2500;
```

Create a ShapeInserter System object to draw rectangles around the abandoned objects.

```
hDrawRectangles1 = vision.ShapeInserter('Fill',true, 'FillColor', 'Custor
    'CustomFillColor', [1 0 0], 'Opacity', 0.5);
```

Create a TextInserter System object to display the number of objects in the video.

```
hDisplayCount = vision.TextInserter('Text', '%4d', 'Color', [1 1 1]);
```

Create a ShapeInserter System object to draw rectangles around all the detected objects in the video.

```
hDrawRectangles2 = vision.ShapeInserter('BorderColor', 'Custom', ...
    'CustomBorderColor', [0 1 0]);
```

Create a ShapeInserter System object to draw a rectangle around the region of interest.

```
hDrawBBox = vision.ShapeInserter('BorderColor', 'Custom', ...
    'CustomBorderColor', [1 1 0]);
```

Create a ShapeInserter System object to draw rectangles around all the identified objects in the segmented video.

```
hDrawRectangles3 = vision.ShapeInserter('BorderColor', 'Custom', ...
    'CustomBorderColor', [0 1 0]);
```

Create System objects to display results.

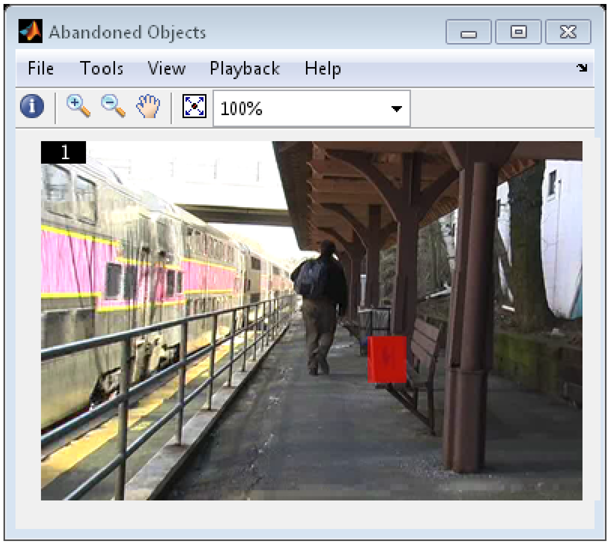
```
pos = [10 300 roi(3)+25 roi(4)+25];
hAbandonedObjects = vision.VideoPlayer('Name', 'Abandoned Objects', 'Pos
pos(1) = 46+roi(3); % move the next viewer to the right
hAllObjects = vision.VideoPlayer('Name', 'All Objects', 'Position', pos)
pos = [80+2*roi(3) 300 roi(3)-roi(1)+25 roi(4)-roi(2)+25];
hThresholdDisplay = vision.VideoPlayer('Name', 'Threshold', 'Position', p
```

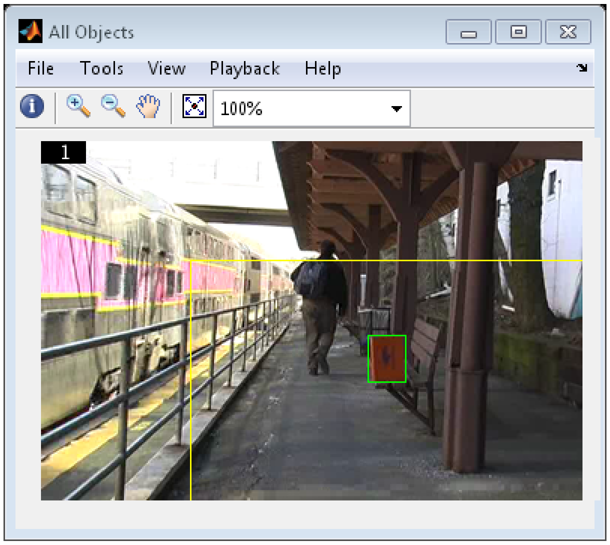
Video Processing Loop

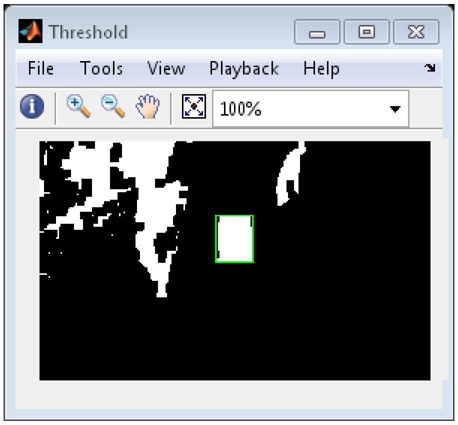
Create a processing loop to perform abandoned object detection on the input video. This loop uses the System objects you instantiated above.

```
firsttime = true;
while ~isDone(hVideoSrc)
    Im = step(hVideoSrc);

    % Select the region of interest from the original video
    OutIm = Im(roi(2):end, roi(1):end, :);

    YCbCr = step(hColorConv, OutIm);
    CbCr  = complex(YCbCr(:,:,2), YCbCr(:,:,3));

    % Store the first video frame as the background
    if firsttime
        firsttime = false;
        BkgY      = YCbCr(:,:,1);
        BkgCbCr   = CbCr;
    end
    SegY    = step(hAutothreshold, abs(YCbCr(:,:,1)-BkgY));
    SegCbCr = abs(CbCr-BkgCbCr) > 0.05;

    % Fill in small gaps in the detected objects
    Segmented = step(hClosing, SegY | SegCbCr);

    % Perform blob analysis
    [Area, Centroid, BBox] = step(hBlob, Segmented);

    % Call the helper function that tracks the identified objects and
    % returns the bounding boxes and the number of the abandoned objects
    [OutCount, OutBBox] = videoobjtracker(Area, Centroid, BBox, maxNumOb
       areaChangeFraction, centroidChangeFraction, maxConsecutiveMiss, .
       minPersistenceRatio, alarmCount);

    % Display the abandoned object detection results
    Imr = step(hDrawRectangles1, Im, OutBBox+PtsOffset);
    Imr(1:15,1:30,:) = 0;
    Imr = step(hDisplayCount, Imr, OutCount);
```

```
        step(hAbandonedObjects, Imr);

        BlobCount = size(BBox,1);

        BBoxOffset = BBox + int32(repmat([roi(1) roi(2) 0  0],[BlobCount 1])
        Imr = step(hDrawRectangles2, Im, BBoxOffset);

        % Display all the detected objects
        Imr(1:15,1:30,:) = 0;
        Imr = step(hDisplayCount, Imr, OutCount);
        Imr = step(hDrawBBox, Imr, roi);
        step(hAllObjects, Imr);

        % Display the segmented video
        SegBBox = PtsOffset;
        SegBBox(1:BlobCount,:) = BBox;
        SegIm = step(hDrawRectangles3, repmat(Segmented,[1 1 3]), SegBBox);
        step(hThresholdDisplay, SegIm);
    end

    release(hVideoSrc);
```







The `Abandoned Objects` window highlights the abandoned objects with a red box. The `All Objects` window marks the region of interest (ROI) with a yellow box and all detected objects with green boxes. The `Threshold` window shows the result of the background subtraction in the ROI.

Annotate Video Files with Frame Numbers

You can use the `vision.TextInserter` System object in MATLAB, or theInsert Text block in a Simulink model, to overlay text on video streams. In this Simulink model example, you add a running count of the number of video frames to a video using the Insert Text block. The model contains the From Multimedia File block to import the video into the Simulink model, a Frame Counter block to count the number of frames in the input video, and two Video Viewer blocks to view the original and annotated videos.

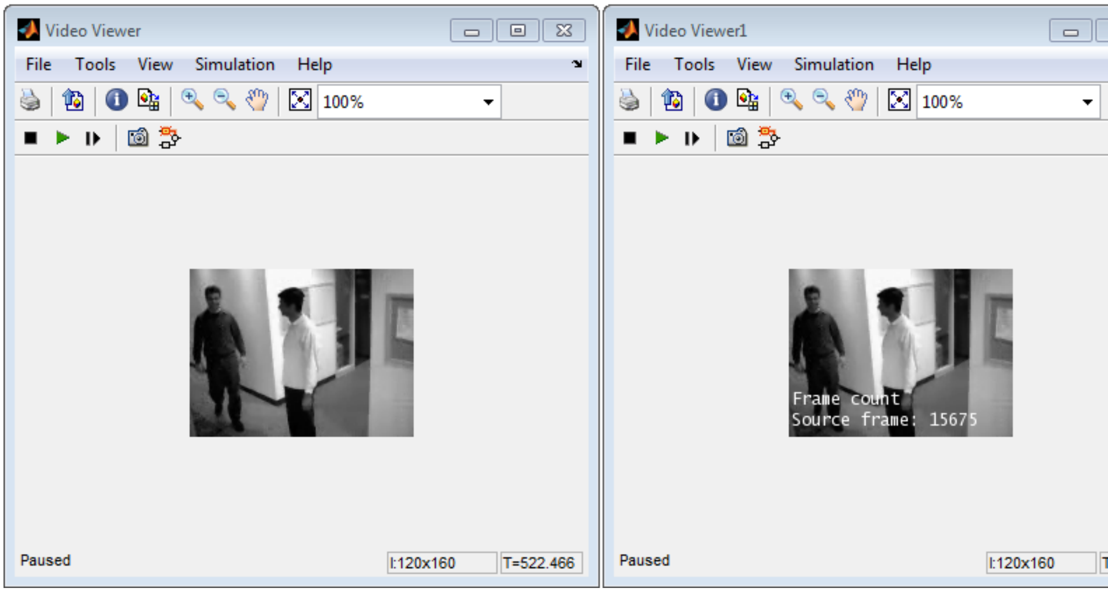
You can open the example model by typing

```
ex_vision_annotate_video_files_with_frame_numbers
```

on the MATLAB command line.

1. Run your model.

2. The model displays the original and annotated videos.



Color Formatting

For this example, the color format for the video was set to `Intensity`, and therefore the color value for the text was set to a scaled value. If instead, you set the color format to `RGB`, then the text value must satisfy this format, and requires a 3-element vector.

Inserting Text

Use the Insert Text block to annotate the video stream with a running frame count. Set the block parameters as follows:

- **Main** pane, **Text** = `['Frame count' sprintf('\n') 'Source frame: %d']`
- **Main** pane, **Color value** = `1`
- **Main** pane, **Location [x y]** = `[2 85]`
- **Font** pane, **Font face** = `LucindaTypewriterRegular`

By setting the **Text** parameter to `['Frame count' sprintf('\n') 'Source frame:  %d']`, you are asking the block to print `Frame count` on one line and the `Source frame:` on a new line. Because you specified `%d`, an ANSI C printf-style format specification, the Variables port appears on the block. The block takes the port input in decimal form and substitutes this input for the `%d` in the string. You used the **Location [x y]** parameter to specify where to print the text. In this case, the location is 85 rows down and 2 columns over from the top-left corner of the image.

Configuration Parameters

Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `inf`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

Registration and Stereo Vision

Feature Detection, Extraction, and Matching

In this section...

Detect Edges in Images

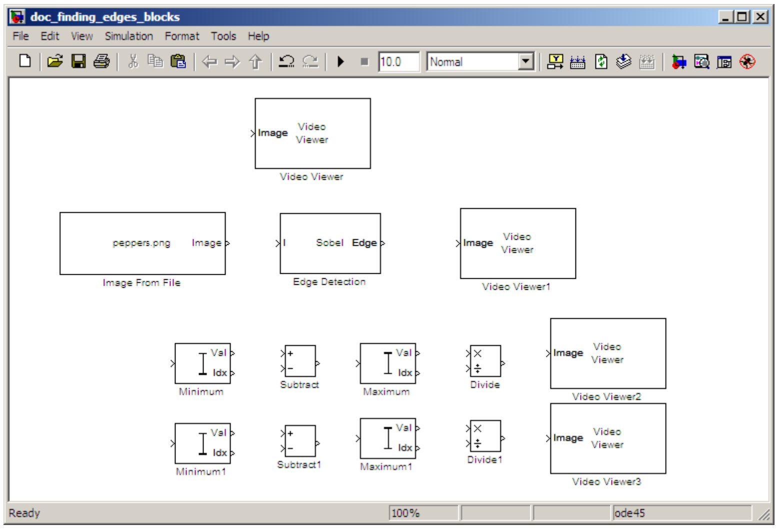
You can use the Edge Detection block to find the edges of objects in an image. This block finds the pixel locations where the magnitude of the gradient of intensity is larger than a threshold value. These locations typically occur at the boundaries of objects. In this section, you use the Edge Detection block to find the edges of rice grains in an intensity image:

1 Create a Simulink model, and add the blocks shown in the following table. Or, start with the model containing the blocks by typing `ex_finding_edges_blocks` at the MATLAB command line.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Edge Detection	Computer Vision System Toolbox > Analysis & Enhancement	1
Minimum	Computer Vision System Toolbox > Statistics	2
Maximum	Computer Vision System Toolbox > Statistics	2
Video Viewer	Computer Vision System Toolbox > Sinks	4

Block	Library	Quantity
Subtract	Simulink > Math Operations	2
Divide	Simulink > Math Operations	2

2 Place the blocks so that your model resembles the following figure.



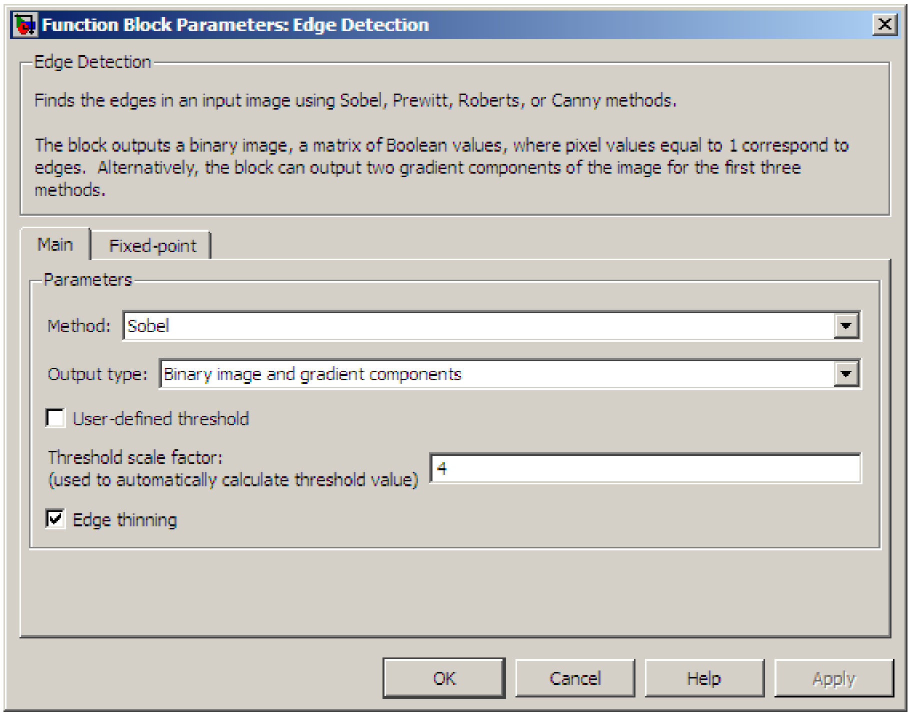
You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

3 Use the Image From File block to import your image. Set the parameters as follows:

- **File name** to `rice.png`.
- **Output data type** to `single`.

4 Use the Edge Detection block to find the edges in the image. Set the block parameters as follows:

- **Output type** = `Binary image and gradient components`
- Select the **Edge thinning** check box.



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.

Main | Fixed-point

Parameters

Method: Sobel

Output type: Binary image and gradient components

☐ User-defined threshold

Threshold scale factor:
(used to automatically calculate threshold value) 4

☑ Edge thinning

The Edge Detection block convolves the input matrix with the Sobel kernel. This calculates the gradient components of the image that correspond to the horizontal and vertical edge responses. The block outputs these components at the **Gh** and **Gv** ports, respectively. Then the block performs a thresholding operation on the gradient components to find the binary image. The binary image is a matrix filled with 1s and 0s. The nonzero elements of this matrix correspond to the edge pixels and the zero elements correspond to the background pixels. The block outputs the binary image at the **Edge** port.

5 View the original image using the Video Viewer block and the binary image using the Video Viewer1 block. Accept the default parameters for both viewers.

6 The matrix values at the **Gv** and **Gh** output ports from of the Edge Detection block are double-precision floating-point. These matrix values need to be scaled between 0 and 1 in order to display them using the Video Viewer blocks. This is done with the Statistics and Math Operation blocks.

7 Use the Minimum blocks to find the minimum value of Gv and Gh matrices. Set the **Mode** parameters to `Value` for both of these blocks.

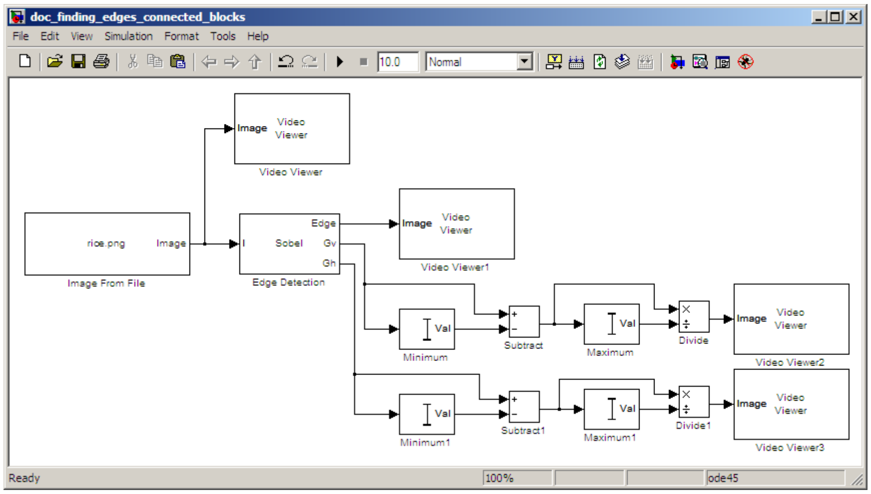
8 Use the Subtract blocks to subtract the minimum values from each element of the Gv and Gh matrices. This process ensures that the minimum value of these matrices is 0. Accept the default parameters.

9 Use the Maximum blocks to find the maximum value of the new Gv and Gh matrices. Set the **Mode** parameters to `Value` for both of these blocks.

10 Use the Divide blocks to divide each element of the Gv and Gh matrices by their maximum value. This normalization process ensures that these matrices range between 0 and 1.Accept the default parameters.

11 View the gradient components of the image using the Video Viewer2 and Video Viewer3 blocks. Accept the default parameters.

12 Connect the blocks as shown in the following figure, or type `ex_finding_edges_connected_blocks` at the MATLAB command line using a connected model.

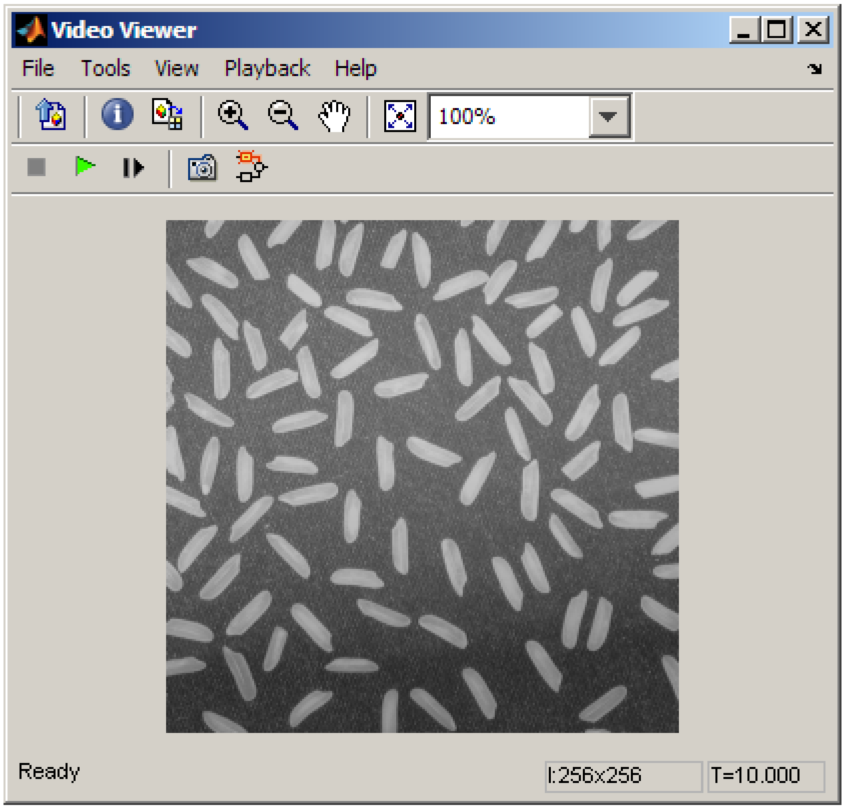


13 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

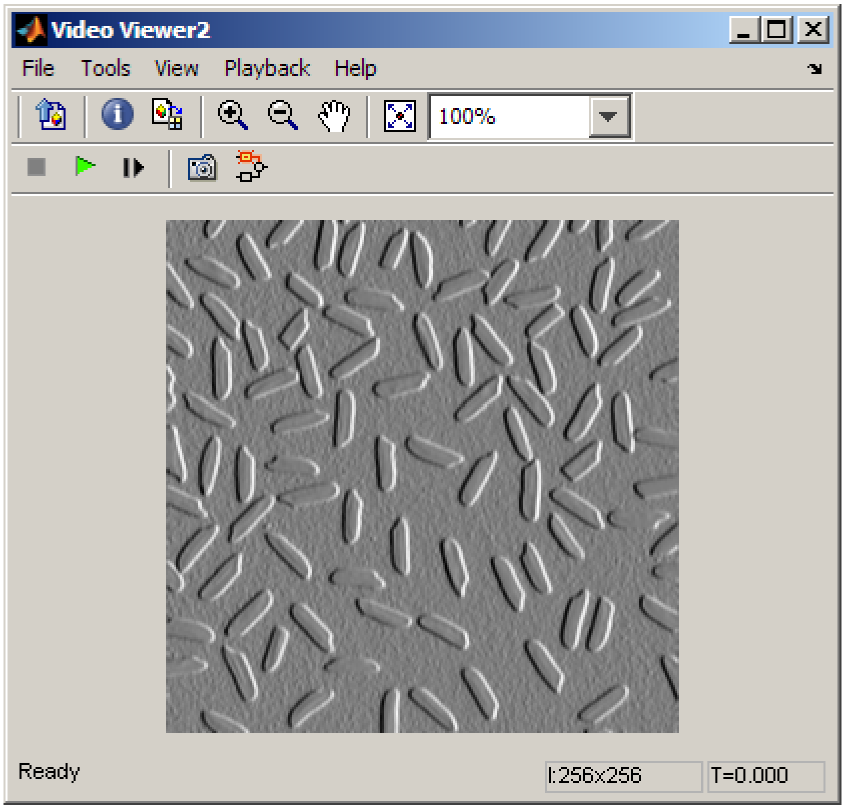
   - **Solver** pane, **Stop time** = `0`
   - **Solver** pane, **Type** = `Fixed-step`
   - **Solver** pane, **Solver** = `Discrete (no continuous states)`
   - **Diagnostics** pane, **Automatic solver parameter selection:** = `none`

14 Run your model.

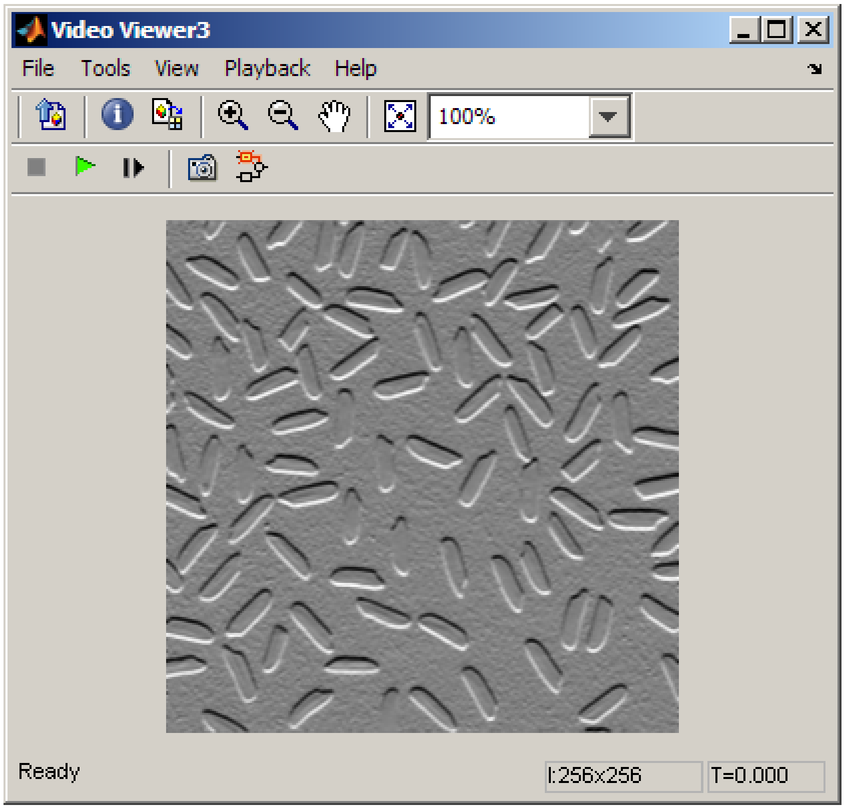
   The Video Viewer window displays the original image. The Video Viewer1 window displays the edges of the rice grains in white and the background in black.



The Video Viewer2 window displays the intensity image of the vertical gradient components of the image. You can see that the vertical edges of the rice grains are darker and more well defined than the horizontal edges.



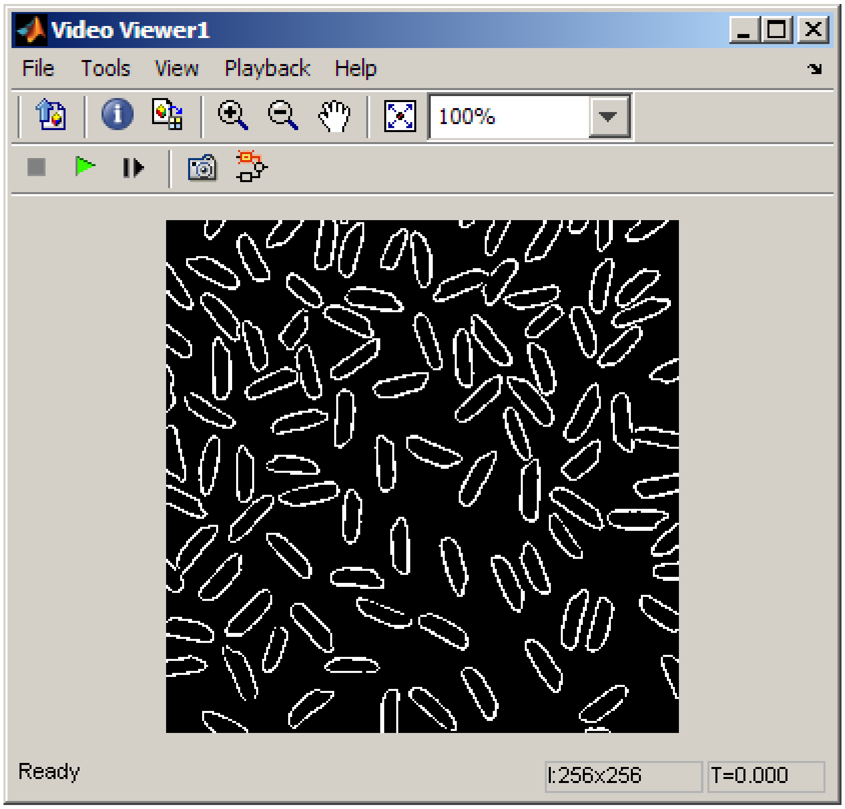
The Video Viewer3 window displays the intensity image of the horizontal gradient components of the image. In this image, the horizontal edges of the rice grains are more well defined.



15 Double-click the Edge Detection block and clear the **Edge thinning** check box.

16 Run your model again.

Your model runs faster because the Edge Detection block is more efficient when you clear the **Edge thinning** check box. However, the edges of rice grains in the Video Viewer window are wider.



You have now used the Edge Detection block to find the object boundaries in an image. For more information on this block, see the Edge Detection block reference page in the *Computer Vision System Toolbox Reference*.

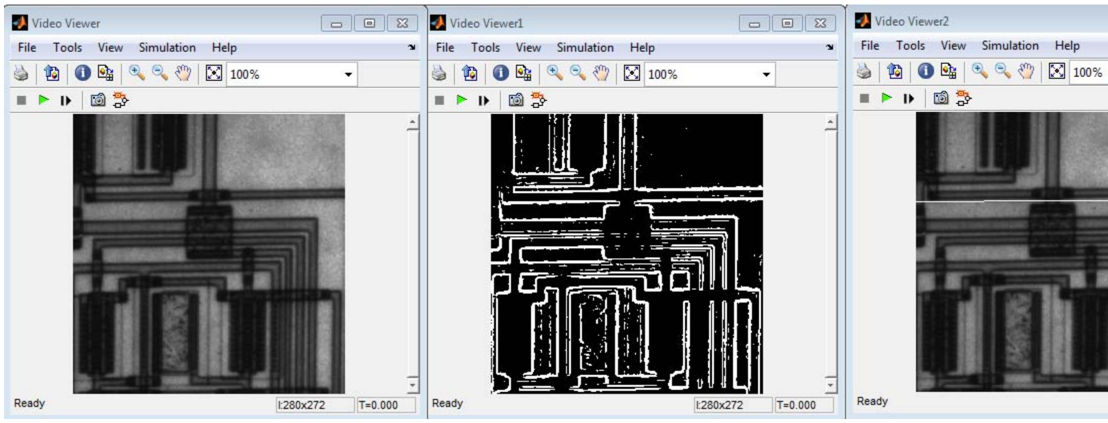
Detect Lines in Images

Finding lines within images enables you to detect, measure, and recognize objects. In this example, you use the Hough Transform, Find Local Maxima, Edge Detectionand Hough Lines blocks to find the longest line in an image.

You can open the model for this example by typing

```
ex_vision_detect_lines_in_image
```

at the MATLAB command line.



The Video Viewer blocks display the original image, the image with all edges found, and the image with the longest line annotated.

The Edge Detection block finds the edges in the intensity image. This process improves the efficiency of the Hough Lines block by reducing the image area over which the block searches for lines. The block also converts the image to a binary image, which is the required input for the Hough Transform block.

For additional examples of the techniques used in this section, see the following list of demos. You can open these demos by typing the demo titles at the MATLAB command prompt:

Demo	**MATLAB**	**Simulink model-based**
Lane Departure Warning System	`videoldws`	`vipldws`
Rotation Correction	`videorotationcorrection`	`viphough`

You can find all demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line.

Setting Block Parameters

Block	Setting
Hough Transform	The Hough Transform block computes the Hough matrix by transforming the input image into the rho-theta parameter space. The block also outputs the rho and theta values associated with the Hough matrix. The parameters are set as follows: • **Theta resolution (radians)** = `pi/360` • Select the **Output theta and rho values** check box.
Find Local Maxima	The Find Local Maxima block finds the location of the maximum value in the Hough matrix. The block parameters are set as follows: • **Maximum number of local maxima** = `1` • **Input is Hough matrix spanning full theta range**
Selector	The Selector blocks separate the indices of the rho and theta values, which the Find Local Maxima block outputs at the **Idx** port. The rho and theta values correspond to the maximum value in the Hough matrix. The Selector blocks parameters are set as follows: • **Number of input dimensions:** `1` • **Index mode** = `One-based`

Block	Setting
	• **Index Option** = `Index vector (port)` • **Input port size** = `2`
Variable Selector	The Variable Selector blocks index into the rho and theta vectors and determine the rho and theta values that correspond to the longest line in the original image. The parameters of the Variable Selector blocks are set as follows: • **Select** = `Columns` • **Index mode** = `One-based`
Hough Lines	The Hough Lines block determines where the longest line intersects the edges of the original image. • **Sine value computation method** = `Trigonometric function`
Draw Shapes	The Draw Shapes block draws a white line over the longest line on the original image. The coordinates are set to superimpose a line on the original image. The block parameters are set as follows: • **Shape** = `Lines` • **Border color** = `White`

Configuration Parameters

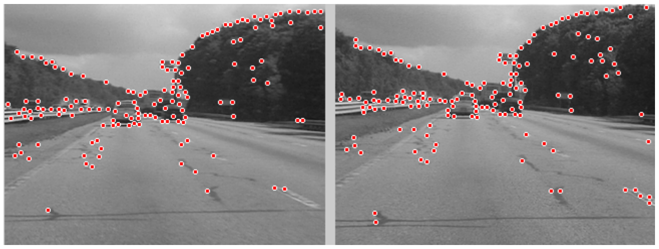
Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`

- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `discrete (no continuous states)`
- **Solver** pane, **Fixed-step size (fundamental sample time):** = `0.2`

Detect Corner Features in an Image

Stabilizing a video that was captured from a jittery or moving platform is an important application in computer vision. One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This example uses corner detection around salient image features.



You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Video Stabilization Using Point Feature Matching demo model directly, by typing `videostabilize_pm` on the MATLAB command line.

Find Possible Point Matches Between Two Images

Stabilizing a video that was captured from a jittery or moving platform is an important application in computer vision. One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This procedure, however, must be bootstrapped with knowledge of where such a salient feature lies in the first video frame. In this example, we explore a method of video stabilization that works without any such a priori knowledge. It instead automatically searches for the "background plane" in a video sequence, and uses its observed distortion to correct for camera motion.

You can launch this example, Video Stabilization Using Point Feature Matching directly, by typing `videostabilize_pm` on the MATLAB command line.

Video mosaicking is the process of stitching video frames together to form a comprehensive view of the scene. The resulting mosaic image is a compact representation of the video data, which is often used in video compression and surveillance applications. This example illustrates how to use the CornerDetector, GeometricTransformEstimator, AlphaBlender, and the GeometricTransformer System objects to create a mosaic image from a video sequence. First, the example identifies the corners in the first reference frame and second video frame. Then, it calculates the projective transformation matrix that best describes the transformation between corner positions in these frames. Finally, the demo overlays the second image onto the first image. The example repeats this process to create a mosaic image of the video scene.

You can launch this example, Video Mosaicking directly, by typing `videomosaicking` on the MATLAB command line.

You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line.

Measure an Angle Between Lines

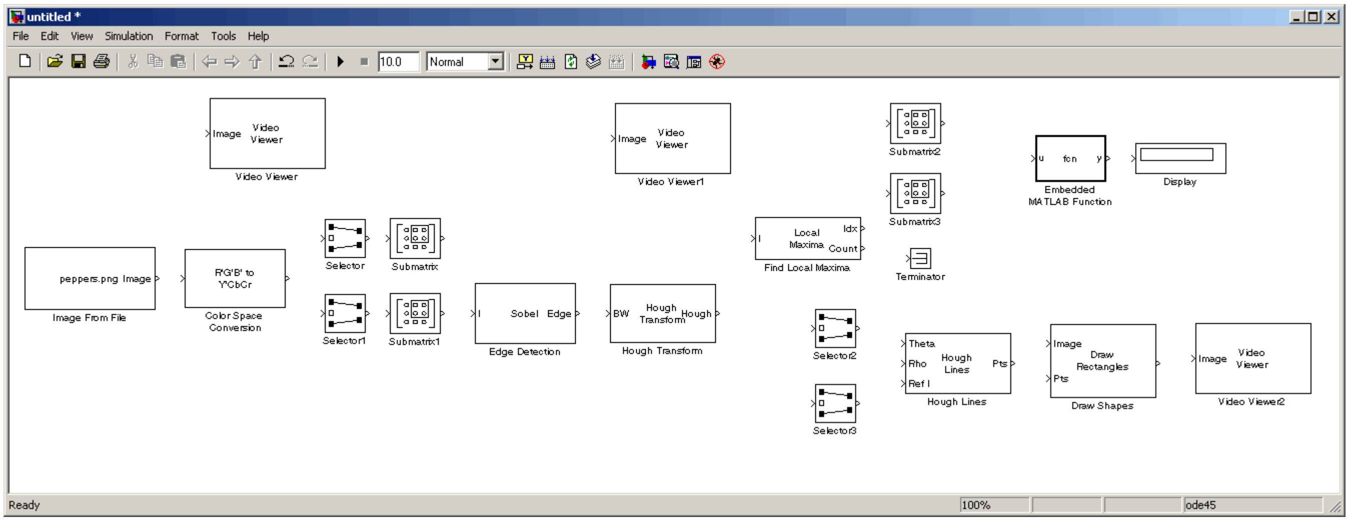
The Hough Transform, Find Local Maxima, and Hough Lines blocks enable you to find lines in images. With the Draw Shapes block, you can annotate images. In the following example, you use these capabilities to draw lines on the edges of two beams and measure the angle between them.

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Color Space Conversion	Computer Vision System Toolbox > Conversions	1

Block	Library	Quantity
Edge Detection	Computer Vision System Toolbox > Analysis & Enhancement	1
Hough Transform	Computer Vision System Toolbox > Transforms	1
Find Local Maxima	Computer Vision System Toolbox > Statistics	1
Draw Shapes	Computer Vision System Toolbox > Text & Graphics	1
Video Viewer	Computer Vision System Toolbox > Sinks	3
Hough Lines	Computer Vision System Toolbox > Transforms	1
Submatrix	DSP System Toolbox > Math Functions > Matrices and Linear Algebra > Matrix Operations	4
Terminator	Simulink > Sinks	1
Selector	Simulink > Signal Routing	4
MATLAB Function	Simulink > User-Defined Functions	1
Display	Simulink > Sinks	1

2 Position the blocks as shown in the following figure.



3 Use the Image From File block to import an image into the Simulink model. Set the parameters as follows:

- **File name** = `gantrycrane.png`
- **Sample time** = `1`

4 Use the Color Space Conversion block to convert the RGB image into the Y'CbCr color space. You perform this conversion to separate the luma information from the color information. Accept the default parameters.

Note In this example, you segment the image using a thresholding operation that performs best on the Cb channel of the Y'CbCr color space.

---

5 Use the Selector and Selector1 blocks to separate the Y' (luminance) and Cb (chrominance) components from the main signal.

The Selector block separates the Y' component from the entire signal. Set its block parameters as follows:

- **Number of input dimensions** = `3`
- **Index mode** = `One-based`

- **1**
  - **Index Option** = `Select all`
- **2**
  - **Index Option** = `Select all`
- **3**
  - **Index Option** = `Index vector (dialog)`
  - **Index** = `1`

The Selector1 block separates the Cb component from the entire signal. Set its block parameters as follows:

- **Number of input dimensions** = `3`
- **Index mode** = `One-based`
- **1**
  - **Index Option** = `Select all`
- **2**
  - **Index Option** = `Select all`
- **3**
  - **Index Option** = `Index vector (dialog)`
  - **Index** = `2`

6 Use the Submatrix and Submatrix1 blocks to crop the Y' and Cb matrices to a particular region of interest (ROI). This ROI contains two beams that are at an angle to each other. Set the parameters as follows:

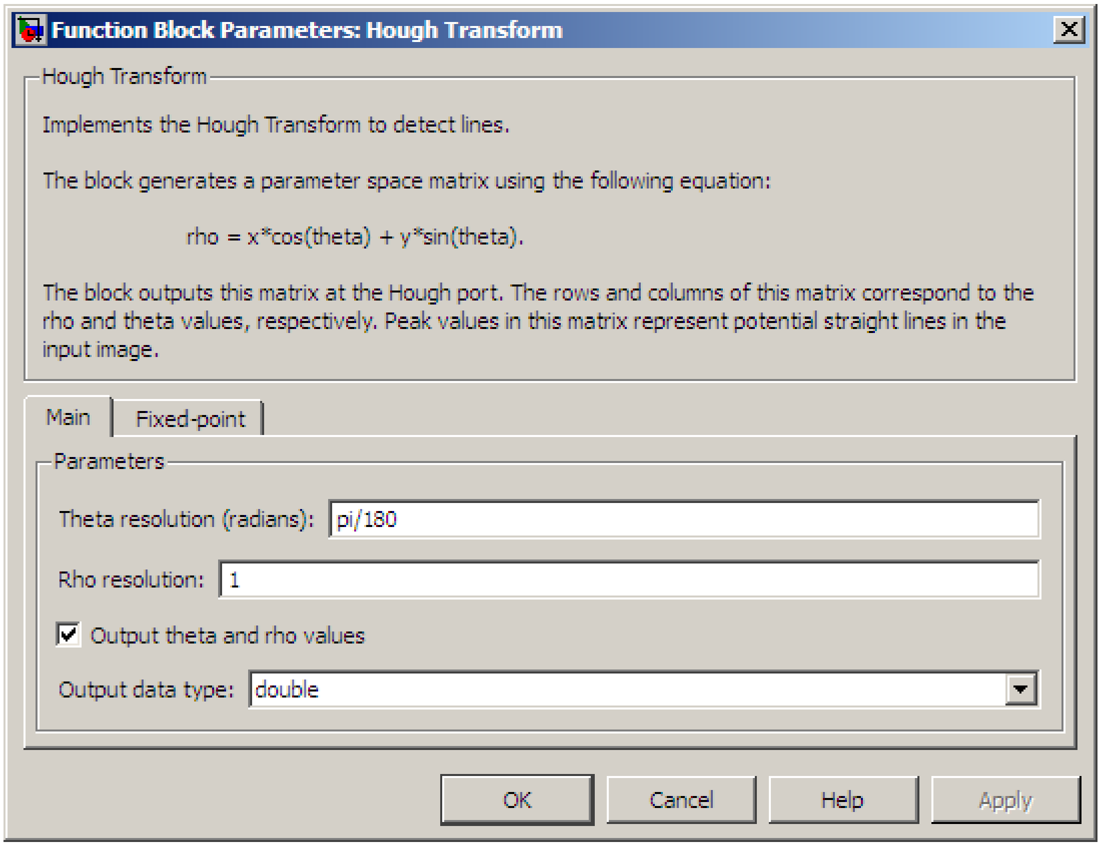
- **Starting row** = `Index`
- **Starting row index** = `66`
- **Ending row** = `Index`
- **Ending row index** = `150`
- **Starting column** = `Index`
- **Starting column index** = `325`

- **Ending column** = `Index`
- **Ending column index** = `400`

7 Use the Edge Detection block to find the edges in the Cb portion of the image. This block outputs a binary image. Set the **Threshold scale factor** parameter to `1`.

8 Use the Hough Transform block to calculate the Hough matrix, which gives you an indication of the presence of lines in an image. Select the **Output theta and rho values** check box as shown in the following figure.

Note In step 11, you find the theta and rho values that correspond to the peaks in the Hough matrix.



Function Block Parameters: Hough Transform

Hough Transform

Implements the Hough Transform to detect lines.

The block generates a parameter space matrix using the following equation:

rho = x*cos(theta) + y*sin(theta).

The block outputs this matrix at the Hough port. The rows and columns of this matrix correspond to the rho and theta values, respectively. Peak values in this matrix represent potential straight lines in the input image.

Theta resolution (radians): pi/180

Rho resolution: 1

[✓] Output theta and rho values

Output data type: double

9 Use the Find Local Maxima block to find the peak values in the Hough matrix. These values represent potential lines in the input image. Set the parameters as follows:

- **Neighborhood size** = `[11 11]`
- **Input is Hough matrix spanning full theta range** = selected

Because you are expecting two lines, leave the **Maximum number of local maxima (N)** parameter set to `2`, and connect the Count port to the Terminator block.

10 Use the Submatrix2 block to find the indices that correspond to the theta values of the two peak values in the Hough matrix. Set the parameters as follows:

- **Starting row** = `Index`
- **Starting row index** = `2`
- **Ending row** = `Index`
- **Ending row index** = `2`

The Idx port of the Find Local Maxima block outputs a matrix whose second row represents the One-based indices of the theta values that correspond to the peaks in the Hough matrix. Now that you have these indices, you can use a Selector block to extract the corresponding theta values from the vector output of the Hough Transform block.

11 Use the Submatrix3 block to find the indices that correspond to the rho values of the two peak values in the Hough matrix. Set the parameters as follows:

- **Ending row** = `Index`
- **Ending row index** = `1`

The Idx port of the Find Local Maxima block outputs a matrix whose first row represents the One-based indices of the rho values that correspond to the peaks in the Hough matrix. Now that you have these indices, you can use a Selector block to extract the corresponding rho values from the vector output of the Hough Transform block.

12 Use the Selector2 and Selector3 blocks to find the theta and rho values that correspond to the peaks in the Hough matrix. These values, output by the Hough Transform block, are located at the indices output by the Submatrix2 and Submatrix3 blocks. Set both block parameters as follows:

- **Index mode** = `One-based`
- **1**
  - **Index Option** = `Index vector (port)`
- **Input port size** = `-1`

You set the **Index mode** to `One-based` because the Find Local Maxima block outputs One-based indices at the Idx port.

13 Use the Hough Lines block to find the Cartesian coordinates of lines that are described by rho and theta pairs. Set the **Sine value computation method** parameter to `Trigonometric function`.

14 Use the Draw Shapes block to draw the lines on the luminance portion of the ROI. Set the parameters as follows:

- **Shape** = `Lines`
- **Border color** = `White`

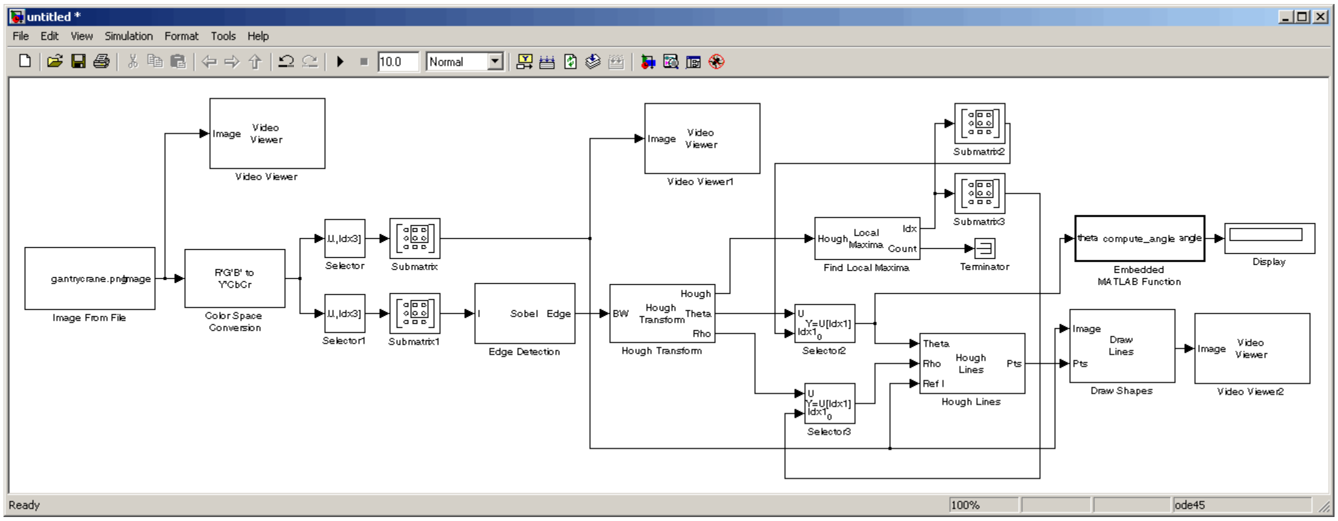
15 Use the MATLAB Function block to calculate the angle between the two lines. Copy and paste the following code into the block:

```
function angle = compute_angle(theta)

%Compute the angle value in degrees
angle = abs(theta(1)-theta(2))*180/pi;
%Always return an angle value less than 90 degrees
if (angle>90)
    angle = 180-angle;
end
```

16 Use the Display block to view the angle between the two lines. Accept the default parameters.

17 Use the Video Viewer blocks to view the original image, the ROI, and the annotated ROI. Accept the default parameters.

18 Connect the blocks as shown in the following figure.

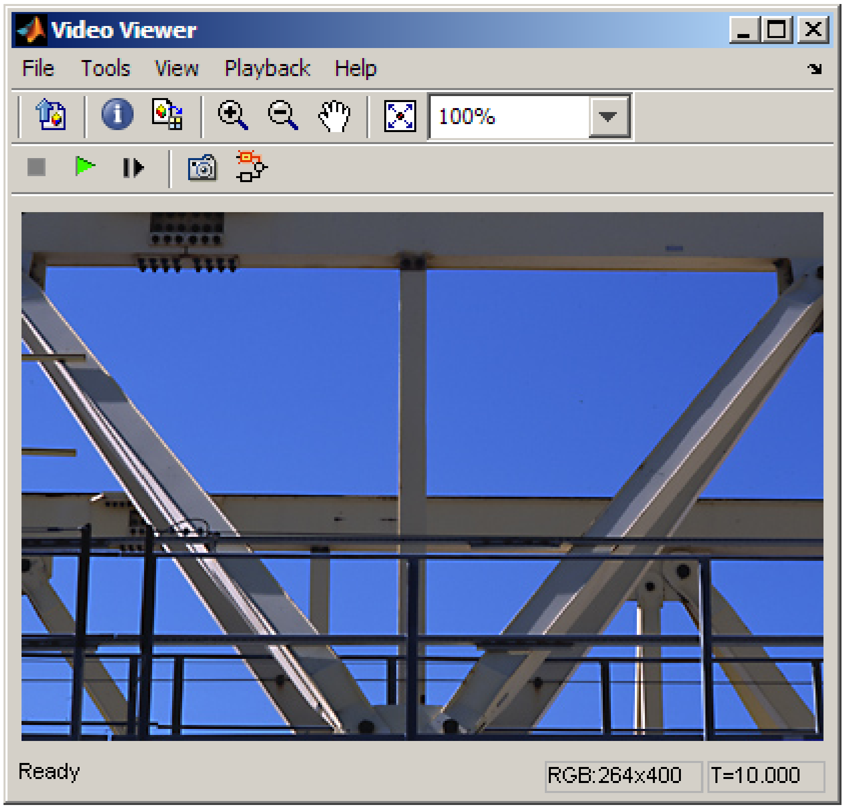


19 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

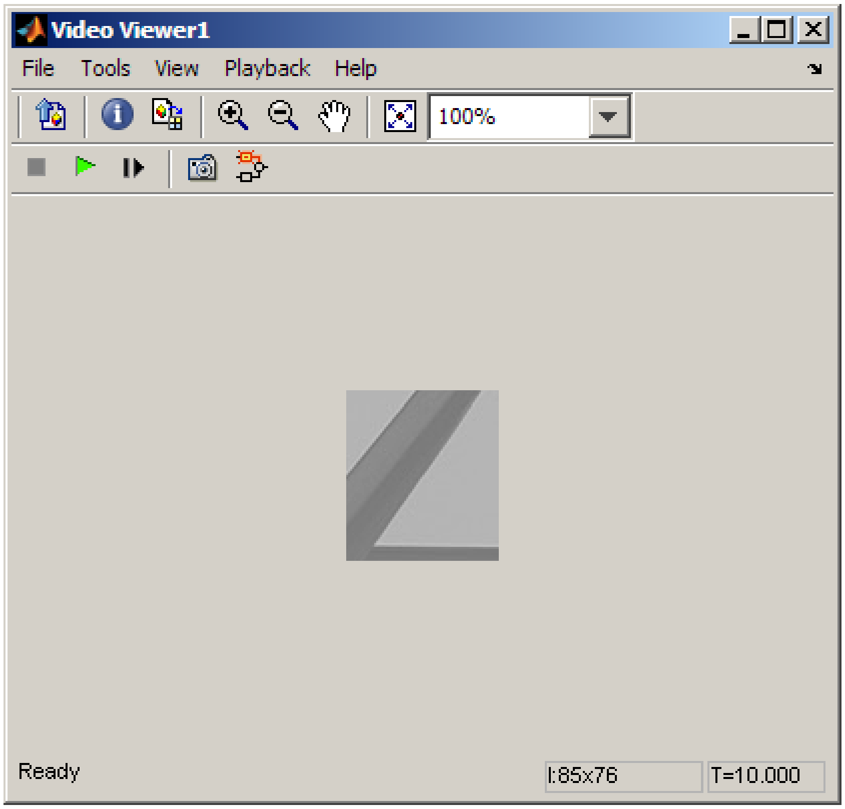
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

20 Run the model.

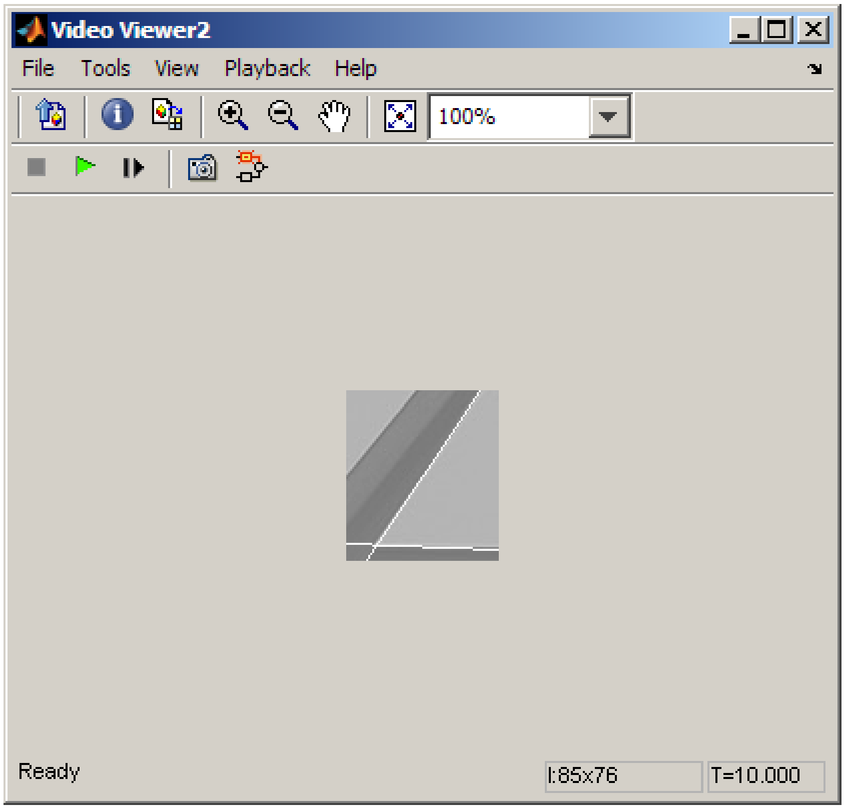
The Video Viewer window displays the original image.



The Video Viewer1 window displays the ROI where two beams intersect.



The Video Viewer2 window displays the ROI that has been annotated with two white lines.



The Display block shows a value of 58, which is the angle in degrees between the two lines on the annotated ROI.

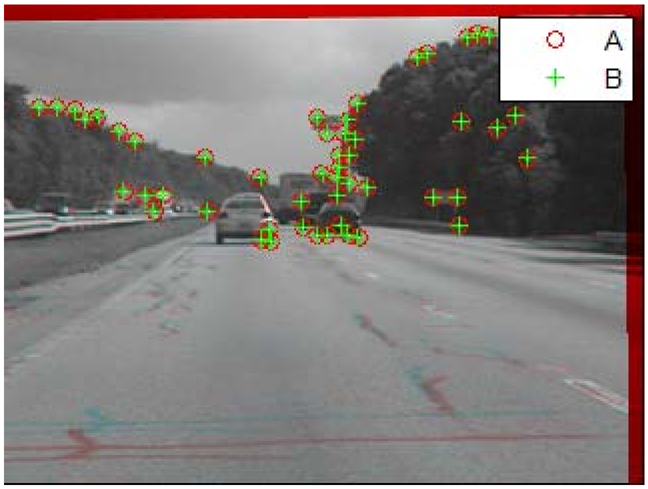
You have now annotated an image with two lines and measured the angle between them. For additional information, see the Hough Transform, Find Local Maxima, Hough Lines, and Draw Shapes block reference pages.

Image Registration

In this section...

Automatically Determine Geometric Transform for Image Registration

Stabilizing a video that was captured from a jittery or moving platform is an important application in computer vision. One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This procedure, however, must be bootstrapped with knowledge of where such a salient feature lies in the first video frame. In this example, we explore a method of video stabilization that works without any such a priori knowledge. It instead automatically searches for the "background plane" in a video sequence, and uses its observed distortion to correct for camera motion.



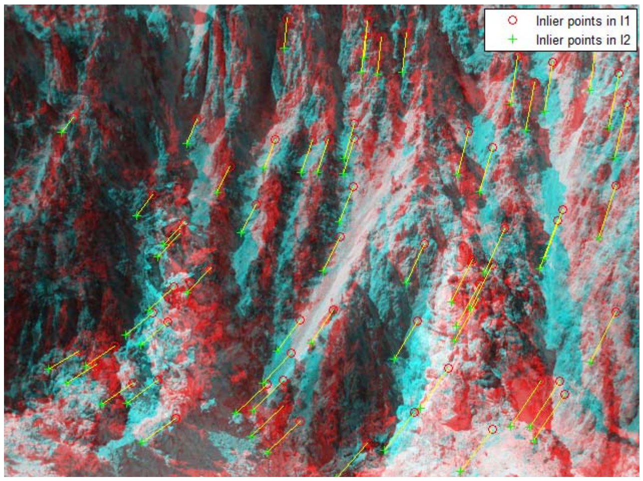
This stabilization algorithm involves two steps. First, we determine the affine image transformations between all neighboring frames of a video sequence using a Random Sampling and Consensus (RANSAC) [1] procedure applied to point correspondences between two images. Second, we warp the video frames to achieve a stabilized video. We use System objects in the Computer Vision System Toolbox™, both for the algorithm and for display.

You can launch this example, Video Stabilization Using Point Feature Matching directly, by typing `videostabilize_pm` on the MATLAB command line.

Transform Images and Display Registration Results

Rectification is the process of transforming stereo images, such that the corresponding points have the same row coordinates in the two images. It is a useful procedure in stereo vision, as the 2-D stereo correspondence problem is reduced to a 1-D problem when rectified image pairs are used.

The Image Rectification demo automatically registers and rectifies stereo images. This example detects corners in stereo images, matches the corners, computes the fundamental matrix, and then rectifies the images.

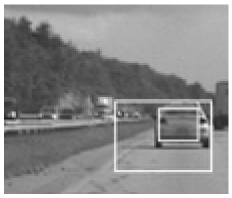


You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Image

Rectification demo model directly, by typing `videorectification` on the MATLAB command line.

Remove the Effect of Camera Motion from a Video Stream.

The video stabilization demo tracks a license plate of a vehicle while reducing the effect of camera motion from a video stream.



In the first video frame, the model defines the target to track. In this case, it is the back of a car and the license plate. It also establishes a dynamic search region, where the last known target location determines the position.

You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can launch the Video Stabilization model directly by typing `vipstabilize` on the MATLAB command line.

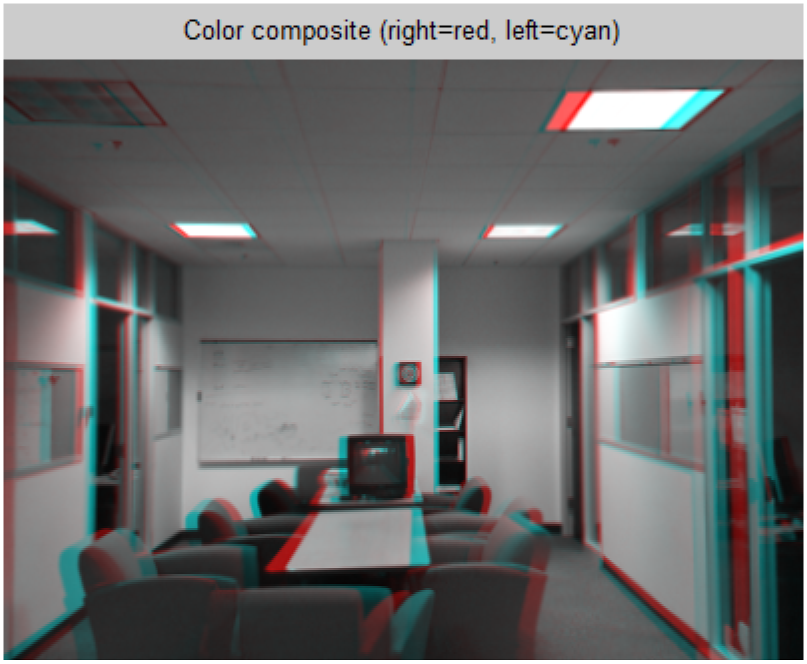
Stereo Vision

In this section...

Compute Disparity Depth Map

Stereo vision is the process of recovering depth from camera images by comparing two or more views of the same scene. Simple, binocular stereo uses only two images, typically taken with parallel cameras that were separated by a horizontal distance known as the "baseline." The output of the stereo computation is a disparity map (which is translatable to a range image) which tells how far each point in the physical scene was from the camera.

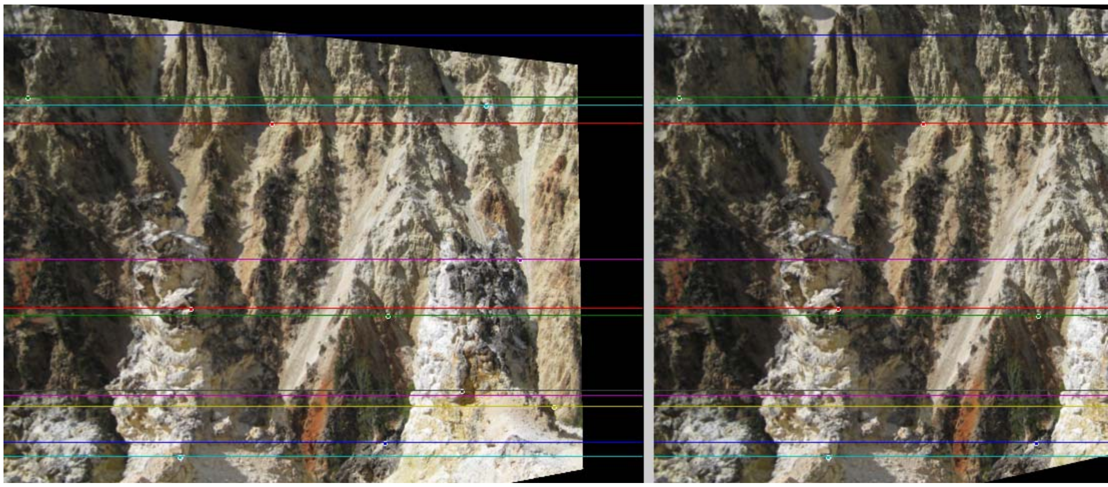
In this example, we use MATLAB® and the Computer Vision System Toolbox™ to compute the depth map between two rectified stereo images.



You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Stereo Vision demo model directly, by typing `videostereo` on the MATLAB command line.

Find Fundamental Matrix Describing Epipolar Geometry

In computer vision, the fundamental matrix is a 3×3 matrix which relates corresponding points in stereo images. When two cameras view a 3D scene from two distinct positions, there are a number of geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points. Two images of the same scene are related by epipolar geometry.



This example takes two stereo images, computes the fundamental matrix from their corresponding points, displays the original stereo images, corresponding points, and epipolar lines for this and for the rectified images.

Load stereo images and cnovert them to double precision:

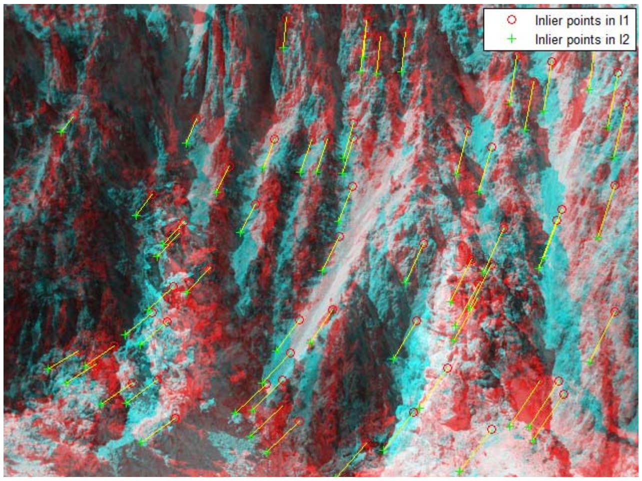
```
% Load the stereo images and convert them to double precision.
    I1 = imread('yellowstone_left.png');
    I2 = imread('yellowstone_right.png');

    % Load the points which are already matched.
    load yellowstone_matched_points;

    % Compute the fundamental matrix from the corresponding points.
    f= estimateFundamentalMatrix(matched_points1, matched_points2,...
      'Method', 'Norm8Point');

    % Display the original stereo images, corresponding points, and
    % epipolar lines.
    cvexShowStereoImages('Original image 1', 'Original image 2', ...
      I1, I2, matched_points1, matched_points2, f);
```

You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line.

Rectify Stereo Images

Rectification is the process of transforming stereo images, such that the corresponding points have the same row coordinates in the two images. It is a useful procedure in stereo vision, as the 2-D stereo correspondence problem is reduced to a 1-D problem when rectified image pairs are used.

The Image Rectification demo automatically registers and rectifies stereo images. This example detects corners in stereo images, matches the corners, computes the fundamental matrix, and then rectifies the images.

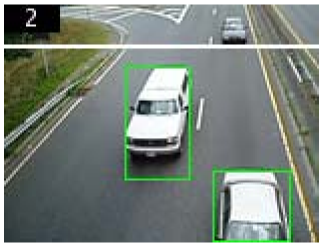


You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Image Rectification demo model directly, by typing `videorectification` on the MATLAB command line.

Motion Analysis and Tracking

Detect and Track Moving Objects Using Gaussian Mixture Models

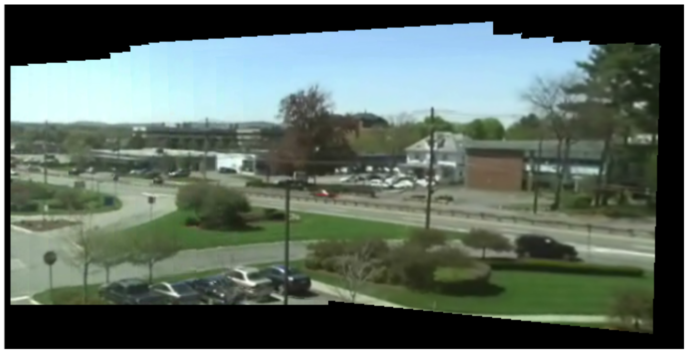
This example illustrates how to detect cars in a video sequence using foreground detection based on Gaussian mixture models (GMMs). After foreground detection, the example processes the binary foreground images using blob analysis. Finally, bounding boxes are drawn around the detected cars.



You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Tracking Cars Using Gaussian Mixture Models demo model directly, by typing `videotrafficgmm` at the MATLAB command line.

Video Mosaicking

Video mosaicking is the process of stitching video frames together to form a comprehensive view of the scene. The resulting mosaic image is a compact representation of the video data, which is often used in video compression and surveillance applications.



You can find demos for the Computer Vision System Toolbox by typing `visiondemos` at the MATLAB command line. You can also launch the Video Mosaicking demo model directly, by typing `videomosaicking` at the MATLAB command line.

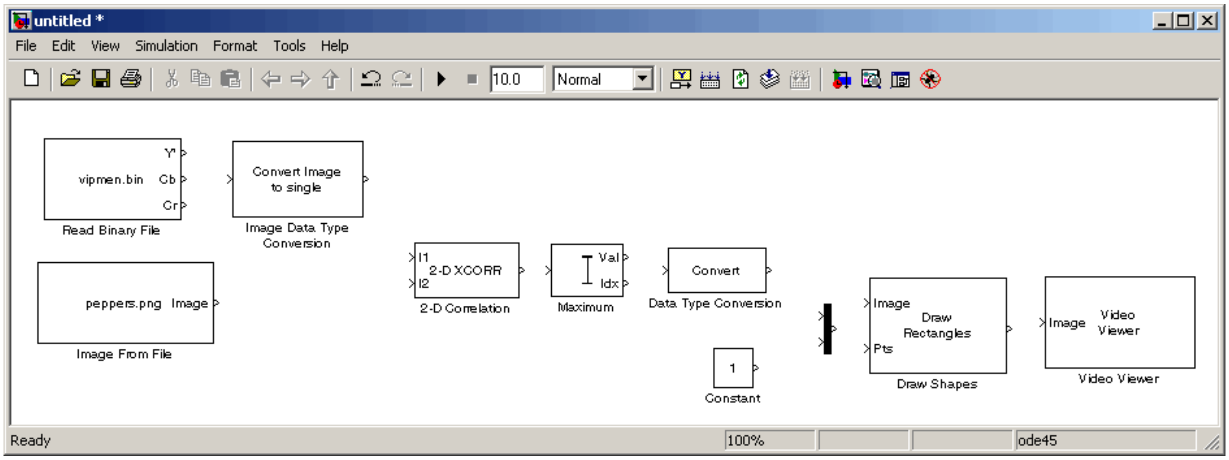
Track an Object Using Correlation

In this example, you use the 2-D Correlation, 2-D Maximum, and Draw Shapes blocks to find and indicate the location of a sculpture in each video frame:

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Read Binary File	Computer Vision System Toolbox > Sources	1
Image Data Type Conversion	Computer Vision System Toolbox > Conversions	1
Image From File	Computer Vision System Toolbox > Sources	1
2-D Correlation	Computer Vision System Toolbox > Statistics	1
2-D Maximum	Computer Vision System Toolbox > Statistics	1
Draw Shapes	Computer Vision System Toolbox > Text & Graphics	1
Video Viewer	Computer Vision System Toolbox > Sinks	1
Data Type Conversion	Simulink > Signal Attributes	1
Constant	Simulink > Sources	1
Mux	Simulink > Signal Routing	1

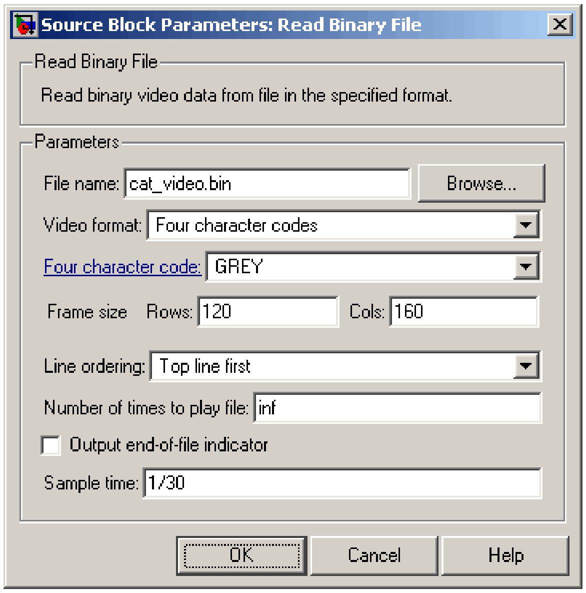
2 Position the blocks as shown in the following figure.



You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

3 Use the Read Binary File block to import a binary file into the model. Set the block parameters as follows:

- **File name** = `cat_video.bin`
- **Four character code** = `GREY`
- **Number of times to play file** = `inf`
- **Sample time** = `1/30`


Source Block Parameters: Read Binary File

Read Binary File

Read binary video data from file in the specified format.

Parameters

File name: cat_video.bin

Video format: Four character codes

Four character code: GREY

Frame size Rows: 120 Cols: 160

Line ordering: Top line first

Number of times to play file: inf

☐ Output end-of-file indicator

Sample time: 1/30

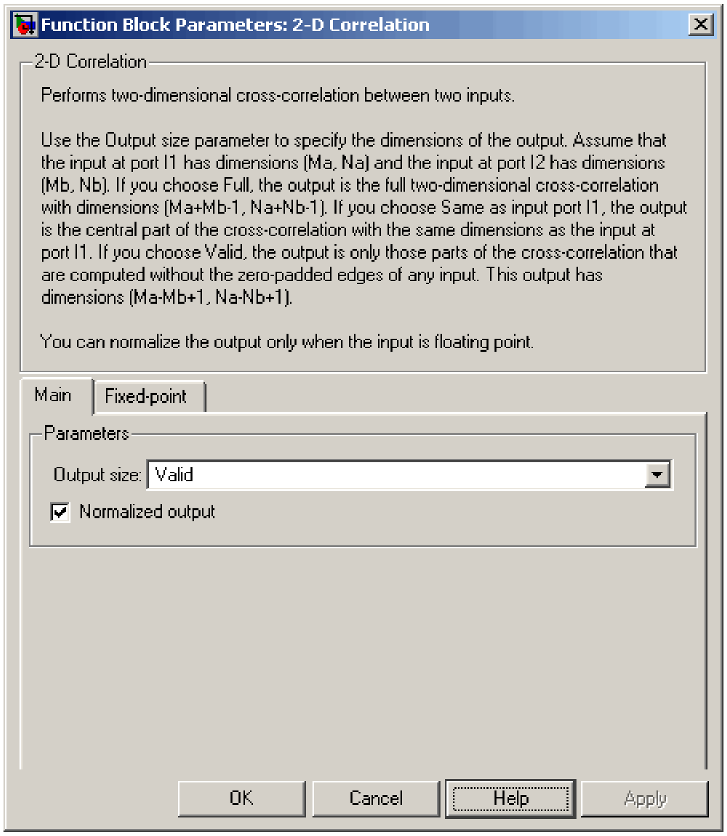
4 Use the Image Data Type Conversion block to convert the data type of the video to single-precision floating point. Accept the default parameter.

5 Use the Image From File block to import the image of the cat sculpture, which is the object you want to track. Set the block parameters as follows:

- **Main** pane, **File name** = `cat_target.png`
- **Data Types** pane, **Output data type** = `single`

6 Use the 2-D Correlation block to determine the portion of each video frame that best matches the image of the cat sculpture. Set the block parameters as follows:

- **Output size** = `Valid`
- Select the **Normalized output** check box.


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.

Parameters

Output size: Valid

☑ Normalized output

Because you chose `Valid` for the **Output size** parameter, the block outputs only those parts of the correlation that are computed without the zero-padded edges of any input.

7 Use the 2-D Maximum block to find the index of the maximum value in each input matrix. Set the **Mode** parameter to `Index`.

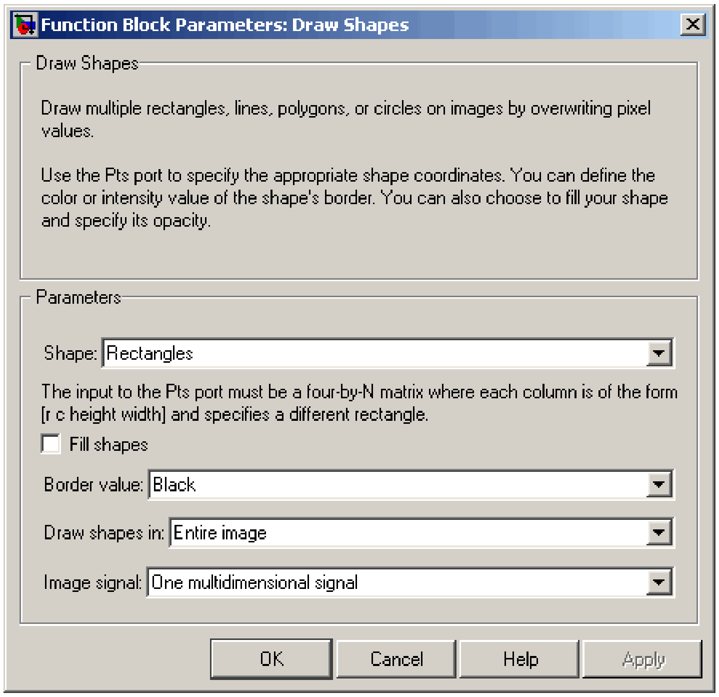
The block outputs the zero-based location of the maximum value as a two-element vector of 32-bit unsigned integers at the Idx port.

8 Use the Data Type Conversion block to change the index values from 32-bit unsigned integers to single-precision floating-point values. Set the **Output data type** parameter to `single`.

9 Use the Constant block to define the size of the image of the cat sculpture. Set the **Constant value** parameter to `single([41 41])`.

10 Use the Mux block to concatenate the location of the maximum value and the size of the image of the cat sculpture into a single vector. You use this vector to define a rectangular region of interest (ROI) that you pass to the Draw Shapes block.

11 Use the Draw Shapes block to draw a rectangle around the portion of each video frame that best matches the image of the cat sculpture. Accept the default parameters.


Function Block Parameters: Draw Shapes

Draw Shapes

Draw multiple rectangles, lines, polygons, or circles on images by overwriting pixel values.

Use the Pts port to specify the appropriate shape coordinates. You can define the color or intensity value of the shape's border. You can also choose to fill your shape and specify its opacity.

Parameters

Shape: Rectangles

The input to the Pts port must be a four-by-N matrix where each column is of the form [r c height width] and specifies a different rectangle.

☐ Fill shapes

Border value: Black

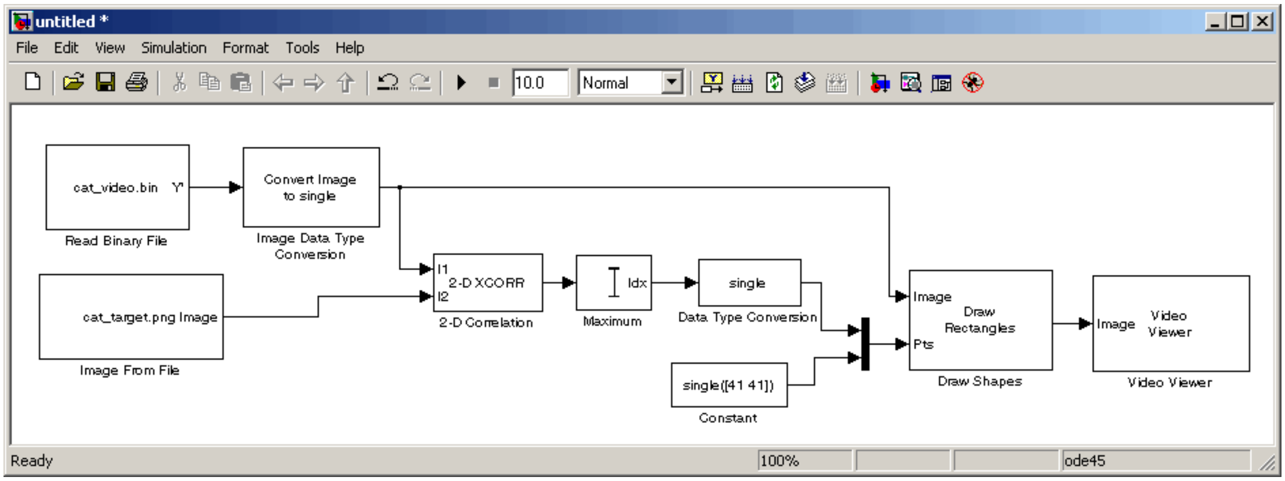
Draw shapes in: Entire image

Image signal: One multidimensional signal

12 Use the Video Viewer block to display the video stream with the ROI displayed on it. Accept the default parameters.

The Video Viewer block automatically displays the video in the Video Viewer window when you run the model. Because the image is represented by single-precision floating-point values, a value of 0 corresponds to black and a value of 1 corresponds to white.

13 Connect the blocks as shown in the following figure.

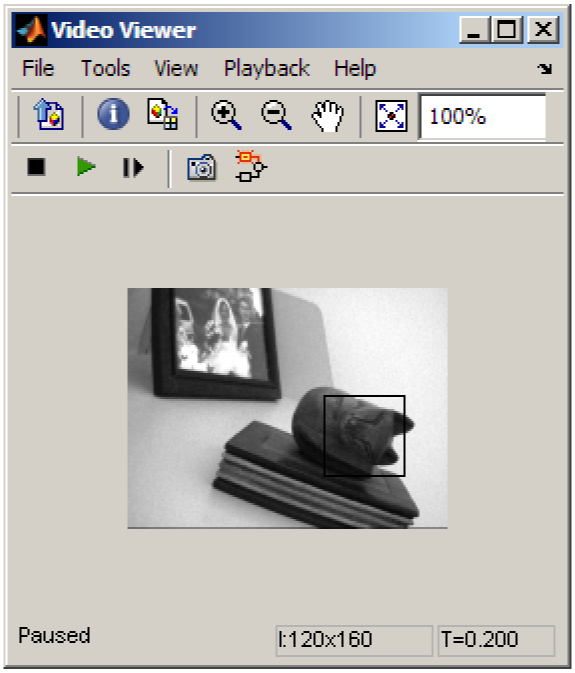


14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

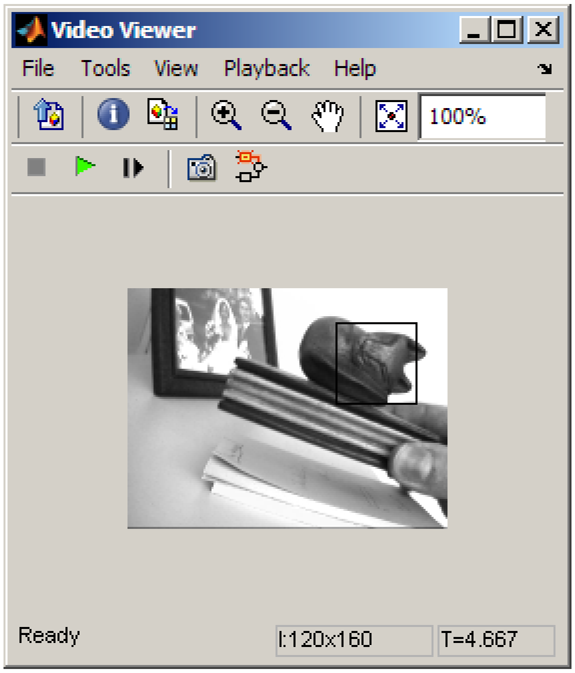
- **Solver** pane, **Stop time** = `inf`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

15 Run the simulation.

The video is displayed in the Video Viewer window and a rectangular box appears around the cat sculpture. To view the video at its true size, right-click the window and select **Set Display To True Size**.



As the video plays, you can watch the rectangular ROI follow the sculpture as it moves.



In this example, you used the 2-D Correlation, 2-D Maximum, and Draw Shapes blocks to track the motion of an object in a video stream. For more information about these blocks, see the 2-D Correlation, 2-D Maximum, and Draw Shapes block reference pages.

Note This example model does not provide an indication of whether or not the sculpture is present in each video frame. For an example of this type of model, type `vippattern` at the MATLAB command prompt.

Create a Panoramic Scene

The motion estimation subsystem for the Panorama Creation demo uses template matching. This model uses the Template Matching block to estimate the motion between consecutive video frames. It then computes the motion vector of a particular block in the current frame with respect to the previous frame. The model uses this motion vector to align consecutive frames of the video to form a panoramic picture.

You can find demos for the Computer Vision System Toolbox by typing `visiondemos` on the MATLAB command line. You can launch the Panorama model directly by typing `vippanorama` on the MATLAB command line.

Geometric Transformations

Rotate an Image

You can use the Rotate block to rotate your image or video stream by a specified angle. In this example, you learn how to use the Rotate block to continuously rotate an image:

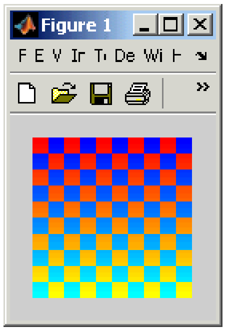
1 Define an RGB image in the MATLAB workspace. At the MATLAB command prompt, type

```
I = checker_board;
```

`I` is a 100-by-100-by-3 array of double-precision values. Each plane of the array represents the red, green, or blue color values of the image.

2 To view the image this matrix represents, at the MATLAB command prompt, type

```
imshow(I)
```

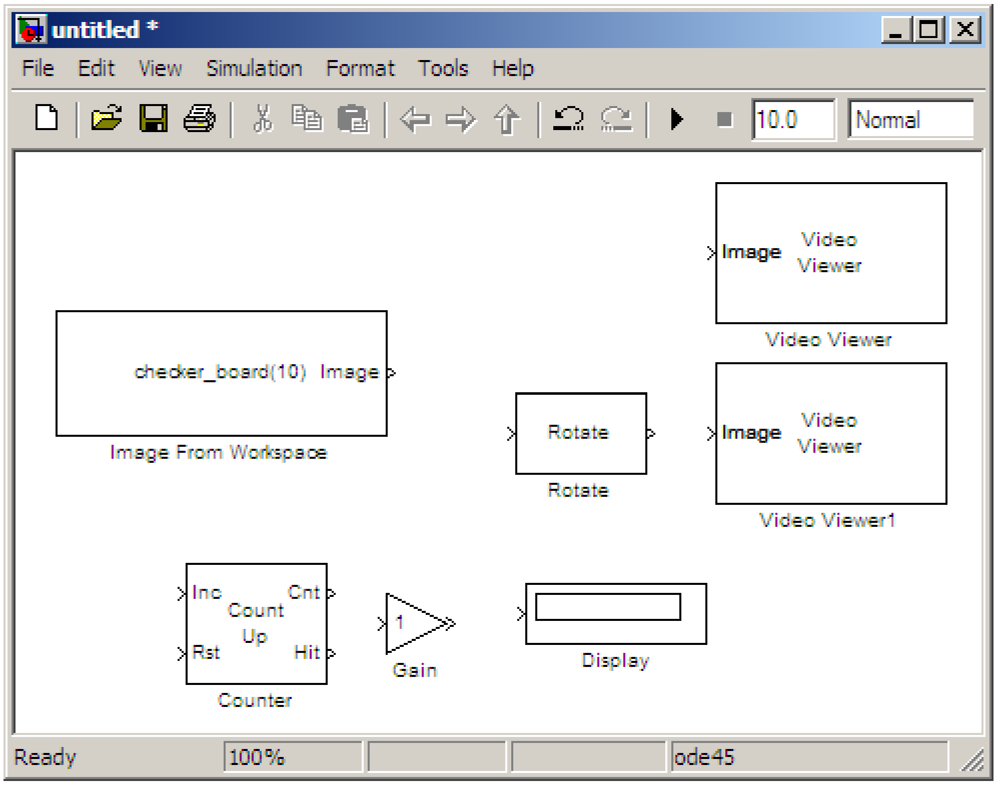


3 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From Workspace	Computer Vision System Toolbox > Sources	1
Rotate	Computer Vision System Toolbox > Geometric Transformations	1

Block	Library	Quantity
Video Viewer	Computer Vision System Toolbox > Sinks	2
Gain	Simulink > Math Operations	1
Display	DSP System Toolbox > Signal Processing Sinks	1
Counter	DSP System Toolbox > Signal Management > Switches and Counters	1

4 Position the blocks as shown in the following figure.



You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

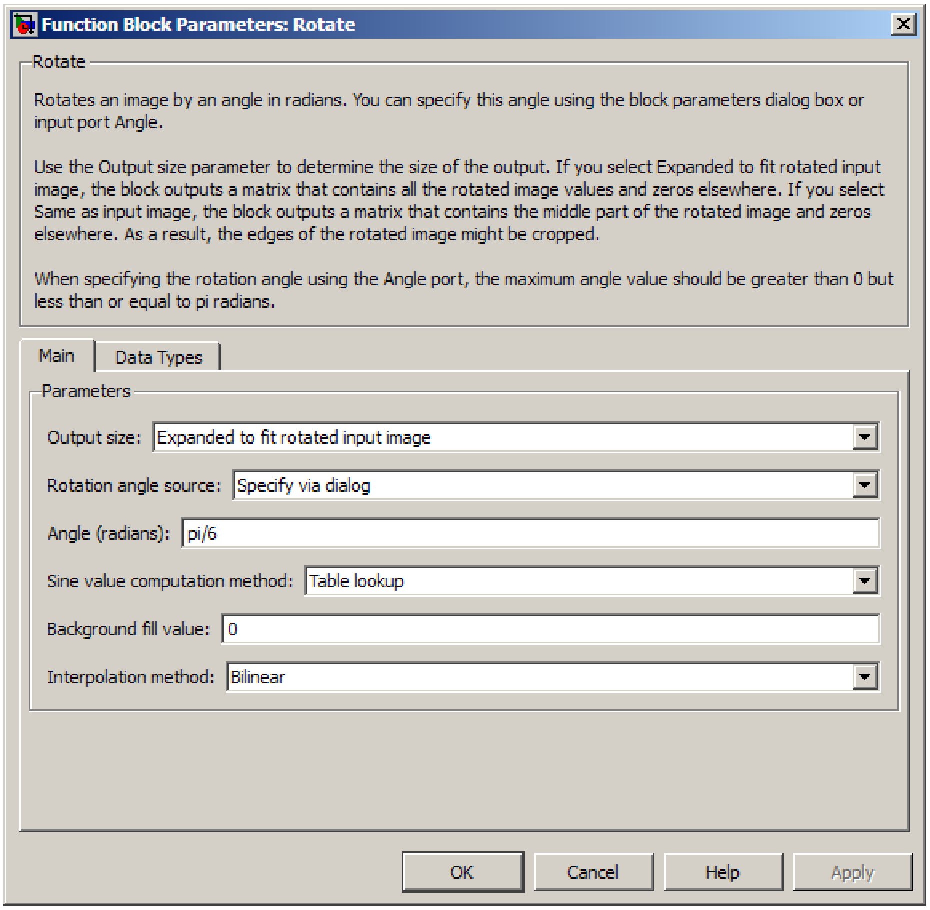
5. Use the Image From Workspace block to import the RGB image from the MATLAB workspace. On the Main pane, set the **Value** parameter to `I`.Each plane of the array represents the red, green, or blue color values of the image.

6. Use the Video Viewer block to display the original image. Accept the default parameters.

   The Video Viewer block automatically displays the original image in the Video Viewer window when you run the model. Because the image is represented by double-precision floating-point values, a value of 0 corresponds to black and a value of 1 corresponds to white.

7. Use the Rotate block to rotate the image. Set the block parameters as follows:

   - **Rotation angle source** = `Input port`
   - **Sine value computation method** = `Trigonometric function`



Rotate

Rotates an image by an angle in radians. You can specify this angle using the block parameters dialog box or input port Angle.

Use the Output size parameter to determine the size of the output. If you select Expanded to fit rotated input image, the block outputs a matrix that contains all the rotated image values and zeros elsewhere. If you select Same as input image, the block outputs a matrix that contains the middle part of the rotated image and zeros elsewhere. As a result, the edges of the rotated image might be cropped.

When specifying the rotation angle using the Angle port, the maximum angle value should be greater than 0 but less than or equal to pi radians.

Parameters

Output size: Expanded to fit rotated input image

Rotation angle source: Specify via dialog

Angle (radians): pi/6

Sine value computation method: Table lookup

Background fill value: 0

Interpolation method: Bilinear

The Angle port appears on the block. You use this port to input a steadily increasing angle. Setting the **Output size** parameter to `Expanded to fit rotated input image` ensures that the block does not crop the output.

8 Use the Video Viewer1 block to display the rotating image. Accept the default parameters.

9 Use the Counter block to create a steadily increasing angle. Set the block parameters as follows:

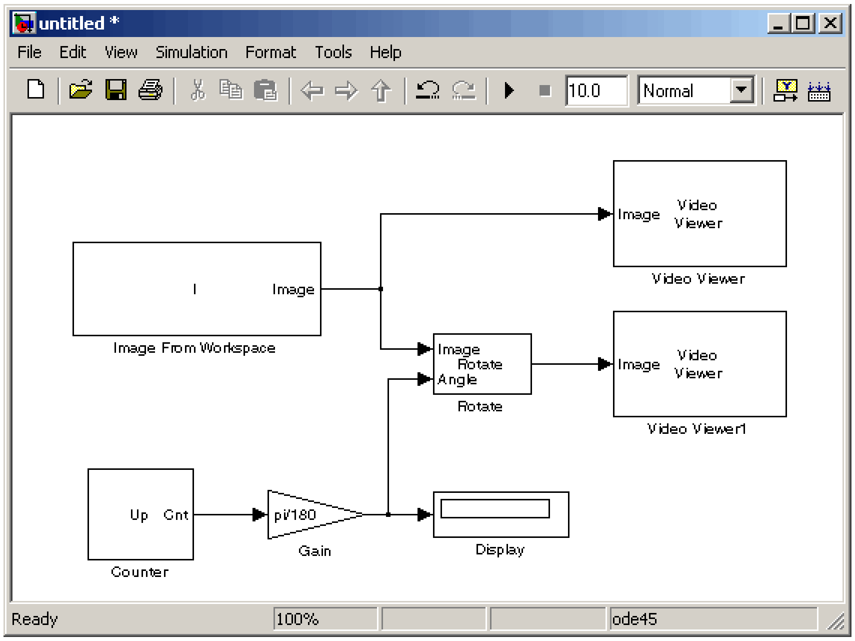
- **Count event** = `Free running`

- **Counter size** = `16 bits`
- **Output** = `Count`
- Clear the **Reset input** check box.
- **Sample time** = `1/30`

The Counter block counts upward until it reaches the maximum value that can be represented by 16 bits. Then, it starts again at zero. You can view its output value on the Display block while the simulation is running. The Counter block's **Count data type** parameter enables you to specify it's output data type.

10 Use the Gain block to convert the output of the Counter block from degrees to radians. Set the **Gain** parameter to `pi/180`.

11 Connect the blocks as shown in the following figure.

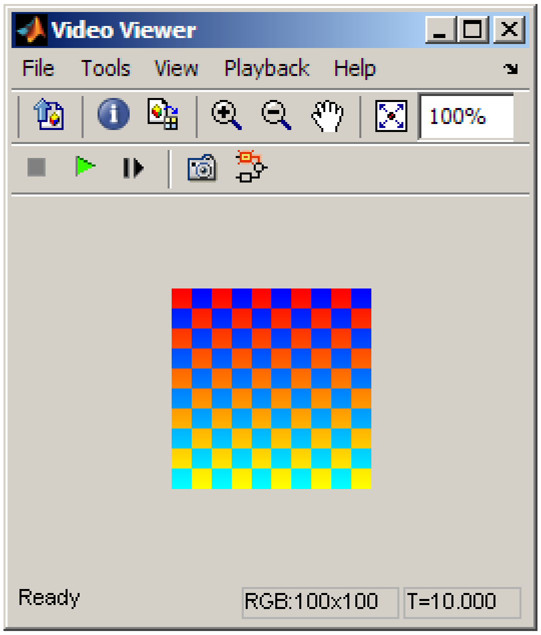


12 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

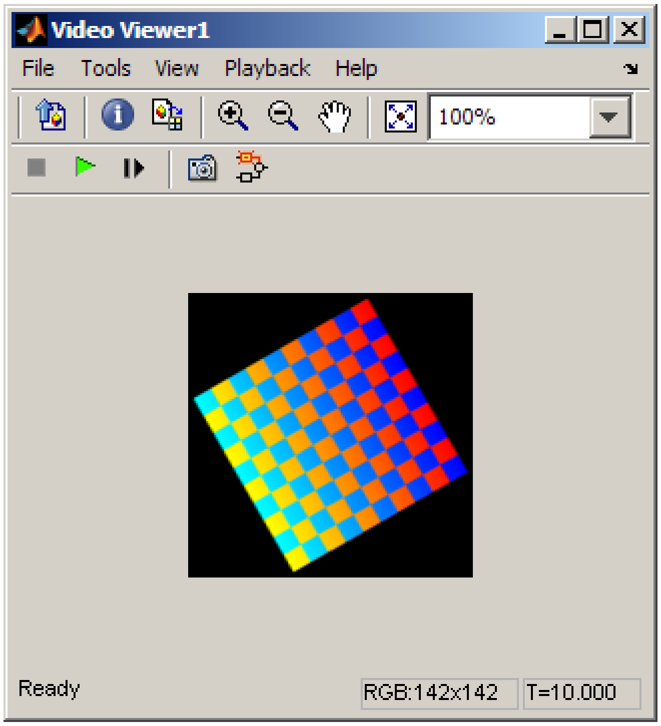
- **Solver** pane, **Stop time** = `inf`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

13 Run the model.

The original image appears in the Video Viewer window.



The rotating image appears in the Video Viewer1 window.



In this example, you used the Rotate block to continuously rotate your image. For more information about this block, see the Rotate block reference page in the *Computer Vision System Toolbox Reference*. For more information about other geometric transformation blocks, see the Resize and Shear block reference pages.

Note If you are on a Windows operating system, you can replace the Video Viewer block with the To Video Display block, which supports code generation.

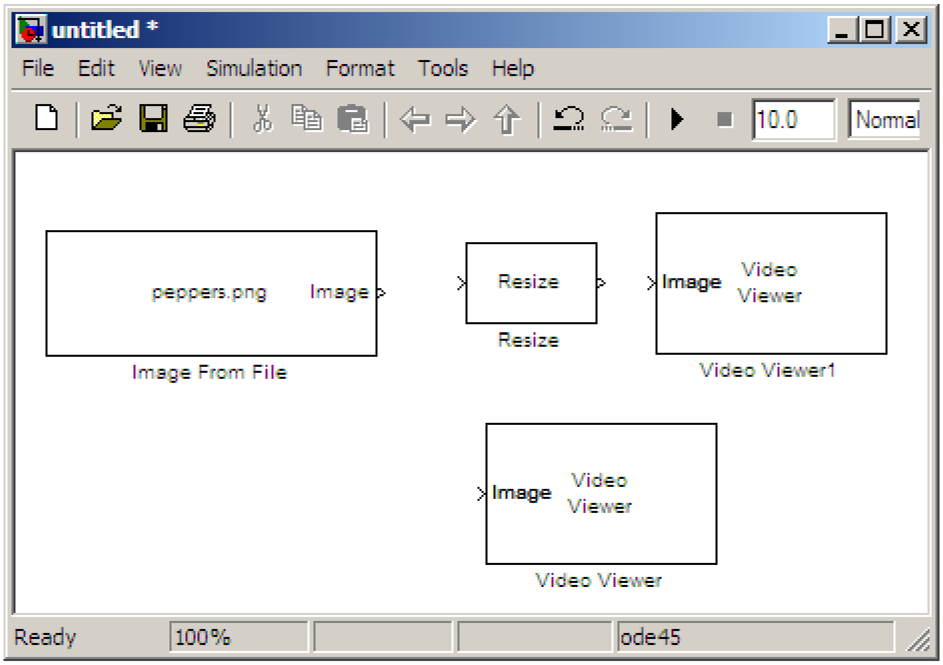
Resize an Image

You can use the Resize block to change the size of your image or video stream. In this example, you learn how to use the Resize block to reduce the size of an image:

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	**Library**	**Quantity**
Image From File	Computer Vision System Toolbox > Sources	1
Resize	Computer Vision System Toolbox > Geometric Transformations	1
Video Viewer	Computer Vision System Toolbox > Sinks	2

2 Position the blocks as shown in the following figure.

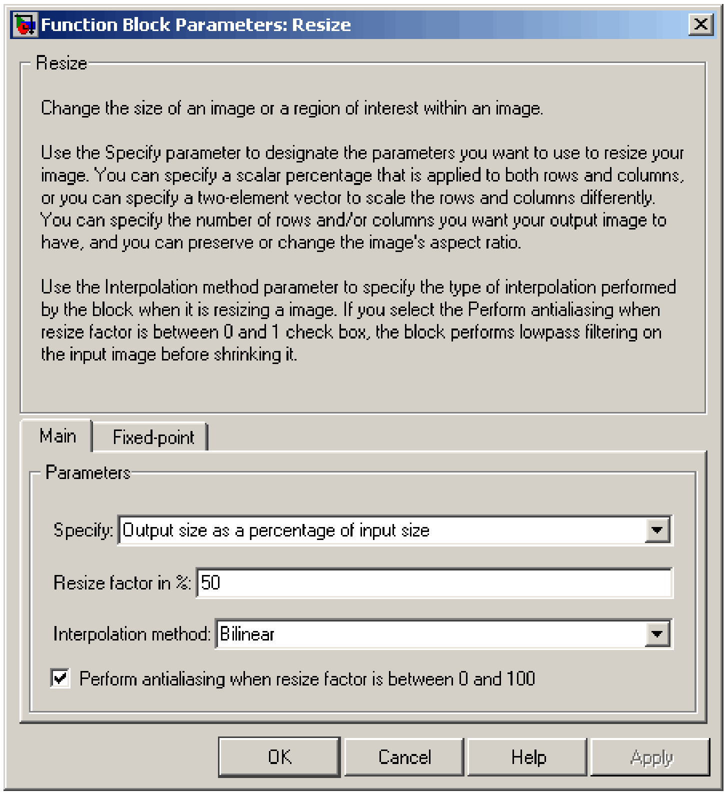


3 Use the Image From File block to import the intensity image. Set the **File name** parameter to `moon.tif`. The tif file is a 537-by-358 matrix of 8-bit unsigned integer values.

4 Use the Video Viewer block to display the original image. Accept the default parameters.

The Video Viewer block automatically displays the original image in the Video Viewer window when you run the model.

5 Use the Resize block to shrink the image. Set the **Resize factor in %** parameter to 50.



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 your image. You can specify a scalar percentage that is applied to both rows and columns, or you can specify a two-element vector to scale the rows and columns differently. You can specify the number of rows and/or columns you want your output image to have, and you can preserve or change the image's aspect ratio.

Use the Interpolation method parameter to specify the type of interpolation performed by the block when it is resizing a image. If you select the Perform antialiasing when resize factor is between 0 and 1 check box, the block performs lowpass filtering on the input image before shrinking it.

Parameters

Specify: Output size as a percentage of input size

Resize factor in %: 50

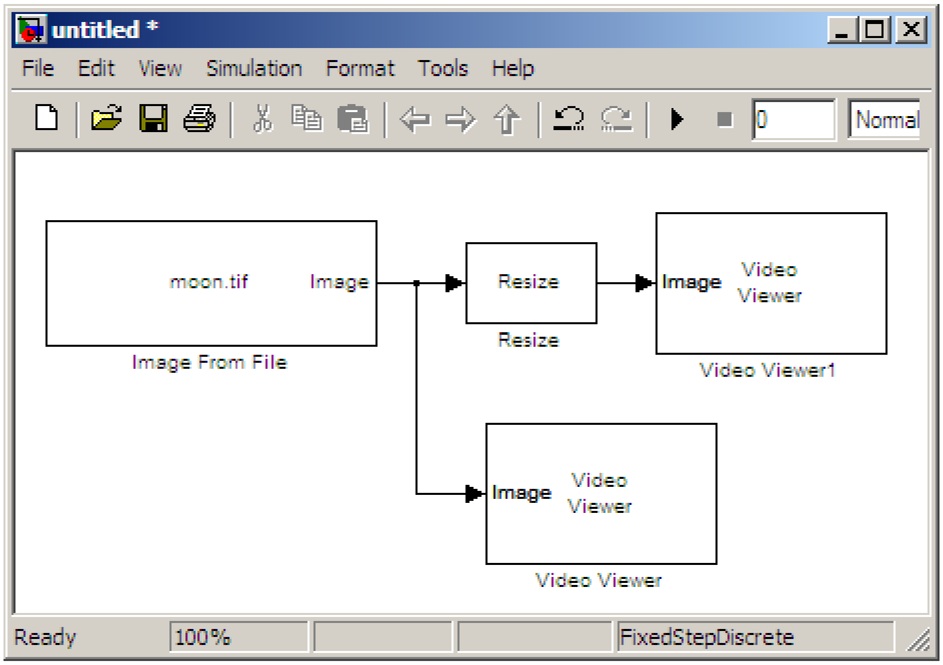
Interpolation method: Bilinear

[✓] Perform antialiasing when resize factor is between 0 and 100

The Resize block shrinks the image to half its original size.

6 Use the Video Viewer1 block to display the modified image. Accept the default parameters.

7 Connect the blocks as shown in the following figure.

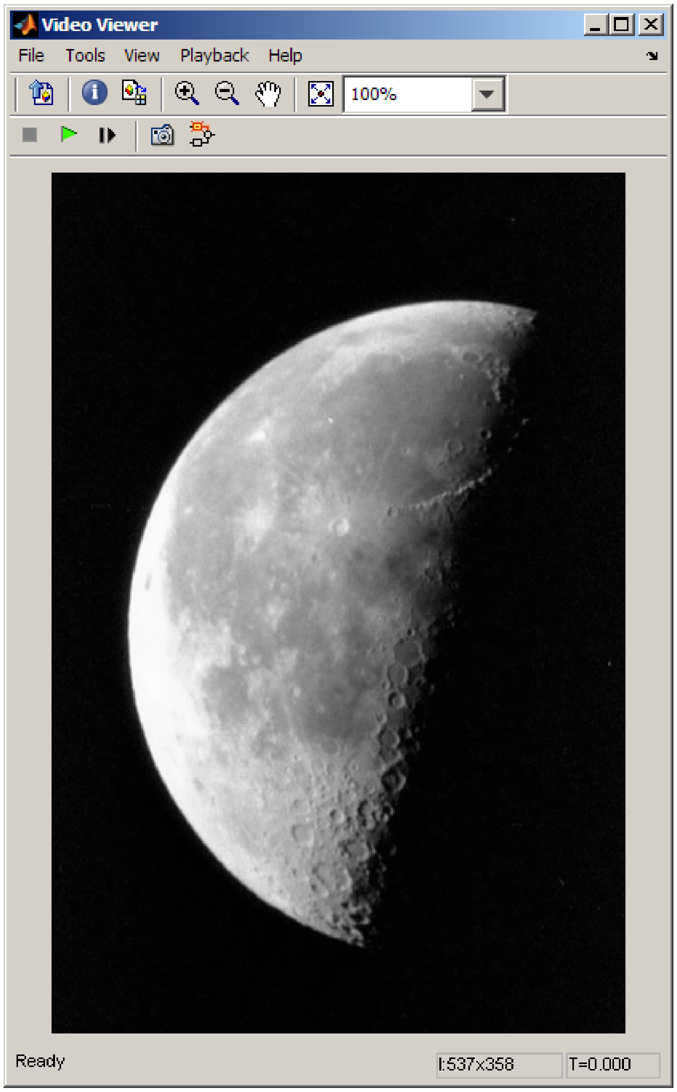


8 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

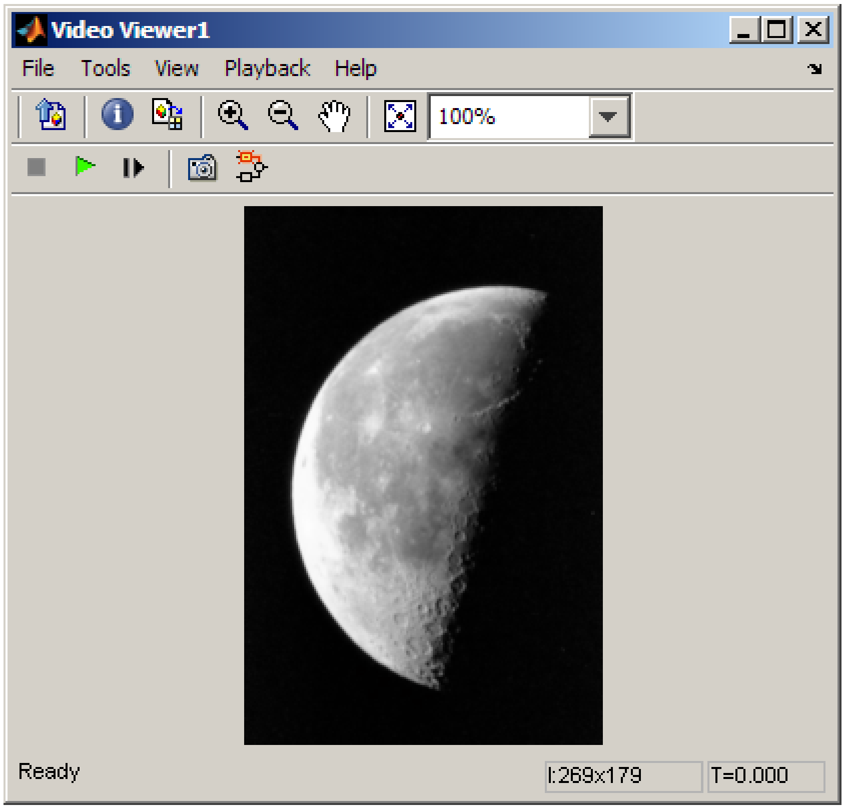
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

9 Run the model.

The original image appears in the Video Viewer window.



The reduced image appears in the Video Viewer1 window.



In this example, you used the Resize block to shrink an image. For more information about this block, see the Resize block reference page. For more information about other geometric transformation blocks, see the Rotate, Shear, and Translate block reference pages.

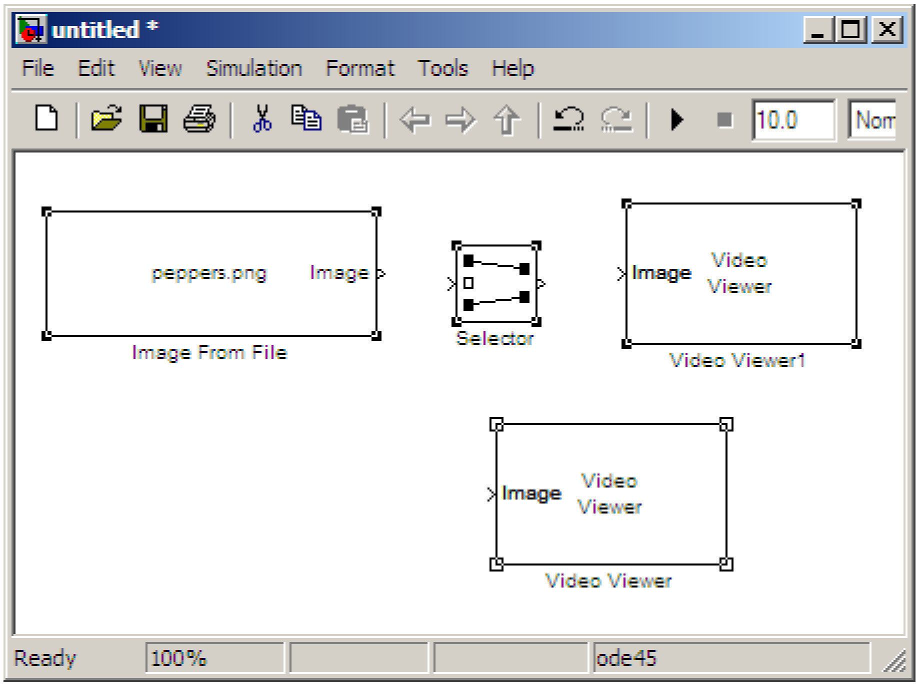
Crop an Image

You can use the Selector block to crop your image or video stream. In this example, you learn how to use the Selector block to trim an image down to a particular region of interest:

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Video Viewer	Computer Vision System Toolbox > Sinks	2
Selector	Simulink > Signal Routing	1

2 Position the blocks as shown in the following figure.



3 Use the Image From File block to import the intensity image. Set the **File name** parameter to `coins.png`. The image is a 246-by-300 matrix of 8-bit unsigned integer values.

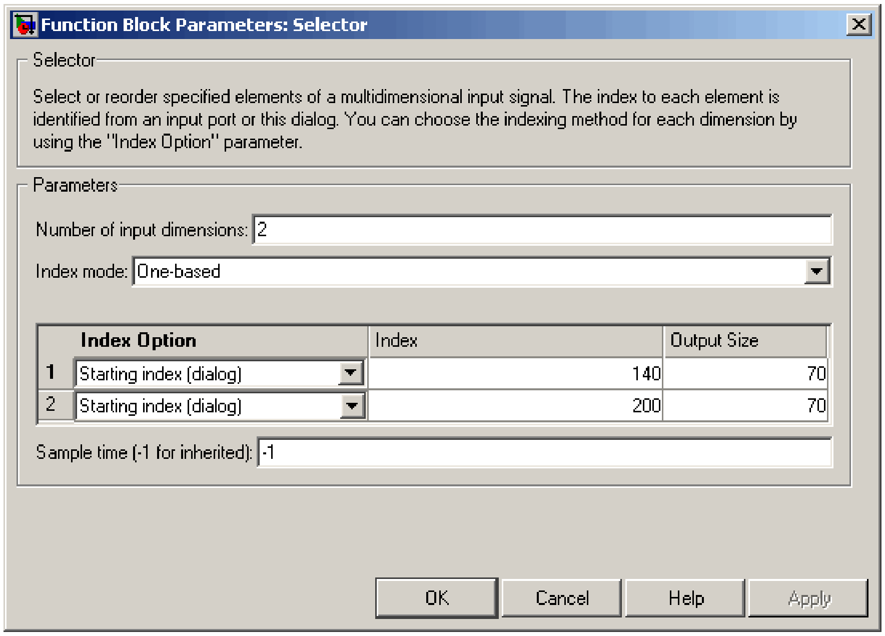
4 Use the Video Viewer block to display the original image. Accept the default parameters.

The Video Viewer block automatically displays the original image in the Video Viewer window when you run the model.

5 Use the Selector block to crop the image. Set the block parameters as follows:

- **Number of input dimensions** = 2
- **1**
  - **Index Option** = `Starting index (dialog)`
  - **Index** = 140

   - **Output Size** = `70`
- **2**
   - **Index Option** = `Starting index (dialog)`
   - **Index** = `200`
   - **Output Size** = `70`



Selector

Select or reorder specified elements of a multidimensional input signal. The index to each element is identified from an input port or this dialog. You can choose the indexing method for each dimension by using the "Index Option" parameter.

Parameters

Number of input dimensions: 2

Index mode: One-based

	Index Option	Index	Output Size
1	Starting index (dialog)	140	70
2	Starting index (dialog)	200	70

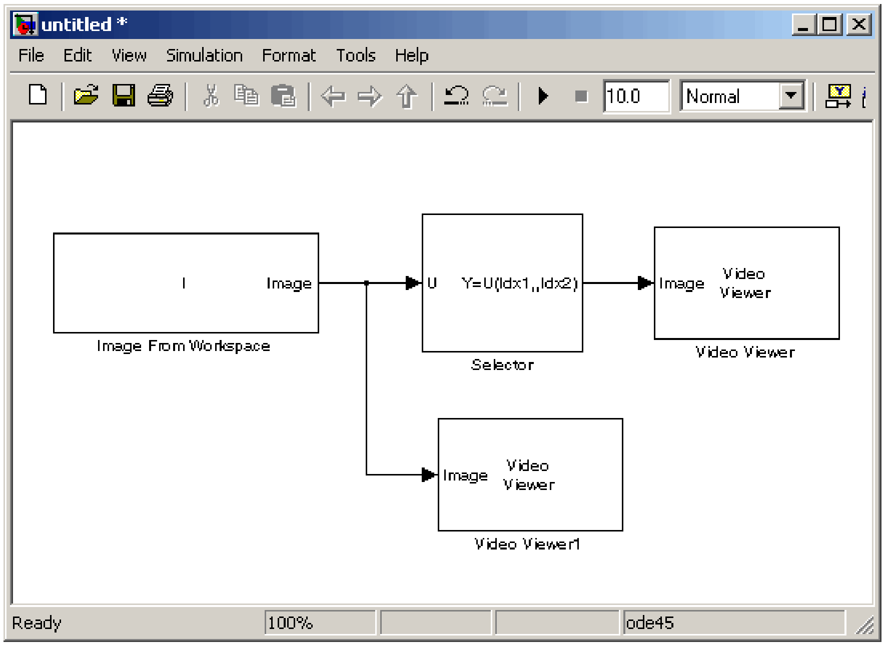
Sample time (-1 for inherited): -1

The Selector block starts at row 140 and column 200 of the image and outputs the next 70 rows and columns of the image.

6 Use the Video Viewer1 block to display the cropped image.

The Video Viewer1 block automatically displays the modified image in the Video Viewer window when you run the model.

7 Connect the blocks as shown in the following figure.

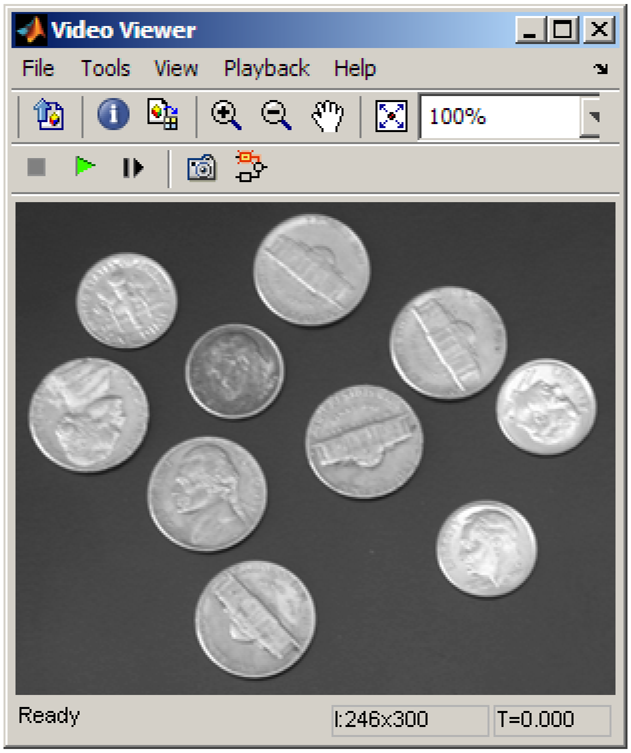


8 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

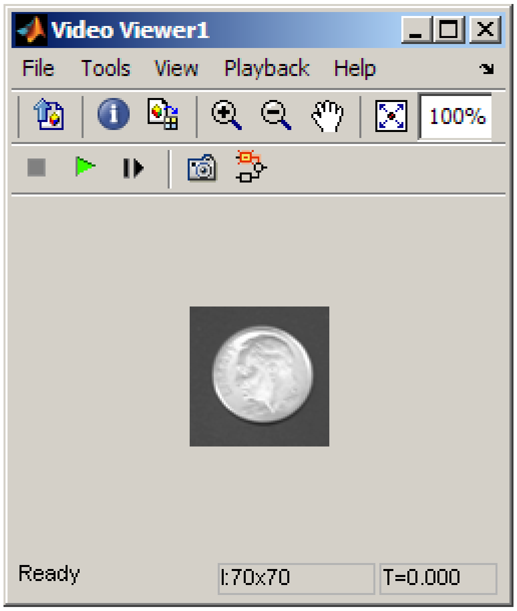
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

9 Run the model.

The original image appears in the Video Viewer window.



The cropped image appears in the Video Viewer window. The following image is shown at its true size.



In this example, you used the Selector block to crop an image. For more information about the Selector block, see the Simulink documentation. For information about the `imcrop` function, see the Image Processing Toolbox documentation.

Interpolation Methods

In this section...

Nearest Neighbor Interpolation

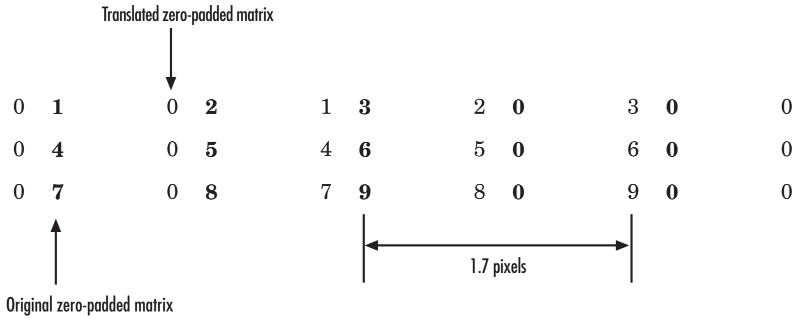
For nearest neighbor interpolation, the block uses the value of nearby translated pixel values for the output pixel values.

For example, suppose this matrix,

```
1  2  3
4  5  6
7  8  9
```

represents your input image. You want to translate this image 1.7 pixels in the positive horizontal direction using nearest neighbor interpolation. The Translate block's nearest neighbor interpolation algorithm is illustrated by the following steps:

1 Zero pad the input matrix and translate it by 1.7 pixels to the right.



2. Create the output matrix by replacing each input pixel value with the translated value nearest to it. The result is the following matrix:

```
0 0 1 2 3
0 0 4 5 6
0 0 7 8 9
```

Note You wanted to translate the image by 1.7 pixels, but this method translated the image by 2 pixels. Nearest neighbor interpolation is computationally efficient but not as accurate as bilinear or bicubic interpolation.

Bilinear Interpolation

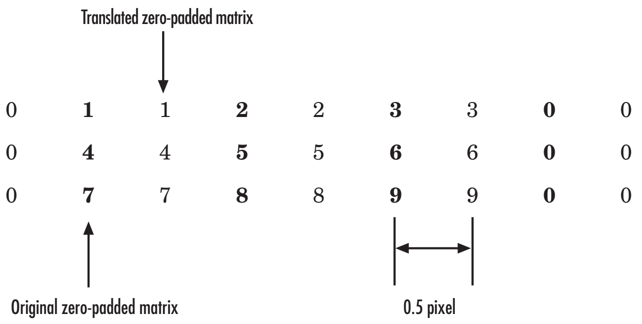
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

For example, suppose this matrix,

```
1 2 3
4 5 6
7 8 9
```

represents your input image. You want to translate this image 0.5 pixel in the positive horizontal direction using bilinear interpolation. The Translate block's bilinear interpolation algorithm is illustrated by the following steps:

1. Zero pad the input matrix and translate it by 0.5 pixel to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the translated values on either side. The result is the following matrix where the output matrix has one more column than the input matrix:

```
0.5  1.5  2.5  1.5
 2   4.5  5.5   3
3.5  7.5  8.5  4.5
```

Bicubic Interpolation

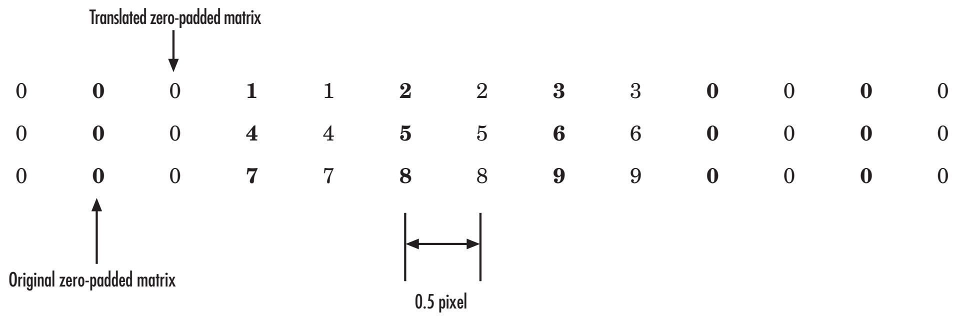
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

For example, suppose this matrix,

```
1  2  3
4  5  6
7  8  9
```

represents your input image. You want to translate this image 0.5 pixel in the positive horizontal direction using bicubic interpolation. The Translate block's bicubic interpolation algorithm is illustrated by the following steps:

1 Zero pad the input matrix and translate it by 0.5 pixel to the right.

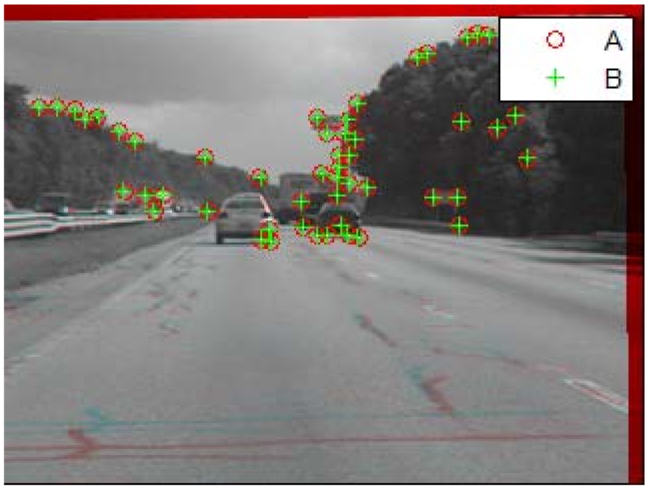


2 Create the output matrix by replacing each input pixel value with the weighted average of the two translated values on either side. The result is the following matrix where the output matrix has one more column than the input matrix:

```
0.375   1.5      3     1.625
1.875   4.875   6.375  3.125
3.375   8.25    9.75   4.625
```

Automatically Determine Geometric Transform for Image Registration

Stabilizing a video that was captured from a jittery or moving platform is an important application in computer vision. One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This procedure, however, must be bootstrapped with knowledge of where such a salient feature lies in the first video frame. In this example, we explore a method of video stabilization that works without any such a priori knowledge. It instead automatically searches for the "background plane" in a video sequence, and uses its observed distortion to correct for camera motion.



This stabilization algorithm involves two steps. First, we determine the affine image transformations between all neighboring frames of a video sequence using a Random Sampling and Consensus (RANSAC) [1] procedure applied to point correspondences between two images. Second, we warp the video frames to achieve a stabilized video. We use System objects in the Computer Vision System Toolbox™, both for the algorithm and for display.

You can launch this example, Video Stabilization Using Point Feature Matching directly, by typing `videostabilize_pm` on the MATLAB command line.

Filters, Transforms, and Enhancements

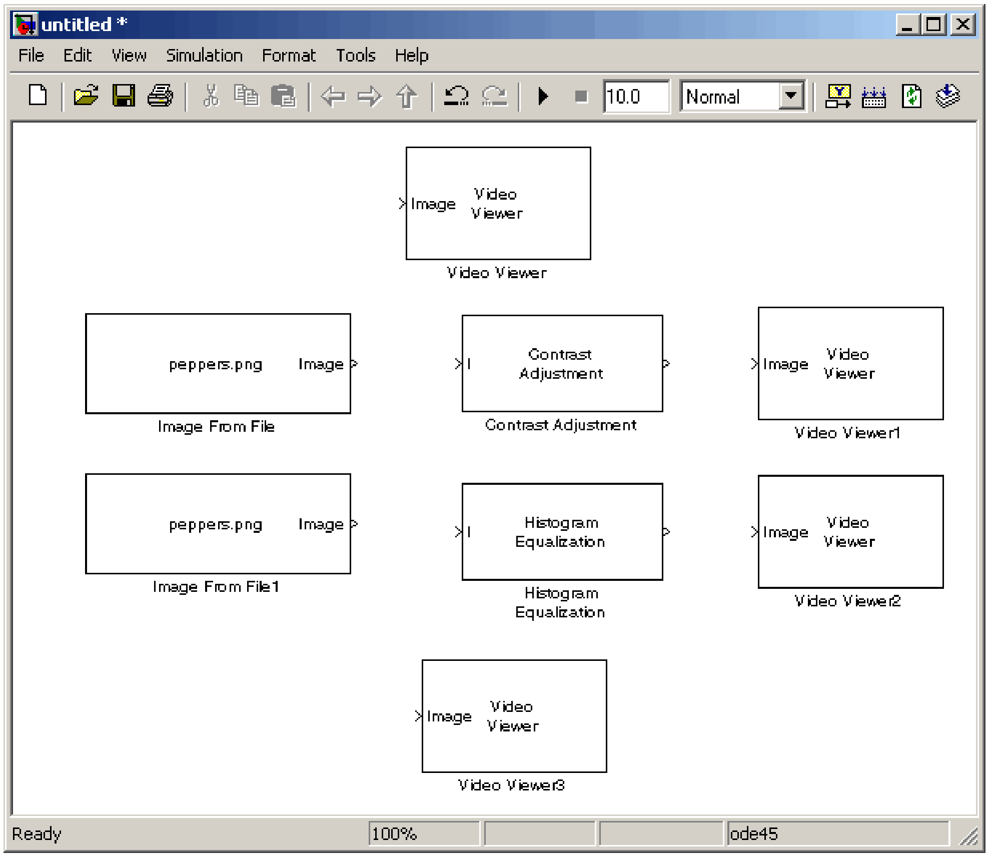
Adjust the Contrast of Intensity Images

This example shows you how to modify the contrast in two intensity images using the Contrast Adjustment and Histogram Equalization blocks.

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	2
Contrast Adjustment	Computer Vision System Toolbox > Analysis & Enhancement	1
Histogram Equalization	Computer Vision System Toolbox > Analysis & Enhancement	1
Video Viewer	Computer Vision System Toolbox > Sinks	4

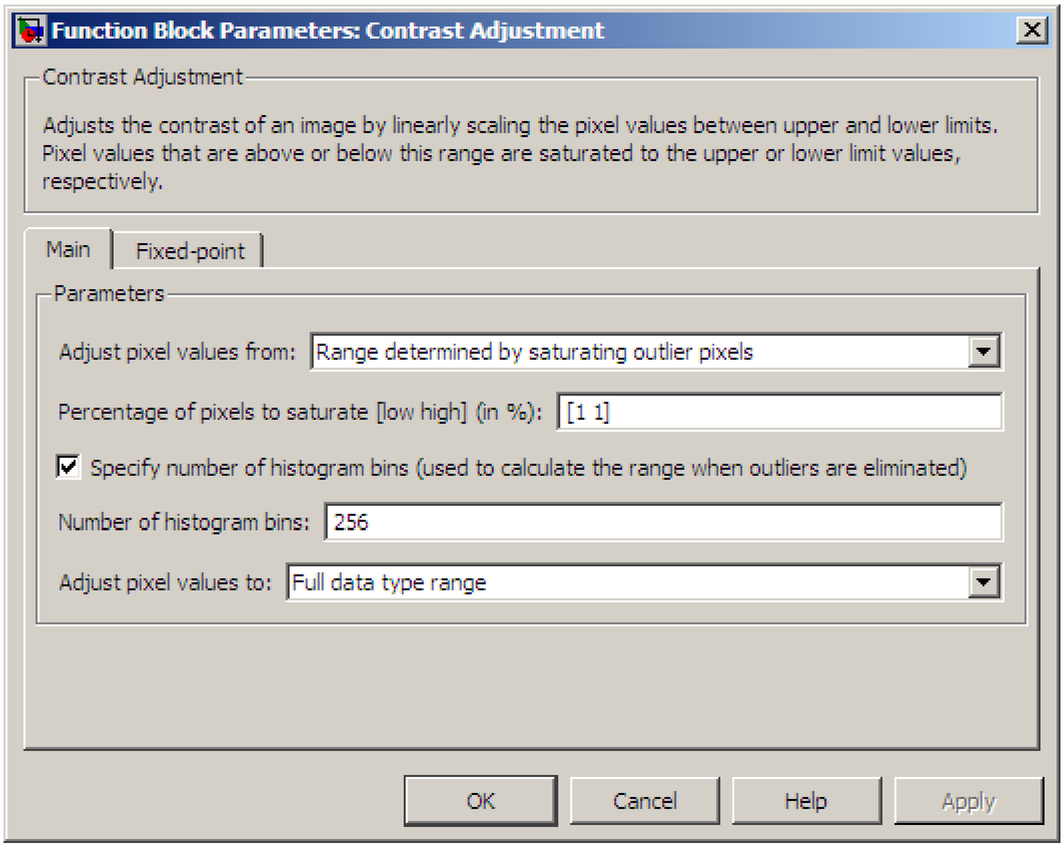
2 Place the blocks so that your model resembles the following figure.



3 Use the Image From File block to import the first image into the Simulink model. Set the **File name** parameter to `pout.tif`.

4 Use the Image From File1 block to import the second image into the Simulink model. Set the **File name** parameter to `tire.tif`.

5 Use the Contrast Adjustment block to modify the contrast in `pout.tif`. Set the **Adjust pixel values from** parameter to `Range determined by saturating outlier pixels`, as shown in the following figure.



Function Block Parameters: Contrast Adjustment

Contrast Adjustment

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 values, respectively.

Parameters

Adjust pixel values from: Range determined by saturating outlier pixels

Percentage of pixels to saturate [low high] (in %): [1 1]

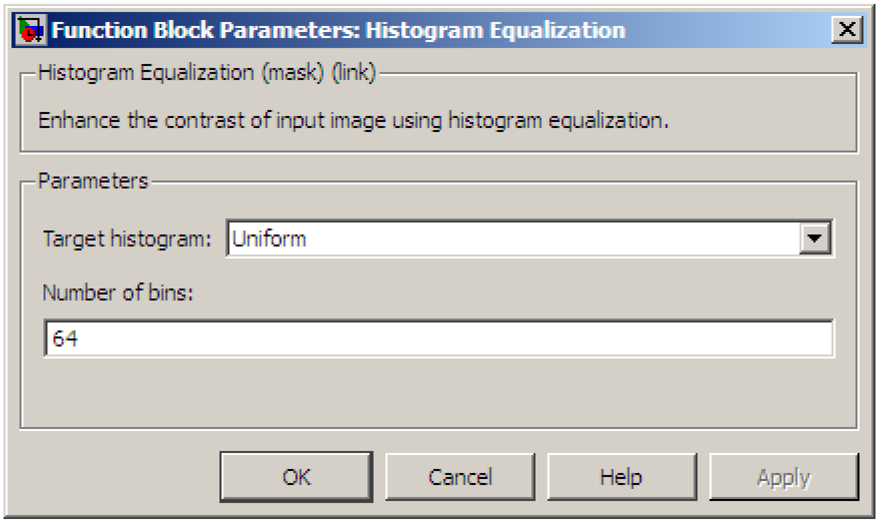
[✓] Specify number of histogram bins (used to calculate the range when outliers are eliminated)

Number of histogram bins: 256

Adjust pixel values to: Full data type range

This block adjusts the contrast of the image by linearly scaling the pixel values between user-specified upper and lower limits.

6 Use the Histogram Equalization block to modify the contrast in `tire.tif`. Accept the default parameters.


Function Block Parameters: Histogram Equalization

Histogram Equalization (mask) (link)

Enhance the contrast of input image using histogram equalization.

Parameters

Target histogram: Uniform

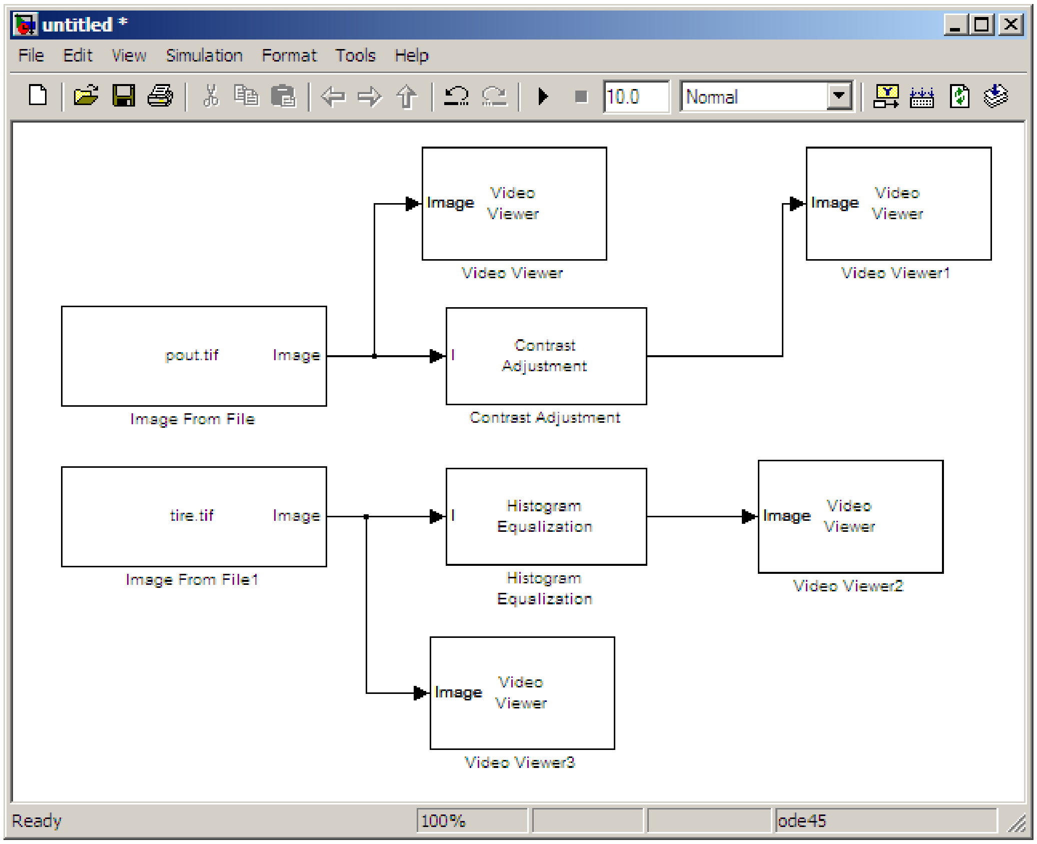
Number of bins:

64

This 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.

7 Use the Video Viewer blocks to view the original and modified images. Accept the default parameters.

8 Connect the blocks as shown in the following figure.

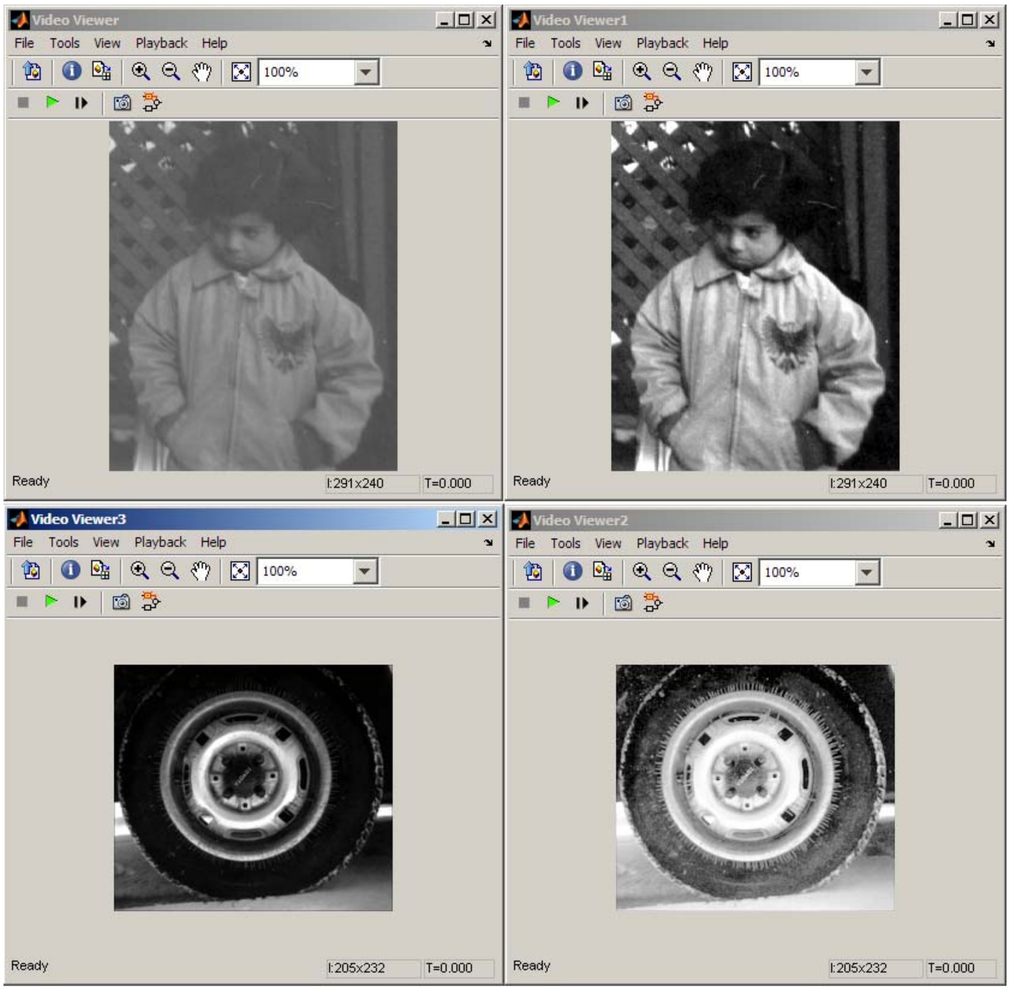


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

10 Run the model.

The results appear in the Video Viewer windows.



In this example, you used the Contrast Adjustment block to linearly scale the pixel values in `pout.tif` between new upper and lower limits. You used the Histogram Equalization block to transform the values in `tire.tif` so that the histogram of the output image approximately matches a uniform histogram. For more information, see the Contrast Adjustment and Histogram Equalization reference pages.

Adjust the Contrast of Color Images

This example shows you how to modify the contrast in color images using the Histogram Equalization block.

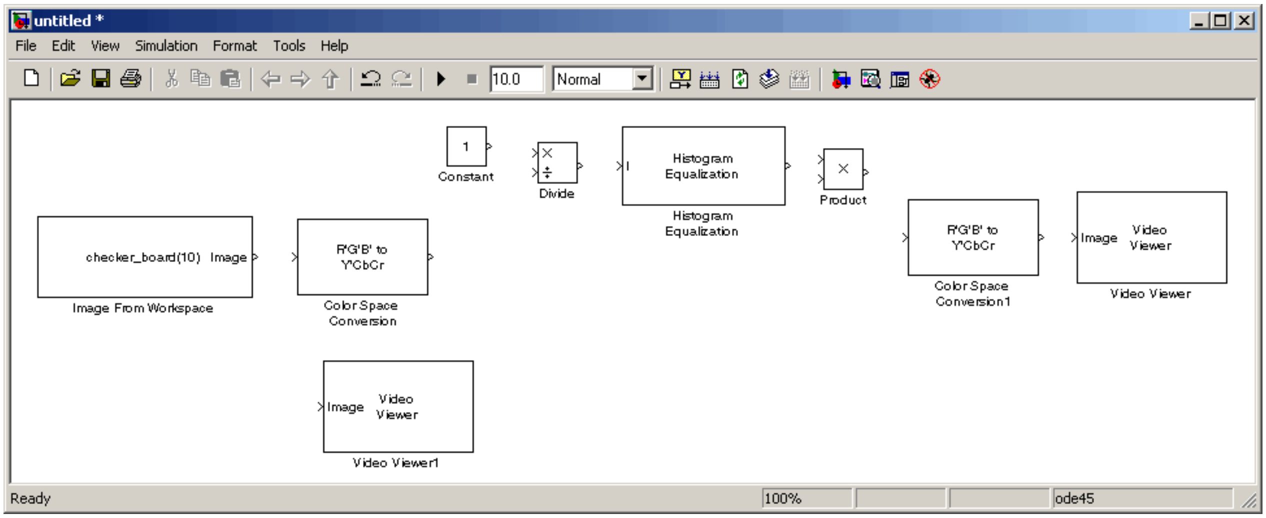
1 Use the following code to read in an indexed RGB image, `shadow.tif`, and convert it to an RGB image.

```
[X map] = imread('shadow.tif');
shadow = ind2rgb(X,map);
```

2 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From Workspace	Computer Vision System Toolbox > Sources	1
Color Space Conversion	Computer Vision System Toolbox > Conversions	2
Histogram Equalization	Computer Vision System Toolbox > Analysis & Enhancement	1
Video Viewer	Computer Vision System Toolbox > Sinks	2
Constant	Simulink > Sources	1
Divide	Simulink > Math Operations	1
Product	Simulink > Math Operations	1

3 Place the blocks so that your model resembles the following figure.



4 Use the Image From Workspace block to import the RGB image from the MATLAB workspace into the Simulink model. Set the block parameters as follows:

- **Value** = `shadow`
- **Image signal** = `Separate color signals`

5 Use the Color Space Conversion block to separate the luma information from the color information. Set the block parameters as follows:

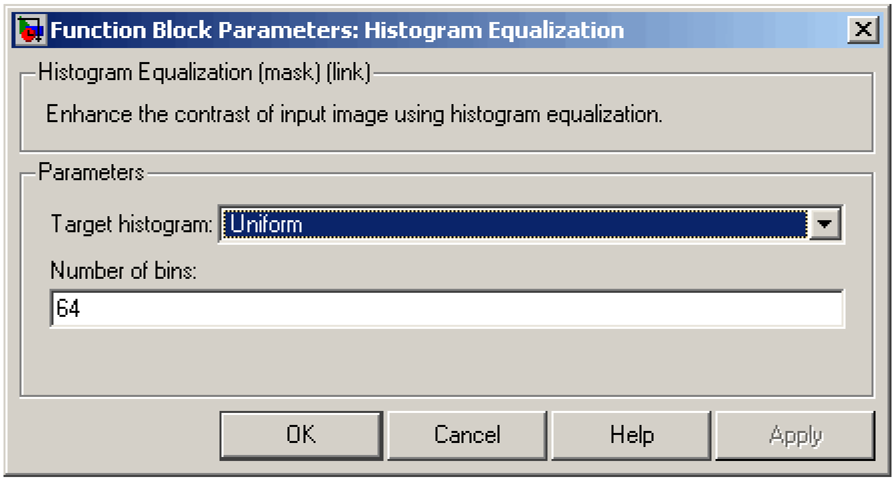
- **Conversion** = `sR'G'B' to L*a*b*`
- **Image signal** = `Separate color signals`

Because the range of the L* values is between `0` and `100`, you must normalize them to be between zero and one before you pass them to the Histogram Equalization block, which expects floating point input in this range.

6 Use the Constant block to define a normalization factor. Set the **Constant value** parameter to `100`.

7 Use the Divide block to normalize the L* values to be between `0` and `1`. Accept the default parameters.

8 Use the Histogram Equalization block to modify the contrast in the image. Accept the default parameters.



Histogram Equalization (mask) (link)

Enhance the contrast of input image using histogram equalization.

Parameters

Target histogram: Uniform

Number of bins:

64

This block enhances the contrast of images by transforming the luma values in the color image so that the histogram of the output image approximately matches a specified histogram.

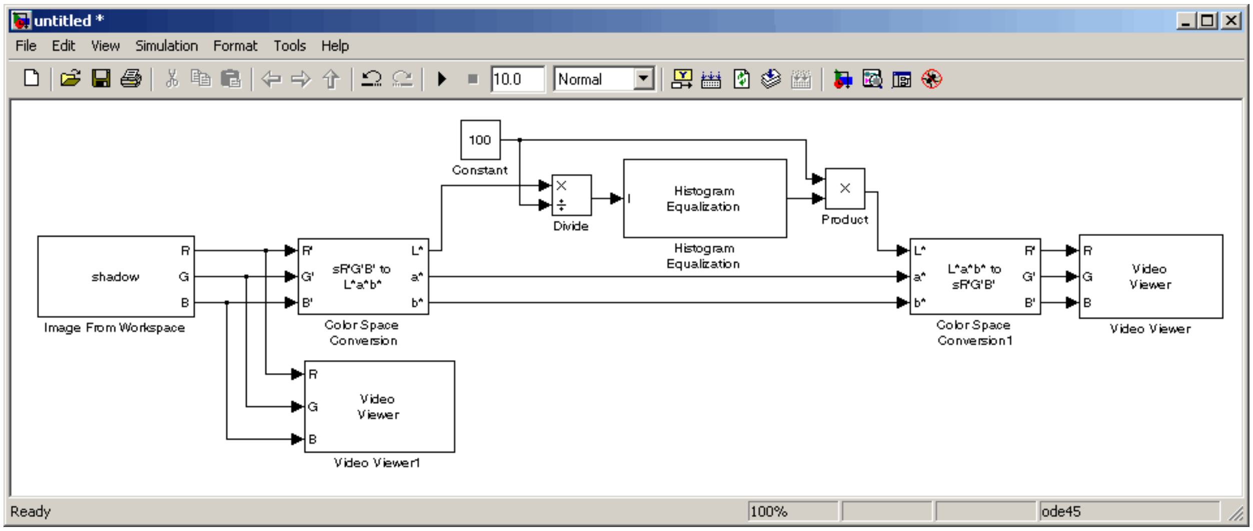
9 Use the Product block to scale the values back to be between the `0` to `100` range. Accept the default parameters.

10 Use the Color Space Conversion1 block to convert the values back to the sR'G'B' color space. Set the block parameters as follows:

- **Conversion** = `L*a*b* to sR'G'B'`
- **Image signal** = `Separate color signals`

11 Use the Video Viewer blocks to view the original and modified images. For each block, set the **Image signal** parameter to `Separate color signals` from the file menu.

12 Connect the blocks as shown in the following figure.

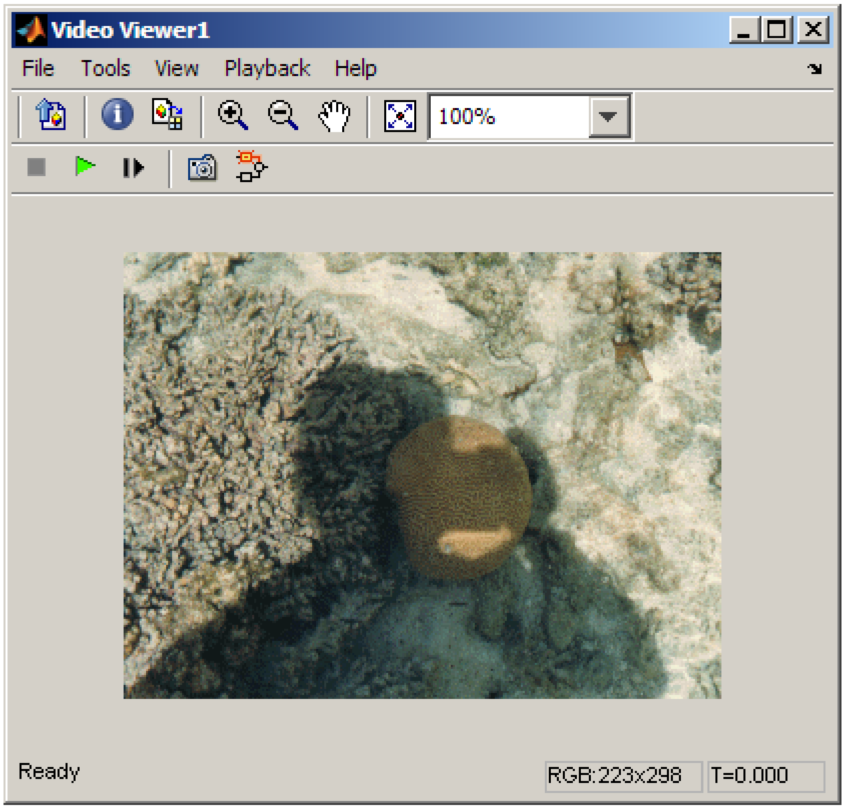


13 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

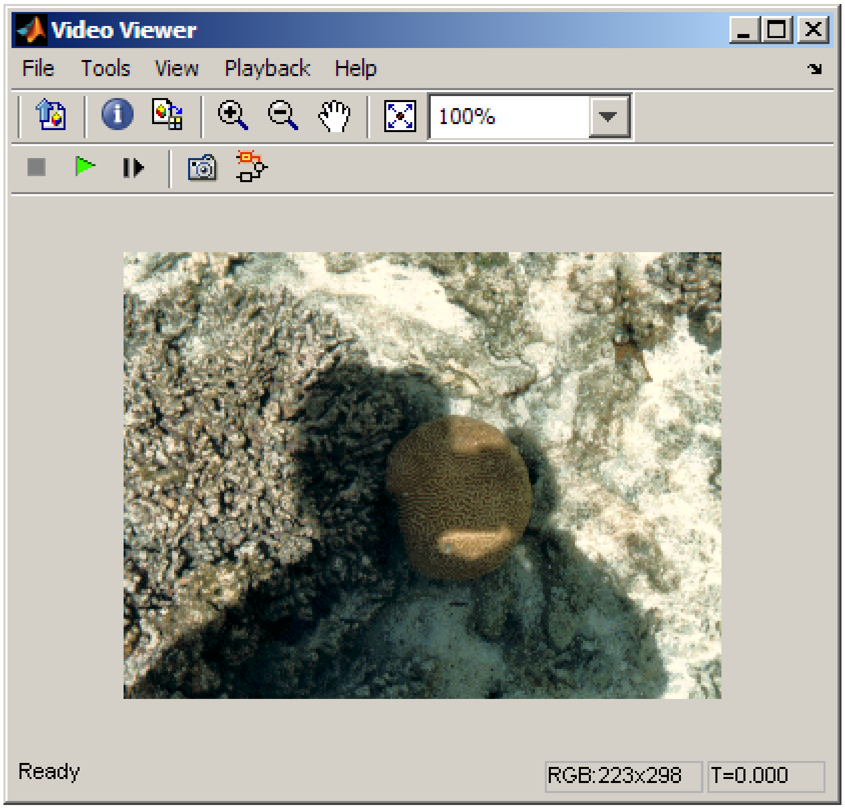
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

14 Run the model.

As shown in the following figure, the model displays the original image in the Video Viewer1 window.



As the next figure shows, the model displays the enhanced contrast image in the Video Viewer window.



In this example, you used the Histogram Equalization block to transform the values in a color image so that the histogram of the output image approximately matches a uniform histogram. For more information, see the Histogram Equalization reference page.

Remove Periodic Noise from a Video

Periodic noise can be introduced into a video stream during acquisition or transmission due to electrical or electromechanical interference. In this example, you remove periodic noise from an intensity video using the 2-D FIR Filter block. You can use this technique to remove noise from other images or video streams, but you might need to modify the filter coefficients to account for the noise frequency content present in your signal:

1 Create a Simulink model, and add the blocks shown in the following table.

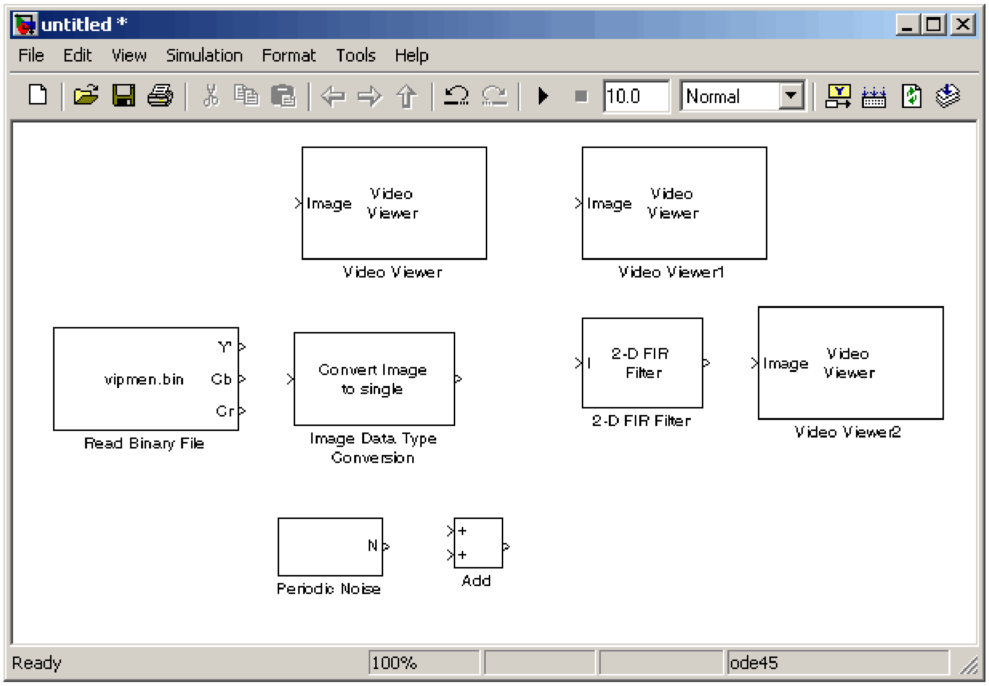
Block	Library	Quantity
Read Binary File	Computer Vision System Toolbox > Sources	1
Image Data Type Conversion	Computer Vision System Toolbox > Conversions	1
2-D FIR Filter	Computer Vision System Toolbox > Filtering	1
Video Viewer	Computer Vision System Toolbox > Sinks	3
Add	Simulink > Math Operations	1

2 Open the Periodic noise reduction demo by typing `vipstripes` at the MATLAB command prompt.

3 Click-and-drag the Periodic Noise block into your model.

The block outputs a sinusoid with a normalized frequency that ranges between 0.61π and 0.69π radians per sample and a phase that varies between zero and three radians. You are using this sinusoid to represent periodic noise.

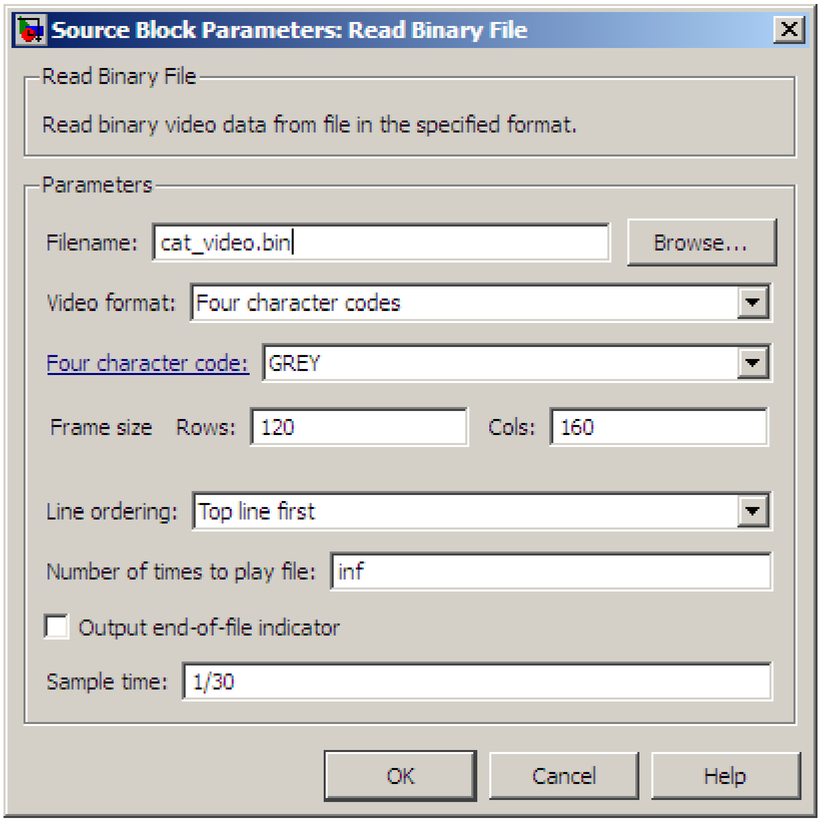
4 Place the blocks so that your model resembles the following figure. The unconnected ports disappear when you set block parameters.



You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

5 Use the Read Binary File block to import a binary file into the model. Set the block parameters as follows:

- **File name** = `cat_video.bin`
- **Four character code** = `GREY`
- **Number of times to play file** = `inf`
- **Sample time** = `1/30`


Source Block Parameters: Read Binary File

Read Binary File

Read binary video data from file in the specified format.

Parameters

Filename: cat_video.bin

Video format: Four character codes

Four character code: GREY

Frame size Rows: 120 Cols: 160

Line ordering: Top line first

Number of times to play file: inf

☐ Output end-of-file indicator

Sample time: 1/30

6 Use the Image Data Type Conversion block to convert the data type of the video to single-precision floating point. Accept the default parameter.

7 Use the Video Viewer block to view the original video. Accept the default parameters.

8 Use the Add block to add the noise video to the original video. Accept the default parameters.

9 Use the Video Viewer1 block to view the noisy video. Accept the default parameters.

10 Define the filter coefficients in the MATLAB workspace. Type the following code at the MATLAB command prompt:

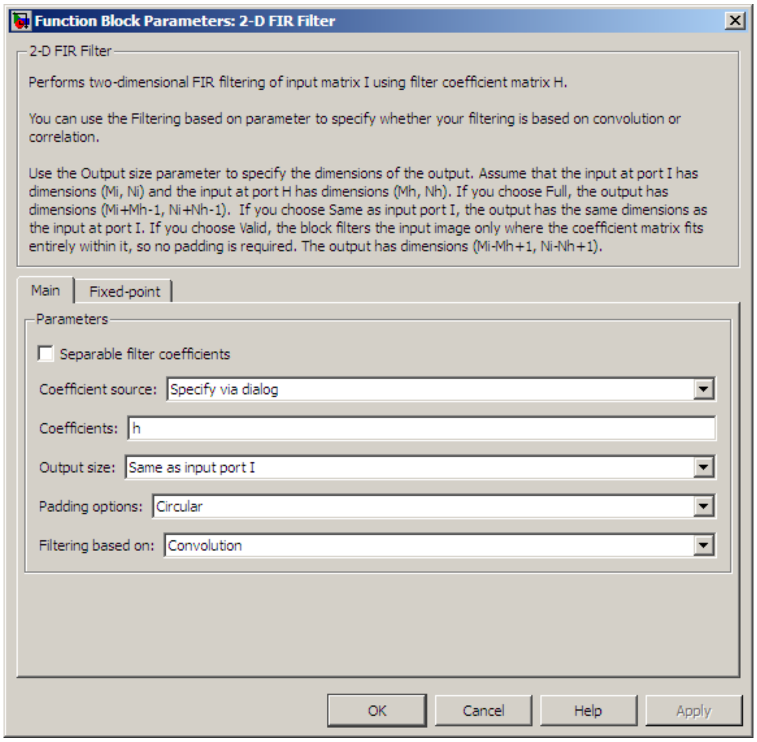
```
vipdh_stripes
```

The variable `h`, as well as several others, are loaded into the MATLAB workspace. The variable `h` represents the coefficients of the band reject filter capable of removing normalized frequencies between 0.61π and 0.69π radians per sample. The coefficients were created using the Filter Design and Analysis Tool (FDATool) and the `ftrans2` function.

11 Use the 2-D FIR Filter block to model a band-reject filter capable of removing the periodic noise from the video. Set the block parameters as follows:

- **Coefficients** = `h`
- **Output size** = `Same as input port I`
- **Padding options** = `Circular`

Choose a type of padding that minimizes the effect of the pixels outside the image on the processing of the image. In this example, circular padding produces the best results because it is most effective at replicating the sinusoidal noise outside the image.


Function Block Parameters: 2-D FIR Filter

2-D FIR Filter

Performs two-dimensional FIR filtering of input matrix I using filter coefficient matrix H.

You can use the Filtering based on parameter to specify whether your filtering is based on convolution or correlation.

Use the Output size parameter to specify the dimensions of the output. Assume that the input at port I has dimensions (Mi, Ni) and the input at port H has dimensions (Mh, Nh). If you choose Full, the output has dimensions (Mi+Mh-1, Ni+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, 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).

Main | Fixed-point

Parameters

☐ Separable filter coefficients

Coefficient source: Specify via dialog

Coefficients: h

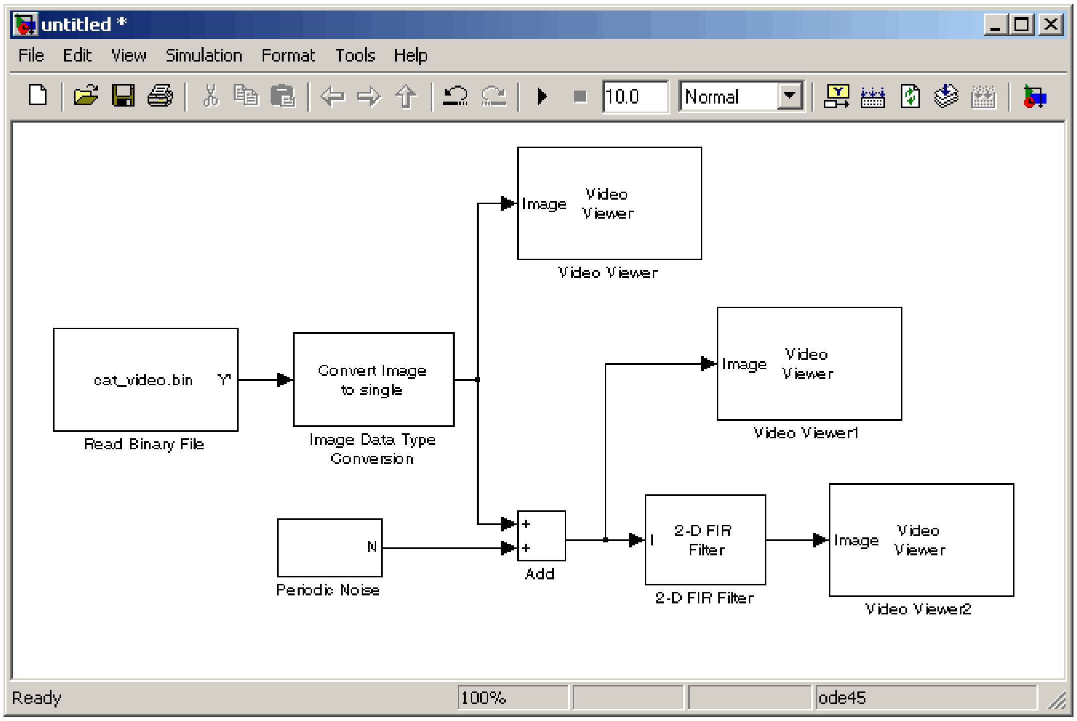
Output size: Same as input port I

Padding options: Circular

Filtering based on: Convolution

12 Use the Video Viewer2 block to view the approximation of the original video. Accept the default parameters.

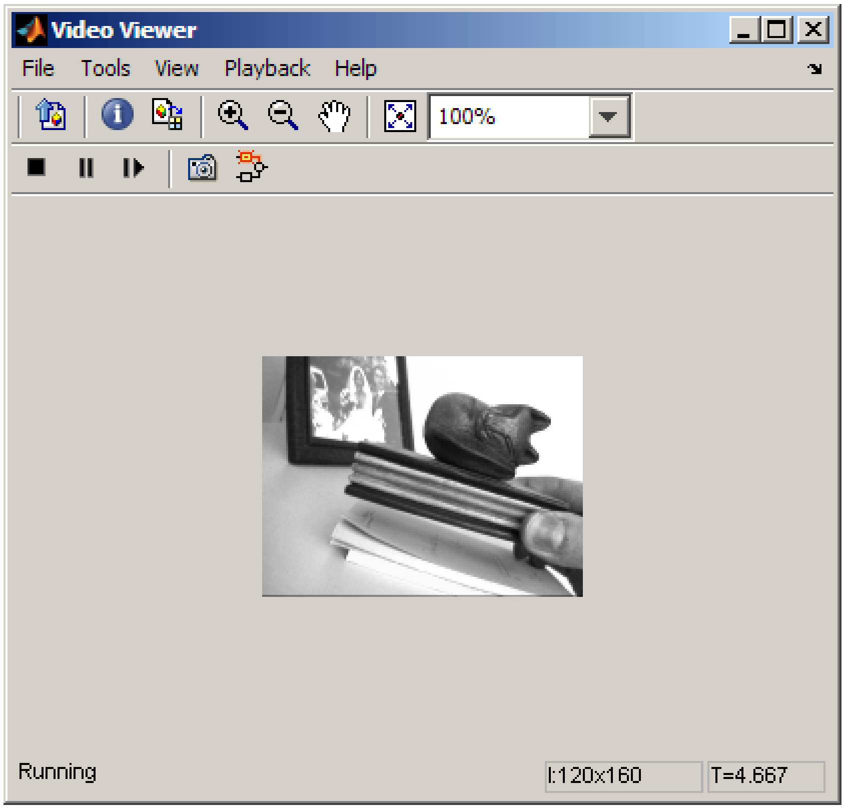
13 Connect the block as shown in the following figure.



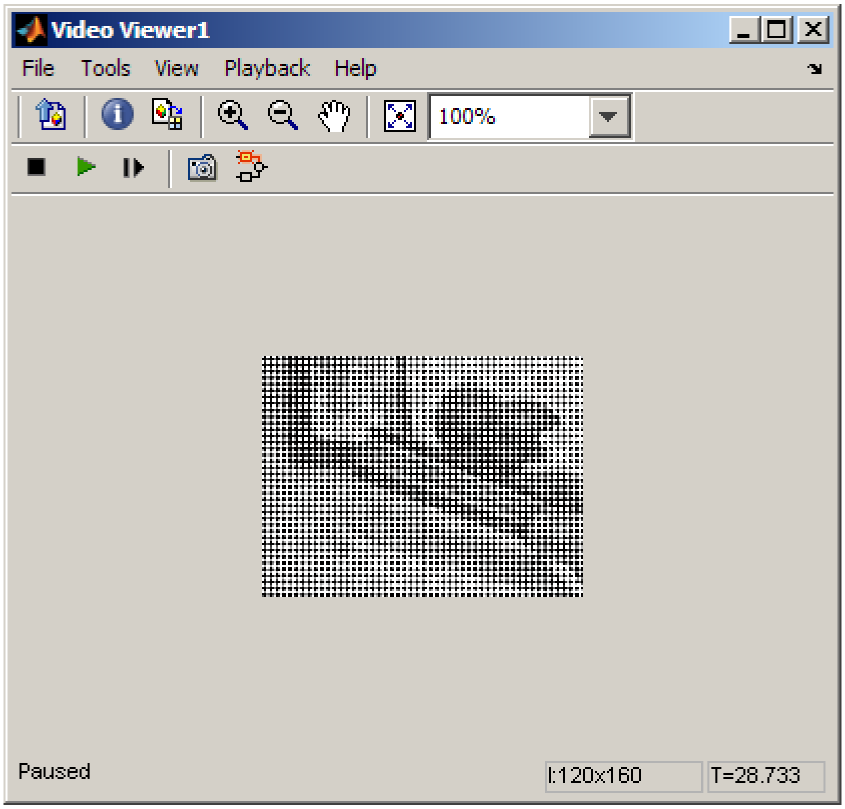
14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `inf`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

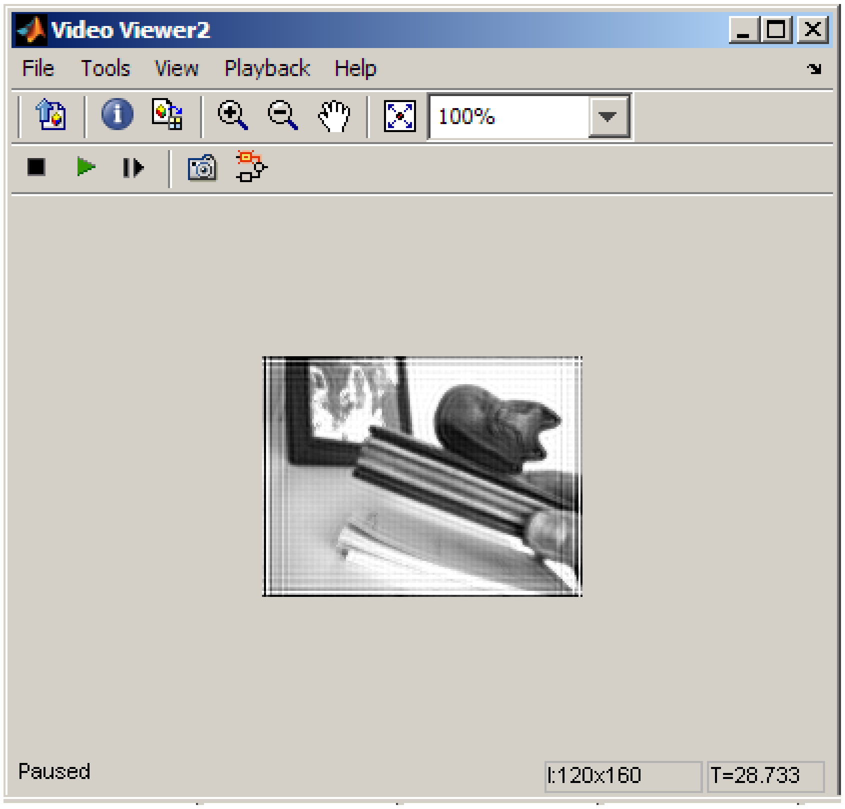
15 Run the model.



The noisy video appears in the Video Viewer1 window. The following video is shown at its true size.



The approximation of the original video appears in the Video Viewer2 window, and the artifacts of the processing appear near the edges of the video. The following video is shown at its true size.



You have used the Read Binary File block to import a binary video into your model, the 2-D FIR Filter to remove periodic noise from this video, and the Video Viewer block to display the results. For more information about these blocks, see the Read Binary File, 2-D FIR Filter, and Video Viewer block reference pages. For more information about the Filter Design and Analysis Tool (FDATool), see the Signal Processing Toolbox documentation. For information about the `ftrans2` function, see the Image Processing Toolbox documentation.

Remove Salt and Pepper Noise from Images

Median filtering is a common image enhancement technique for removing salt and pepper noise. Because this filtering is less sensitive than linear techniques to extreme changes in pixel values, it can remove salt and pepper noise without significantly reducing the sharpness of an image. In this topic, you use the Median Filter block to remove salt and pepper noise from an intensity image:

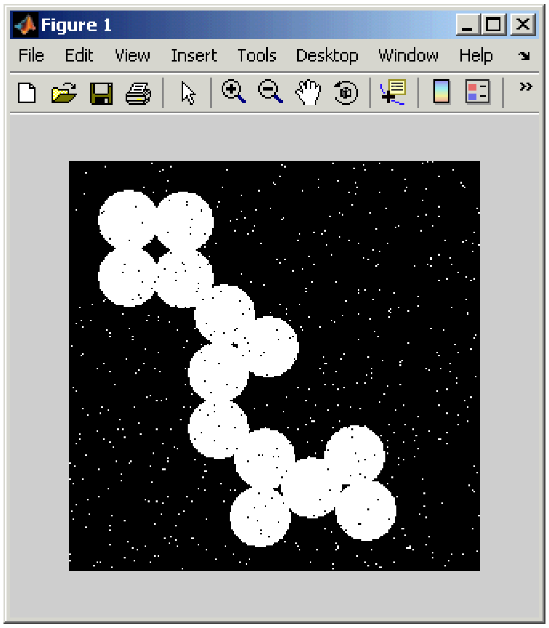
1 Define an intensity image in the MATLAB workspace and add noise to it by typing the following at the MATLAB command prompt:

   ```
   I= double(imread('circles.png'));
   I= imnoise(I,'salt & pepper',0.02);
   ```

   `I` is a 256-by-256 matrix of 8-bit unsigned integer values.

2 To view the image this matrix represents, at the MATLAB command prompt, type

   ```
   imshow(I)
   ```

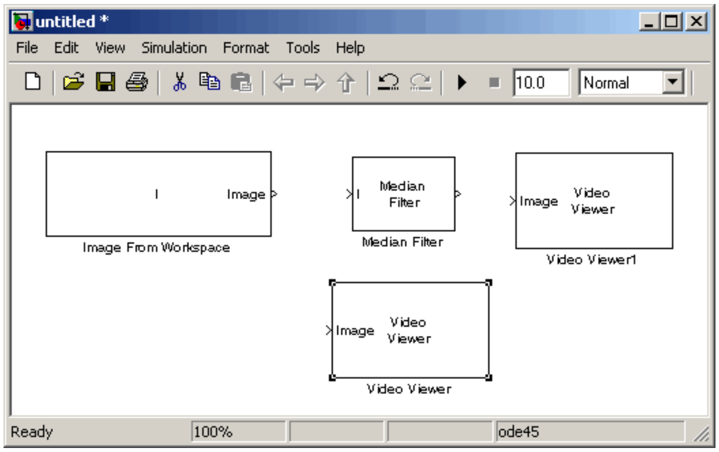


The intensity image contains noise that you want your model to eliminate.

3 Create a Simulink model, and add the blocks shown in the following table.

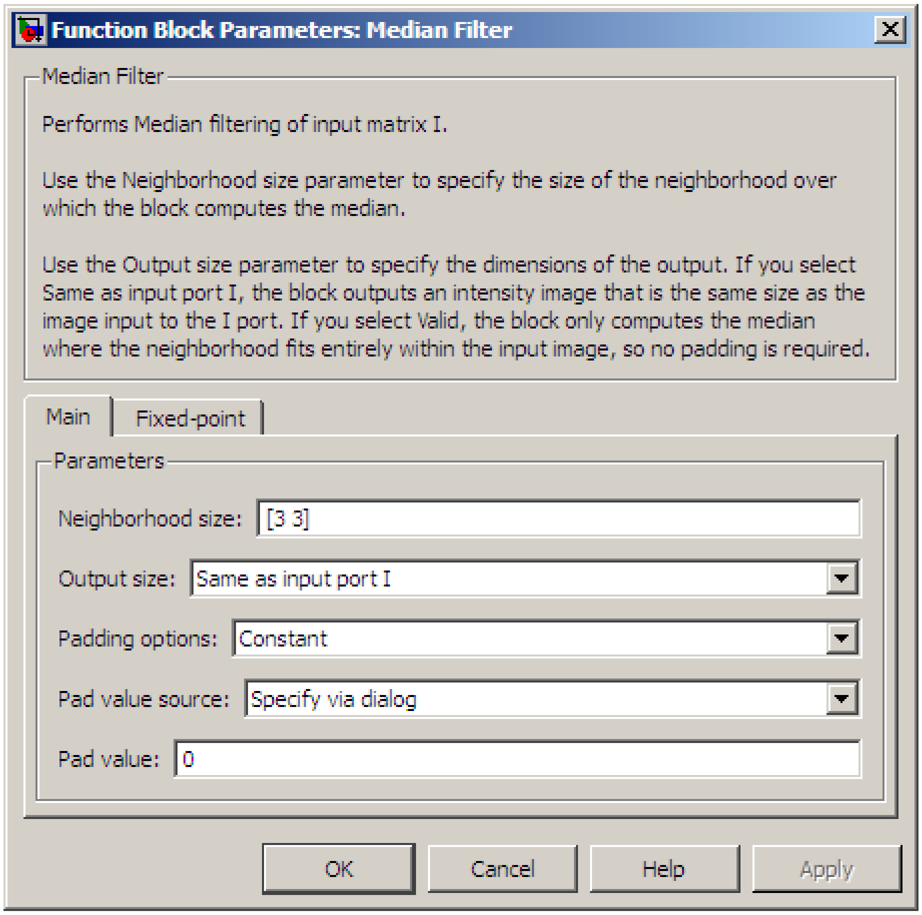
Block	Library	Quantity
Image From Workspace	Computer Vision System Toolbox > Sources	1
Median Filter	Computer Vision System Toolbox > Filtering	1
Video Viewer	Computer Vision System Toolbox > Sinks	2

4 Position the blocks as shown in the following figure.



5 Use the Image From Workspace block to import the noisy image into your model. Set the **Value** parameter to `I`.

6 Use the Median Filter block to eliminate the black and white speckles in the image. Use the default parameters.



Median Filter

Performs Median filtering of input matrix I.

Use the Neighborhood size parameter to specify the size of the neighborhood over which the block computes the median.

Use the Output size parameter to specify the dimensions of the output. If you select Same as input port I, the block outputs an intensity image that is the same size as the image input to the I port. If you select Valid, the block only computes the median where the neighborhood fits entirely within the input image, so no padding is required.

Parameters

Neighborhood size: [3 3]

Output size: Same as input port I

Padding options: Constant

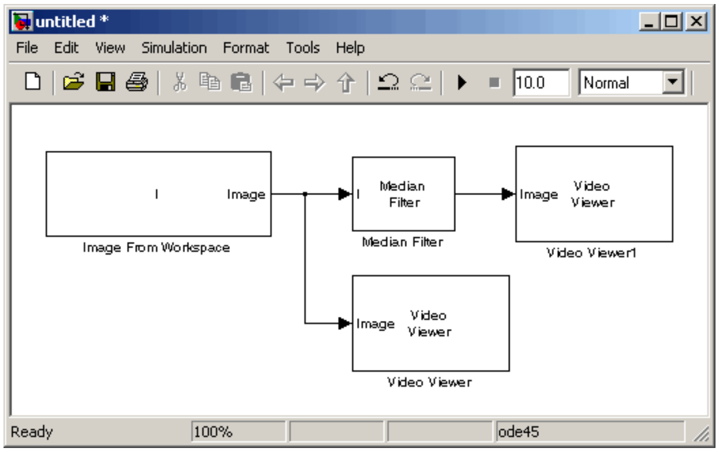
Pad value source: Specify via dialog

Pad value: 0

The Median Filter block replaces the central value of the 3-by-3 neighborhood with the median value of the neighborhood. This process removes the noise in the image.

7 Use the Video Viewer blocks to display the original noisy image, and the modified image. Images are represented by 8-bit unsigned integers. Therefore, a value of 0 corresponds to black and a value of 255 corresponds to white. Accept the default parameters.

8 Connect the blocks as shown in the following figure.

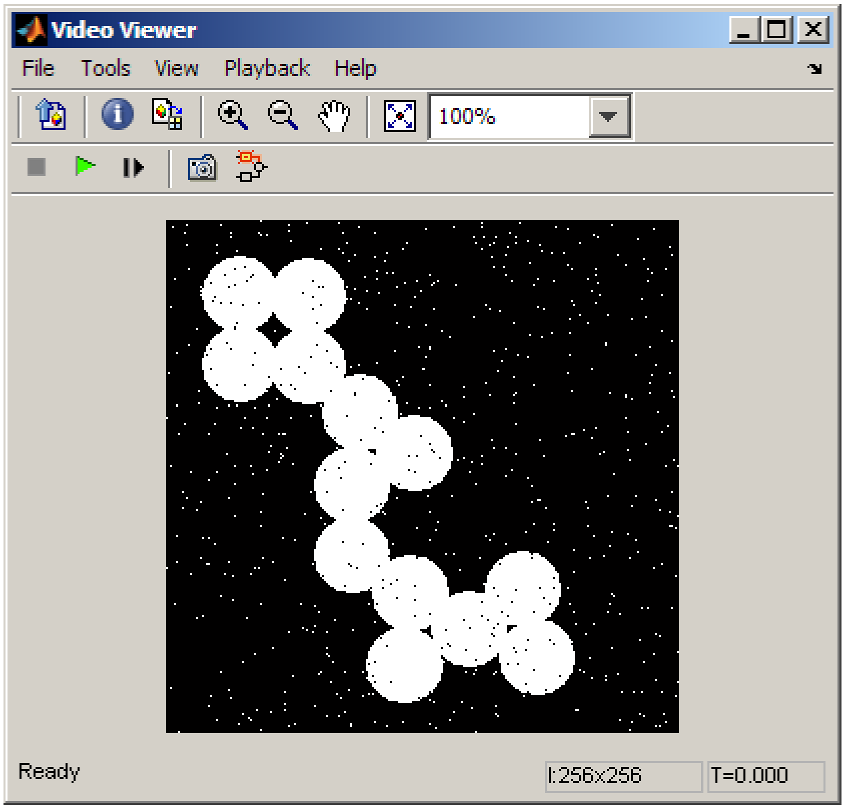


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

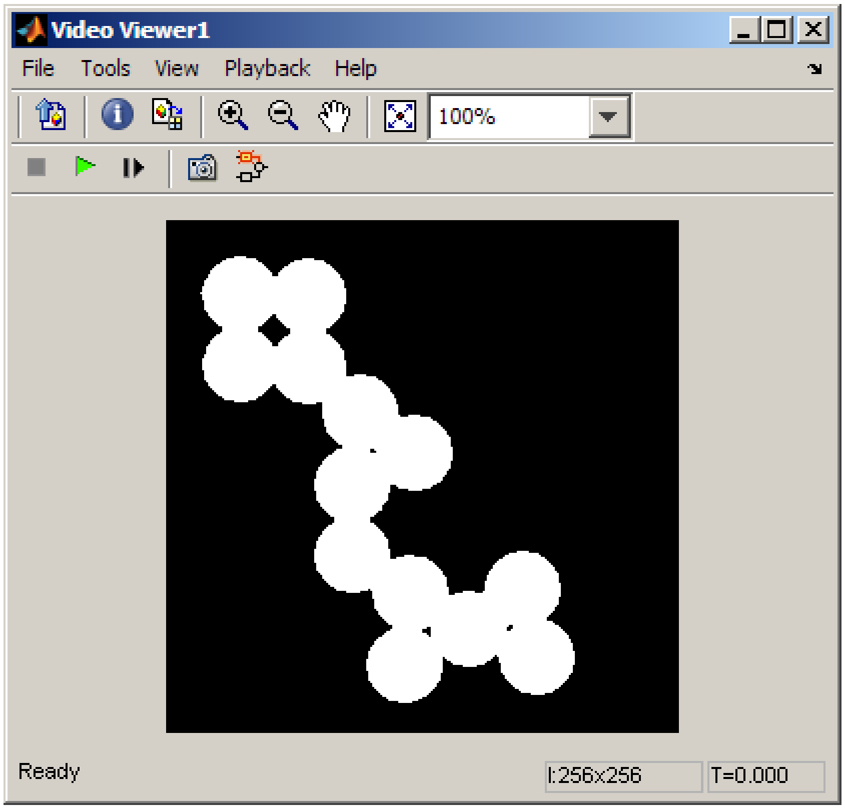
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

10 Run the model.

The original noisy image appears in the Video Viewer window. To view the image at its true size, right-click the window and select **Set Display To True Size**.



The cleaner image appears in the Video Viewer1 window. The following image is shown at its true size.



You have used the Median Filter block to remove noise from your image. For more information about this block, see the Median Filter block reference page in the *Computer Vision System Toolbox Reference*.

Sharpen an Image

To sharpen a color image, you need to make the luma intensity transitions more acute, while preserving the color information of the image. To do this, you convert an R'G'B' image into the Y'CbCr color space and apply a highpass filter to the luma portion of the image only. Then, you transform the image back to the R'G'B' color space to view the results. To blur an image, you apply a lowpass filter to the luma portion of the image. This example shows how to use the 2-D FIR Filter block to sharpen an image. The prime notation indicates that the signals are gamma corrected.

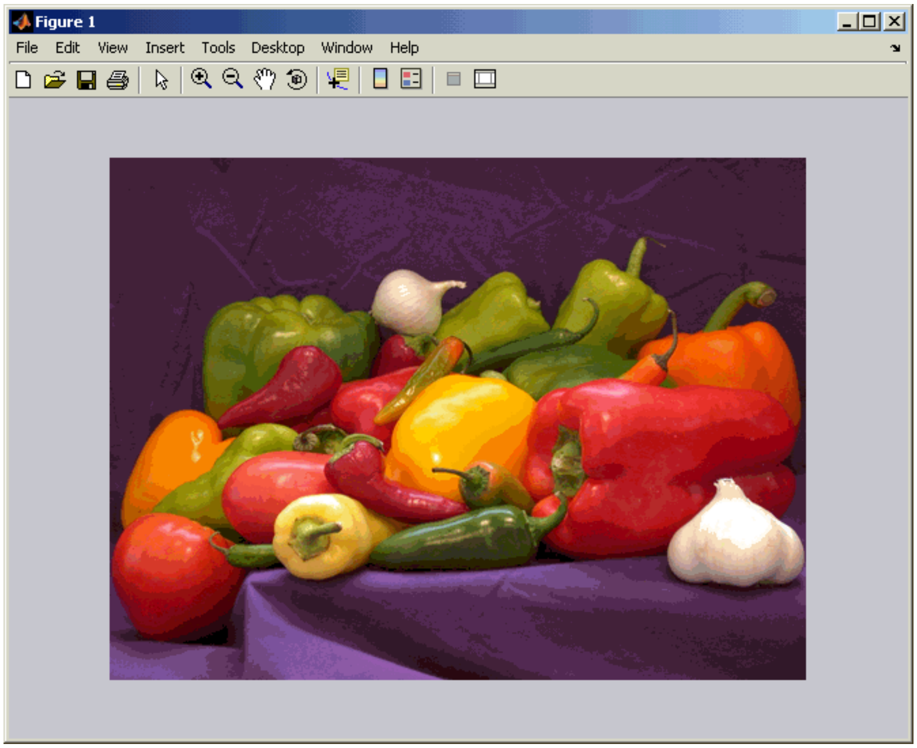
1 Define an R'G'B' image in the MATLAB workspace. To read in an R'G'B' image from a PNG file and cast it to the double-precision data type, at the MATLAB command prompt, type

   ```
   I= im2double(imread('peppers.png'));
   ```

   `I` is a 384-by-512-by-3 array of double-precision floating-point values. Each plane of this array represents the red, green, or blue color values of the image.

2 To view the image this array represents, at the MATLAB command prompt, type

   ```
   imshow(I)
   ```

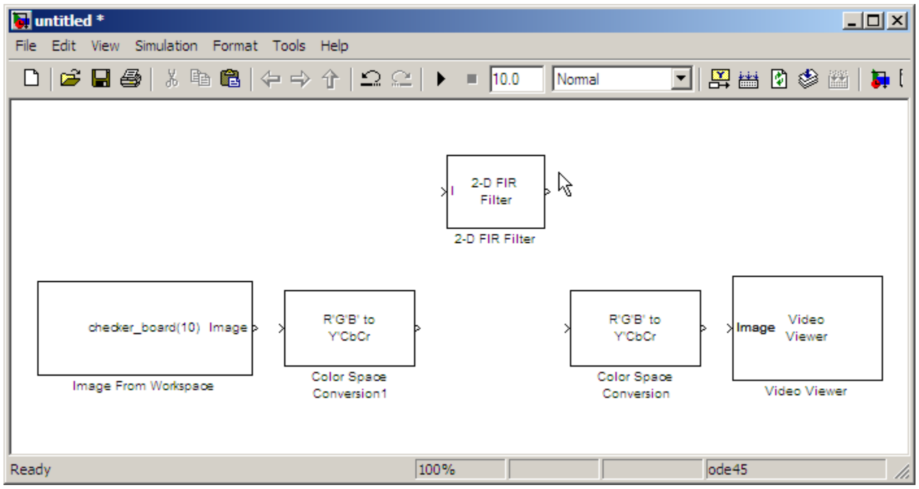


Now that you have defined your image, you can create your model.

3 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	**Library**	**Quantity**
Image From Workspace	Computer Vision System Toolbox > Sources	1
Color Space Conversion	Computer Vision System Toolbox > Conversions	2
2-D FIR Filter	Computer Vision System Toolbox > Filtering	1
Video Viewer	Computer Vision System Toolbox > Sinks	1

4 Position the blocks as shown in the following figure.

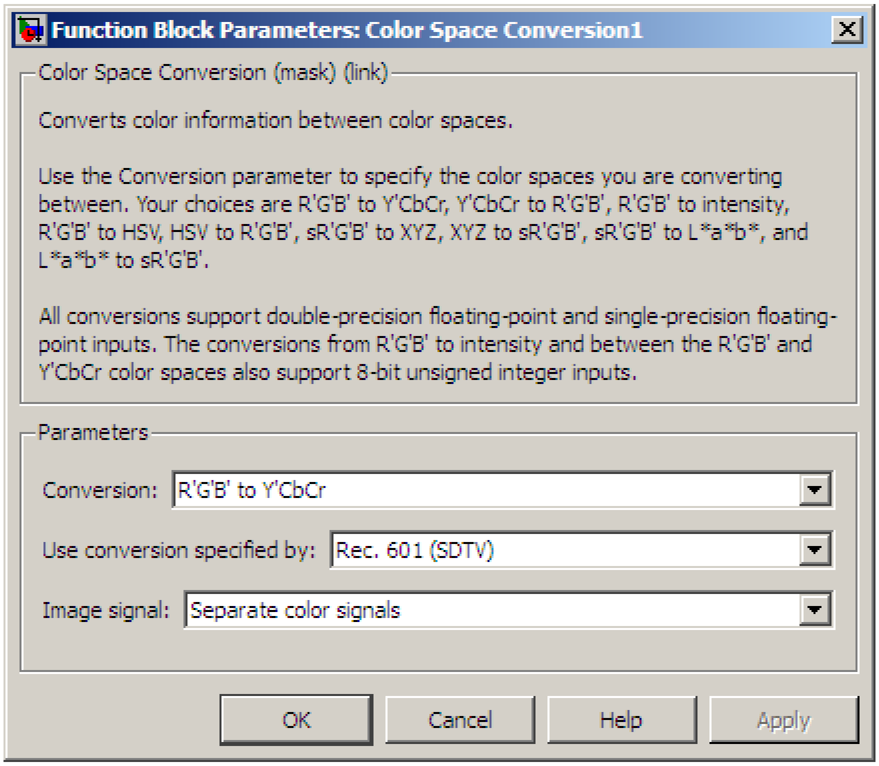


5 Use the Image From Workspace block to import the R'G'B' image from the MATLAB workspace. Set the parameters as follows:

- **Main** pane, **Value** = `I`
- **Main** pane, **Image signal** = `Separate color signals`

The block outputs the R', G', and B' planes of the `I` array at the output ports.

6 The first Color Space Conversion block converts color information from the R'G'B' color space to the Y'CbCr color space. Set the **Image signal** parameter to `Separate color signals`



Color Space Conversion (mask) (link)

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'.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

Parameters

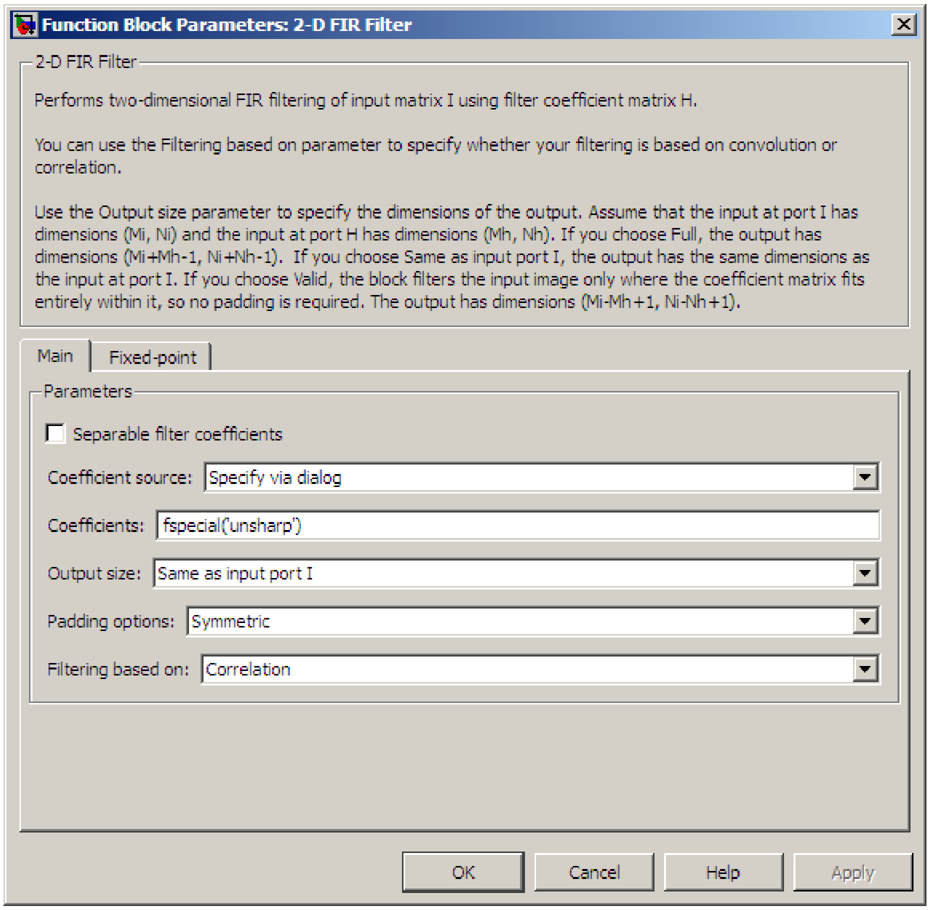
Conversion: R'G'B' to Y'CbCr

Use conversion specified by: Rec. 601 (SDTV)

Image signal: Separate color signals

7 Use the 2-D FIR Filter block to filter the luma portion of the image. Set the block parameters as follows:

- **Coefficients** = `fspecial('unsharp')`
- **Output size** = `Same as input port I`
- **Padding options** = `Symmetric`
- **Filtering based on** = `Correlation`


Function Block Parameters: 2-D FIR Filter

2-D FIR Filter

Performs two-dimensional FIR filtering of input matrix I using filter coefficient matrix H.

You can use the Filtering based on parameter to specify whether your filtering is based on convolution or correlation.

Use the Output size parameter to specify the dimensions of the output. Assume that the input at port I has dimensions (Mi, Ni) and the input at port H has dimensions (Mh, Nh). If you choose Full, the output has dimensions (Mi+Mh-1, Ni+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, 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).

Main | Fixed-point

Parameters

☐ Separable filter coefficients

Coefficient source: Specify via dialog

Coefficients: fspecial('unsharp')

Output size: Same as input port I

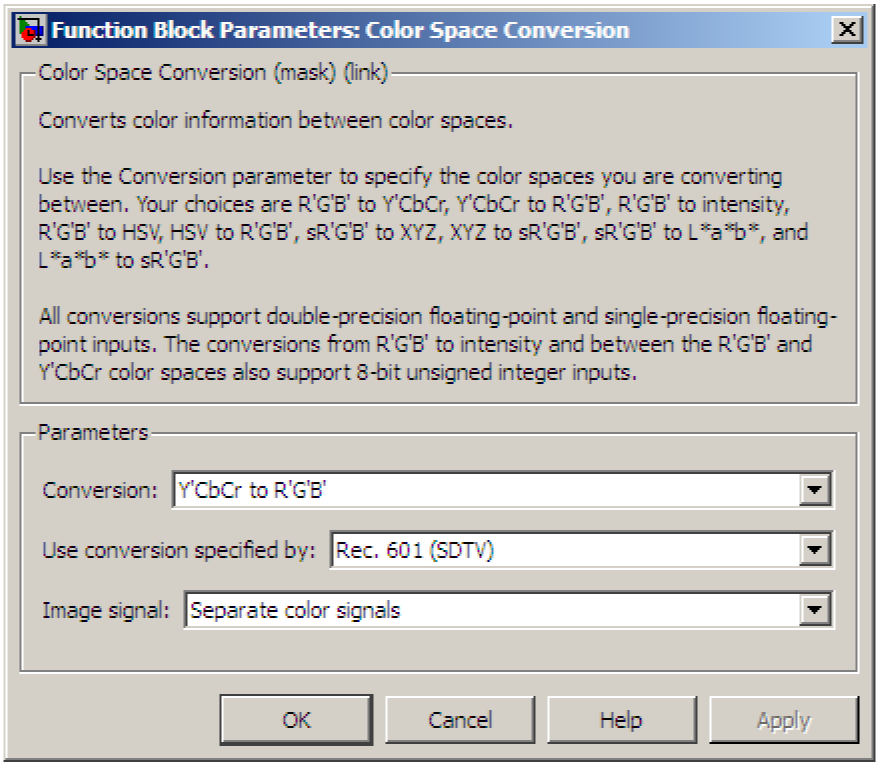
Padding options: Symmetric

Filtering based on: Correlation

The `fspecial('unsharp')` command creates two-dimensional highpass filter coefficients suitable for correlation. This highpass filter sharpens the image by removing the low frequency noise in it.

8 The second Color Space Conversion block converts the color information from the Y'CbCr color space to the R'G'B' color space. Set the block parameters as follows:

- **Conversion** = `Y'CbCr to R'G'B'`
- **Image signal** = `Separate color signals`



Color Space Conversion (mask) (link)

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'.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

Parameters

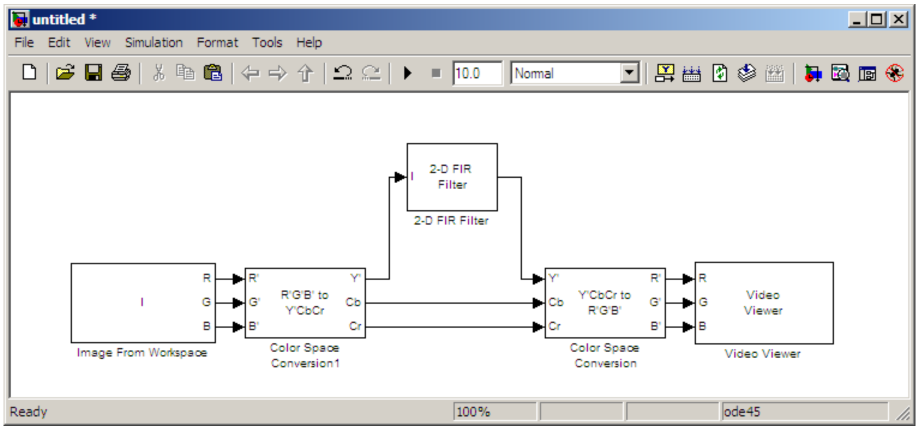
Conversion: Y'CbCr to R'G'B'

Use conversion specified by: Rec. 601 (SDTV)

Image signal: Separate color signals

9 Use the Video Viewer block to automatically display the new, sharper image in the Video Viewer window when you run the model. Set the **Image signal** parameter to `Separate color signals`, by selecting **File > Image Signal**.

10 Connect the blocks as shown in the following figure.

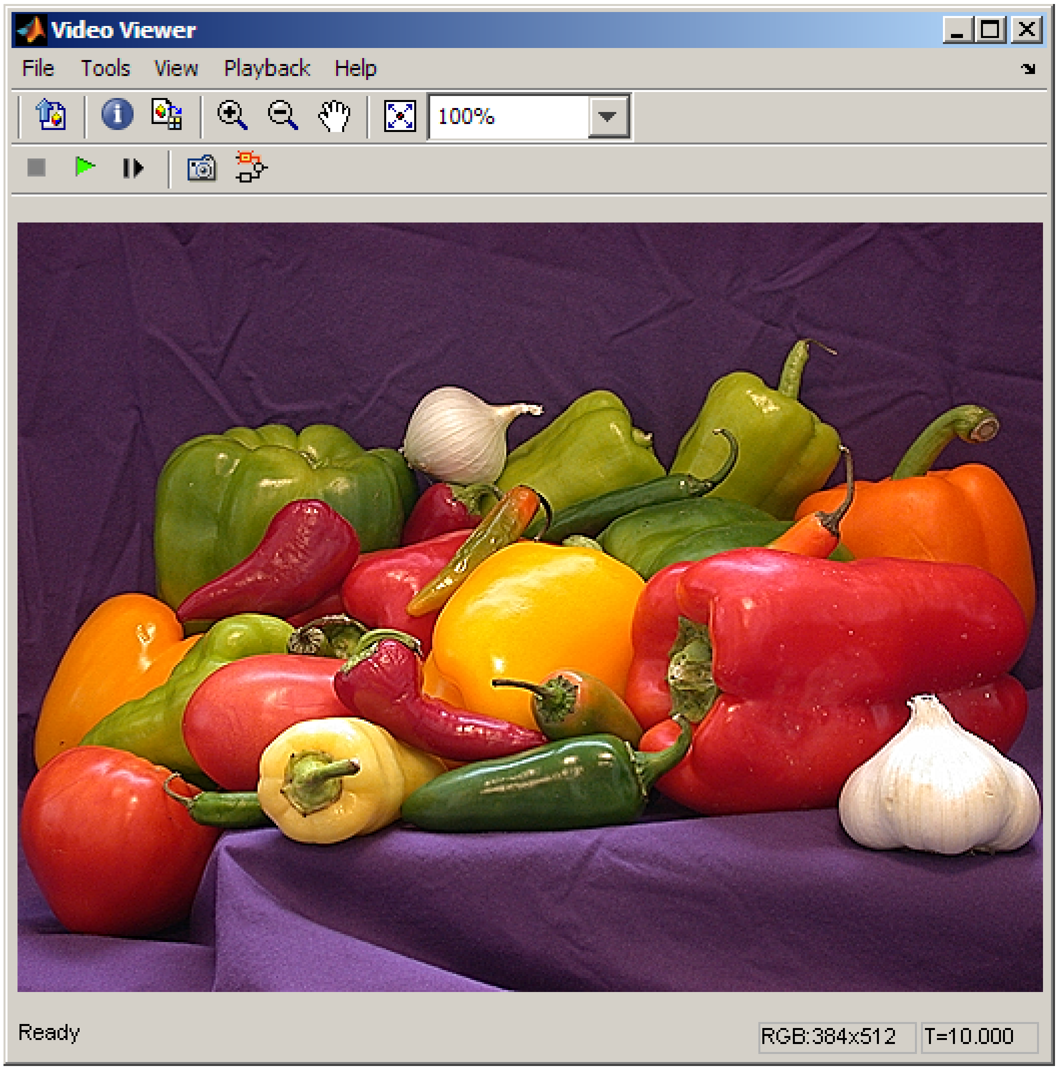


11 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

12 Run the model.

A sharper version of the original image appears in the Video Viewer window.



To blur the image, double-click the 2-D FIR Filter block. Set **Coefficients** parameter to `fspecial('gaussian',[15 15],7)` and then click **OK**. The `fspecial('gaussian',[15 15],7)` command creates two-dimensional Gaussian lowpass filter coefficients. This lowpass filter blurs the image by removing the high frequency noise in it.

In this example, you used the Color Space Conversion and 2-D FIR Filter blocks to sharpen an image. For more information, see the Color Space Conversion and 2-D FIR Filter, and `fspecial` reference pages.

Statistics and Morphological Operations

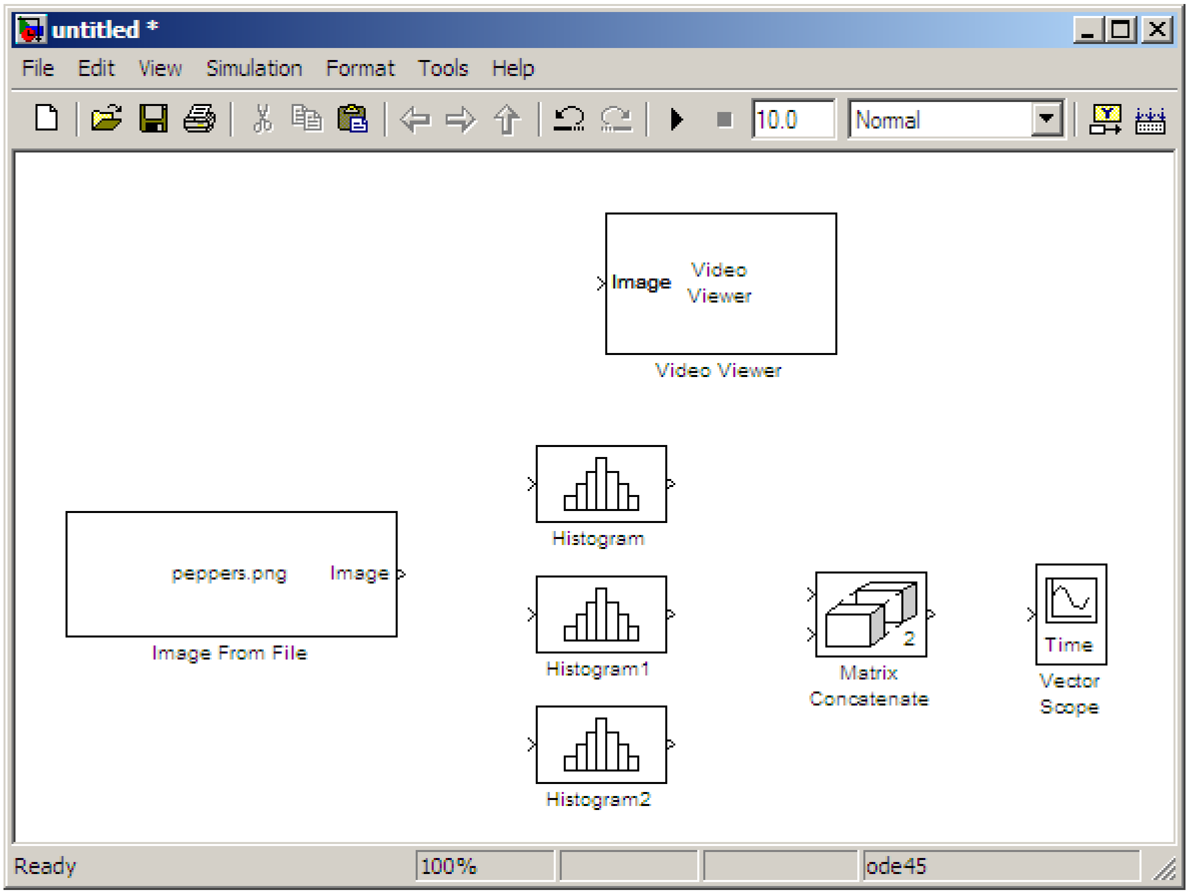
Find the Histogram of an Image

The Histogram block computes the frequency distribution of the elements in each input image by sorting the elements into a specified number of discrete bins. You can use the Histogram block to calculate the histogram of the R, G, and/or B values in an image. This example shows you how to accomplish this task:

1. Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Video Viewer	Computer Vision System Toolbox > Sinks	1
Matrix Concatenate	Simulink > Math Operations	1
Vector Scope	DSP System Toolbox > Signal Processing Sinks	1
Histogram	DSP System Toolbox > Statistics	3

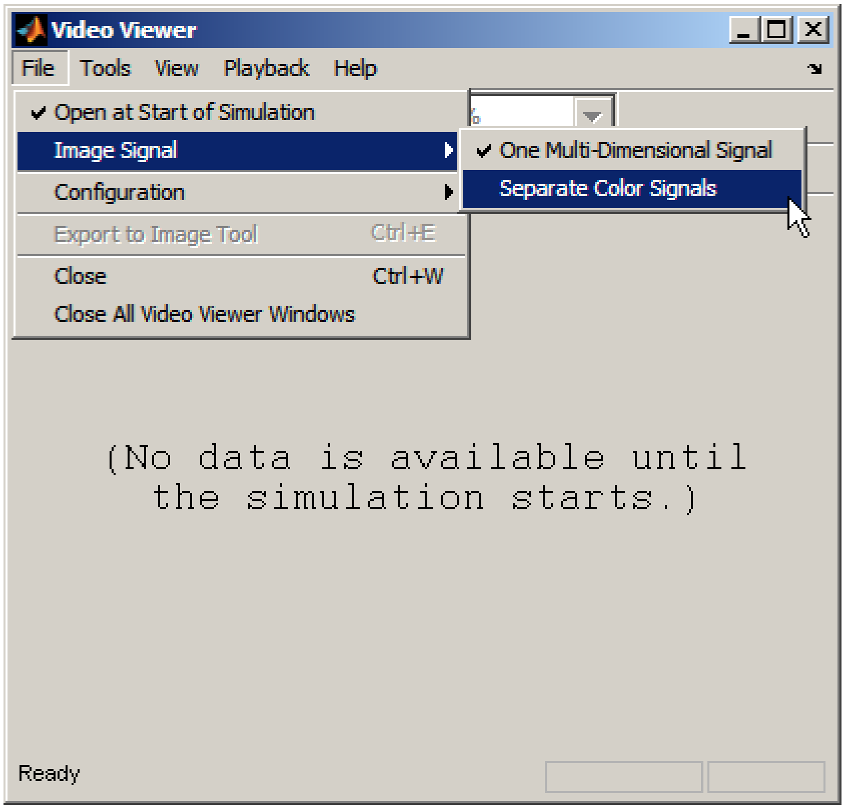
2. Place the blocks so that your model resembles the following figure.



3 Use the Image From File block to import an RGB image. Set the block parameters as follows:

- **Sample time** = `inf`
- **Image signal** = `Separate color signals`
- **Output port labels:** = `R|G|B`
- **Output data type:** = `double`

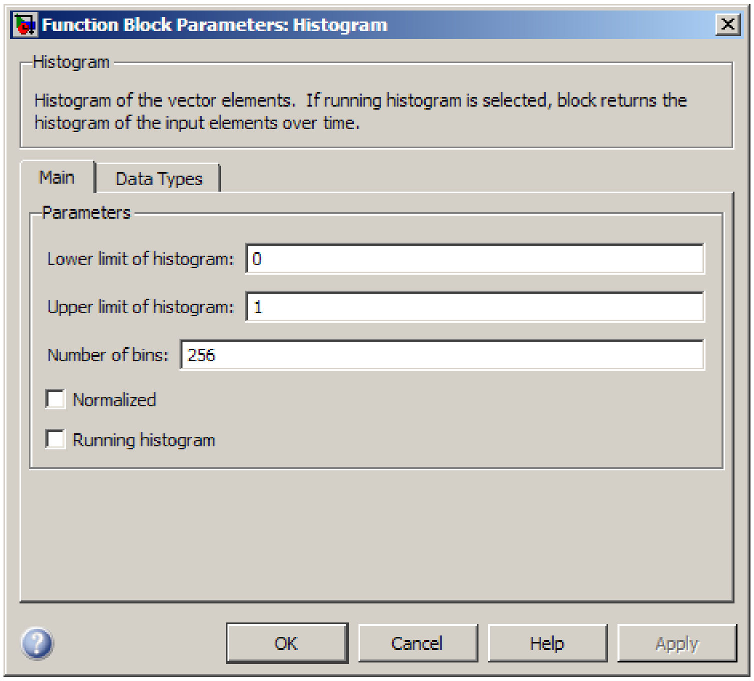
4 Use the Video Viewer block to automatically display the original image in the viewer window when you run the model. Set the **Image signal** parameter to `Separate color signals`.


Video Viewer
File Tools View Playback Help
✓ Open at Start of Simulation
Image Signal
Configuration
Export to Image Tool Ctrl+E
Close Ctrl+W
Close All Video Viewer Windows
✓ One Multi-Dimensional Signal
Separate Color Signals
Ready

(No data is available until the simulation starts.)

5 Use the Histogram blocks to calculate the histogram of the R, G, and B values in the image. Set the Main tab block parameters for the three Histogram blocks as follows:

- **Lower limit of histogram:** 0
- **Upper limit of histogram:** 1
- **Number of bins:** = 256


Function Block Parameters: Histogram

Histogram

Histogram of the vector elements. If running histogram is selected, block returns the histogram of the input elements over time.

Main | Data Types

Parameters

Lower limit of histogram: 0

Upper limit of histogram: 1

Number of bins: 256

☐ Normalized

☐ Running histogram

The **R**, **G**, and **B** input values to the Histogram block are double-precision floating point and range between 0 and 1. The block creates 256 bins between the maximum and minimum input values and counts the number of R, G, and B values in each bin.

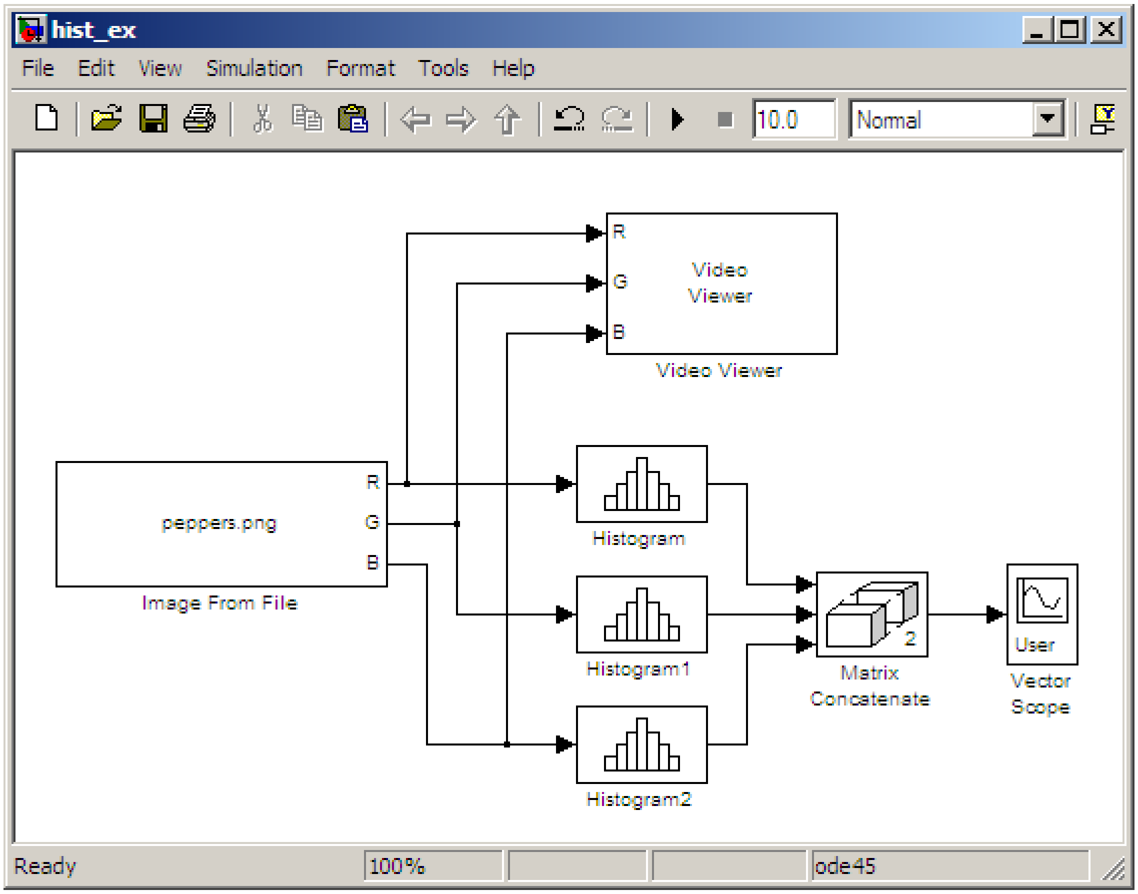
6 Use the Matrix Concatenate block to concatenate the R, G, and B column vectors into a single matrix so they can be displayed using the Vector Scope block. Set the **Number of inputs** parameter to 3.

7 Use the Vector Scope block to display the histograms of the R, G, and B values of the input image. Set the block parameters as follows:

- **Scope Properties** pane, **Input domain** = `User-defined`
- **Display Properties** pane, clear the **Frame number** check box
- **Display Properties** pane, select the **Channel legend** check box
- **Display Properties** pane, select the **Compact display** check box

- **Axis Properties** pane, clear the **Inherit sample increment from input** check box.
- **Axis Properties** pane, **Minimum Y-limit** = `0`
- **Axis Properties** pane, **Maximum Y-limit** = `1`
- **Axis Properties** pane, **Y-axis label** = `Count`
- **Line Properties** pane, **Line markers** = `.|s|d`
- **Line Properties** pane, **Line colors** = `[1 0 0]|[0 1 0]|[0 0 1]`

8 Connect the blocks as shown in the following figure.

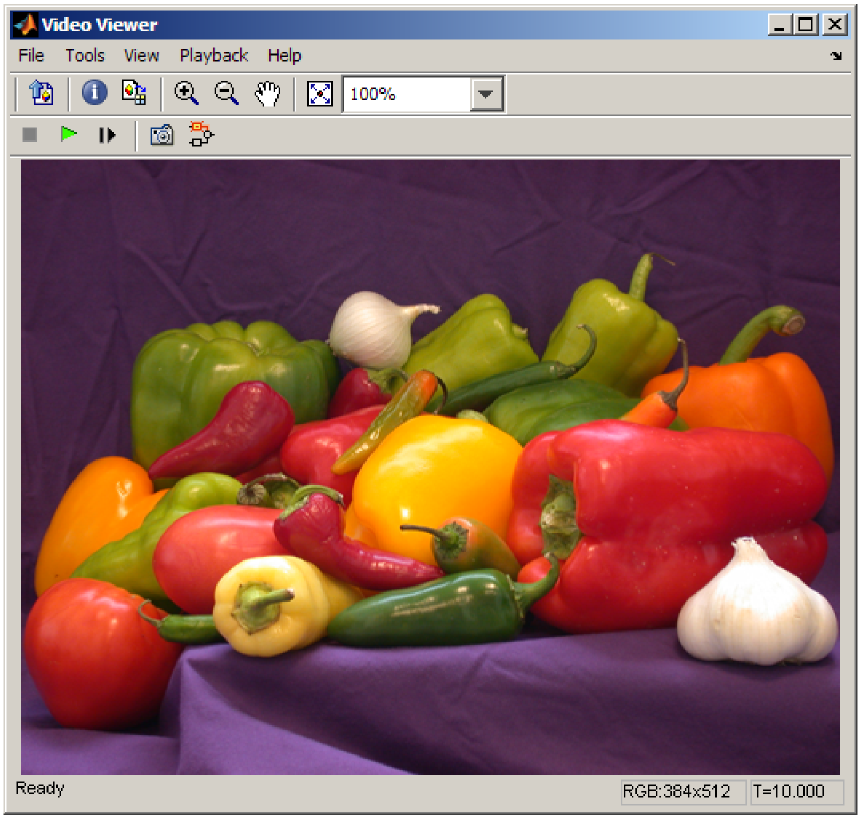


9 Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `Discrete (no continuous states)`

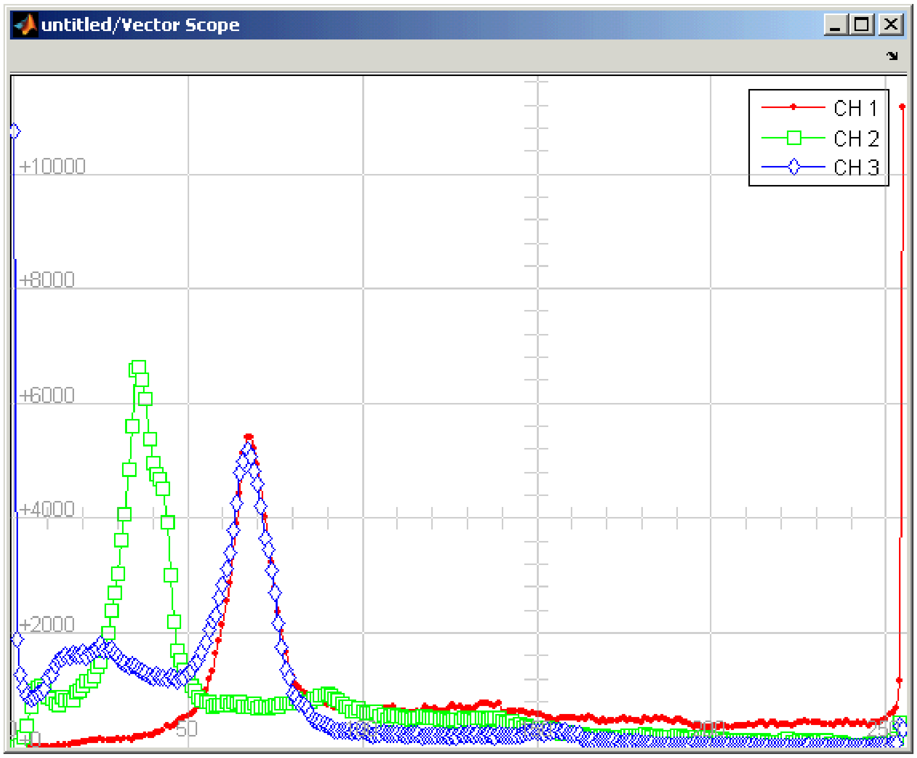
10 Run the model using either the simulation button, or by selecting Simulation > Start.

The original image appears in the Video Viewer window.



11 Right-click in the Vector Scope window and select **Autoscale**.

The scaled histogram of the image appears in the Vector Scope window.



You have now used the Histogram block to calculate the histogram of the R, G, and B values in an RGB image. For more information about this block, see the Histogram reference page. To open a demo model that illustrates how to use this block to calculate the histogram of the R, G, and B values in an RGB video stream, type `viphistogram` at the MATLAB command prompt.

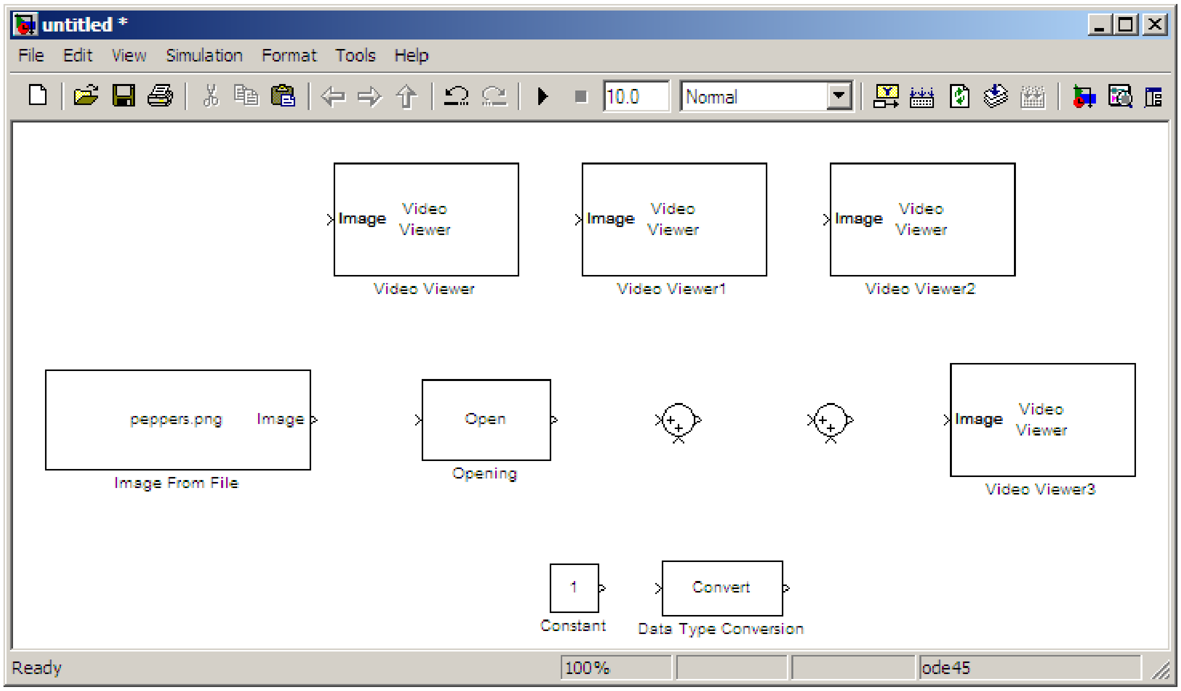
Correct Nonuniform Illumination

Global threshold techniques, which are often the first step in object measurement, cannot be applied to unevenly illuminated images. To correct this problem, you can change the lighting conditions and take another picture, or you can use morphological operators to even out the lighting in the image. Once you have corrected for nonuniform illumination, you can pick a global threshold that delineates every object from the background. In this topic, you use the Opening block to correct for uneven lighting in an intensity image:

1. Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Computer Vision System Toolbox > Sources	1
Opening	Computer Vision System Toolbox > Morphological Operations	1
Video Viewer	Computer Vision System Toolbox > Sinks	4
Constant	Simulink > Sources	1
Sum	Simulink > Math Operations	2
Data Type Conversion	Simulink > Signal Attributes	1

2. Position the blocks as shown in the following figure.

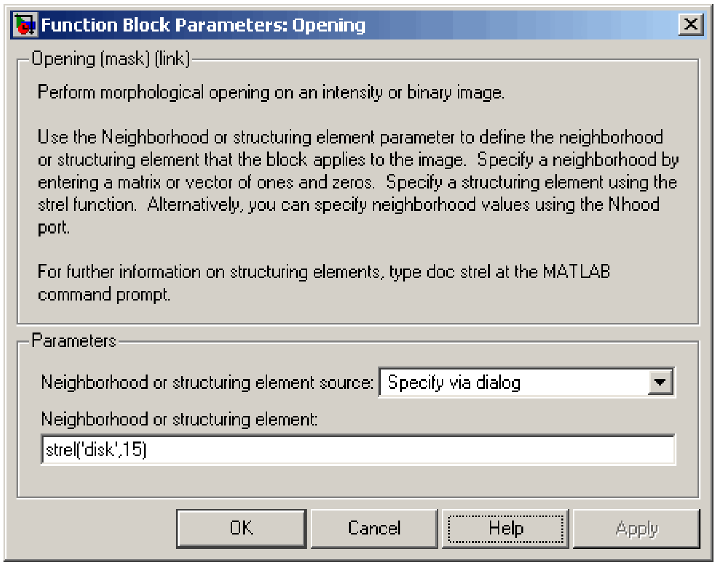


Once you have assembled the blocks required to correct for uneven illumination, you need to set your block parameters. To do this, double-click the blocks, modify the block parameter values, and click **OK**.

3 Use the Image From File block to import the intensity image. Set the **File name** parameter to `rice.png`. This image is a 256-by-256 matrix of 8-bit unsigned integer values.

4 Use the Video Viewer block to view the original image. Accept the default parameters.

5 Use the Opening block to estimate the background of the image. Set the **Neighborhood or structuring element** parameter to `strel('disk',15)`.


Function Block Parameters: Opening

Opening (mask) (link)

Perform morphological opening on an intensity or binary image.

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 ones and zeros. Specify a structuring element using the strel function. Alternatively, you can specify neighborhood values using the Nhood port.

For further information on structuring elements, type doc strel at the MATLAB command prompt.

Parameters

Neighborhood or structuring element source: Specify via dialog

Neighborhood or structuring element:

strel('disk',15)

The `strel` function creates a circular STREL object with a radius of 15 pixels. When working with the Opening block, pick a STREL object that fits within the objects you want to keep. It often takes experimentation to find the neighborhood or STREL object that best suits your application.

6 Use the Video Viewer1 block to view the background estimated by the Opening block. Accept the default parameters.

7 Use the first Sum block to subtract the estimated background from the original image. Set the block parameters as follows:

- **Icon shape** = `rectangular`
- **List of signs** = `-+`

8 Use the Video Viewer2 block to view the result of subtracting the background from the original image. Accept the default parameters.

9 Use the Constant block to define an offset value. Set the **Constant value** parameter to `80`.

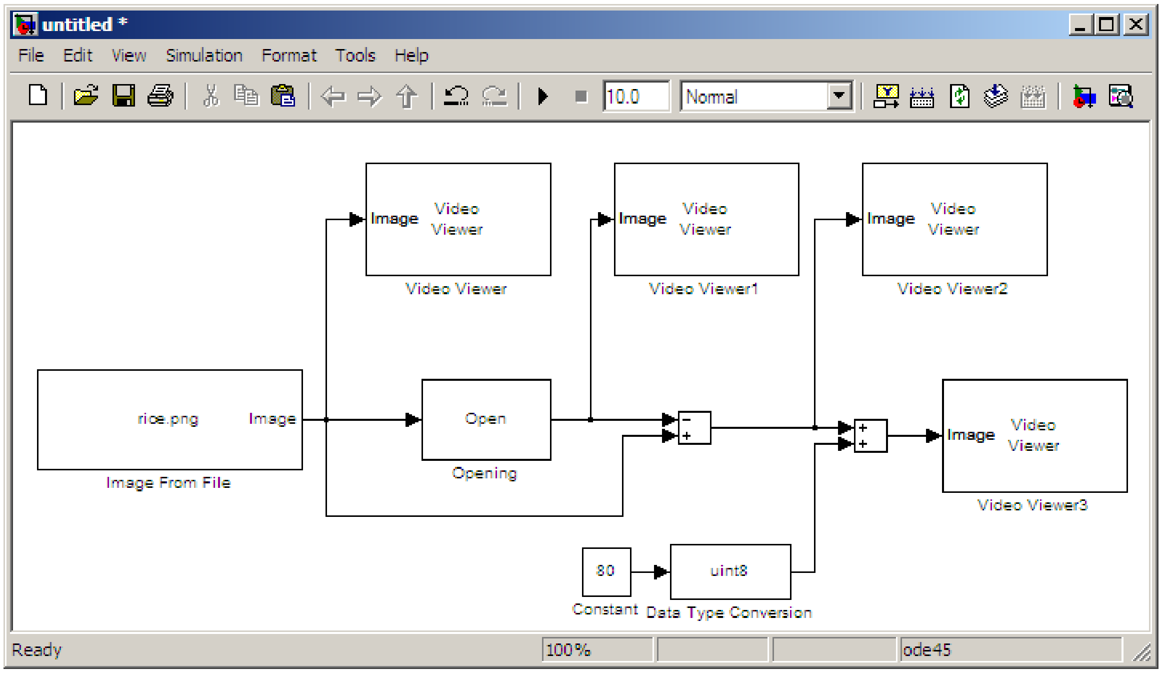
10 Use the Data Type Conversion block to convert the offset value to an 8-bit unsigned integer. Set the **Output data type mode** parameter to `uint8`.

11 Use the second Sum block to lighten the image so that it has the same brightness as the original image. Set the block parameters as follows:

- **Icon shape** = `rectangular`
- **List of signs** = `++`

12 Use the Video Viewer3 block to view the corrected image. Accept the default parameters.

13 Connect the blocks as shown in the following figure.

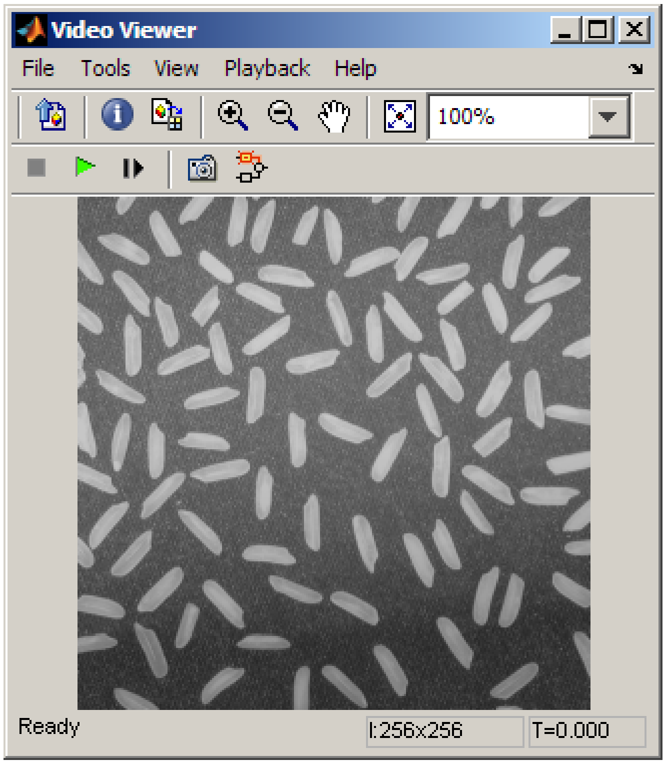


14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

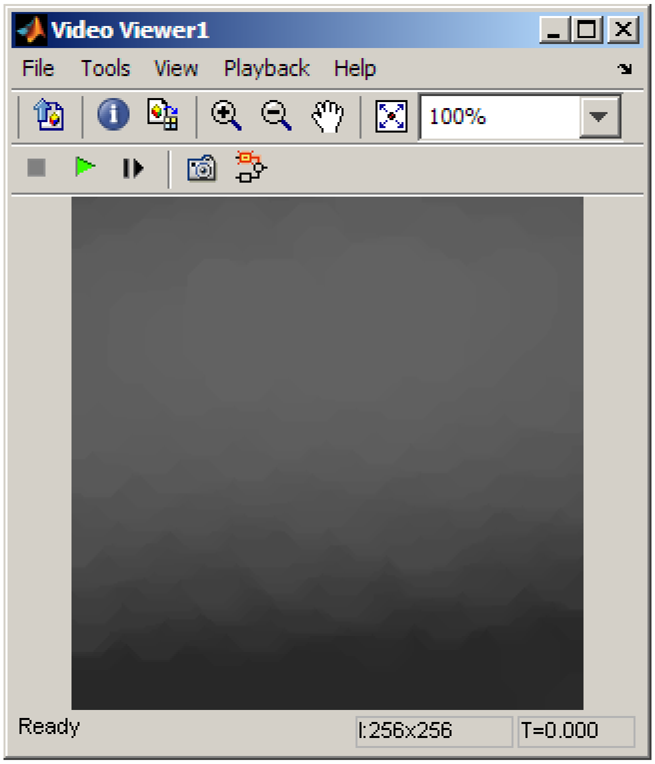
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `discrete (no continuous states)`

15 Run the model.

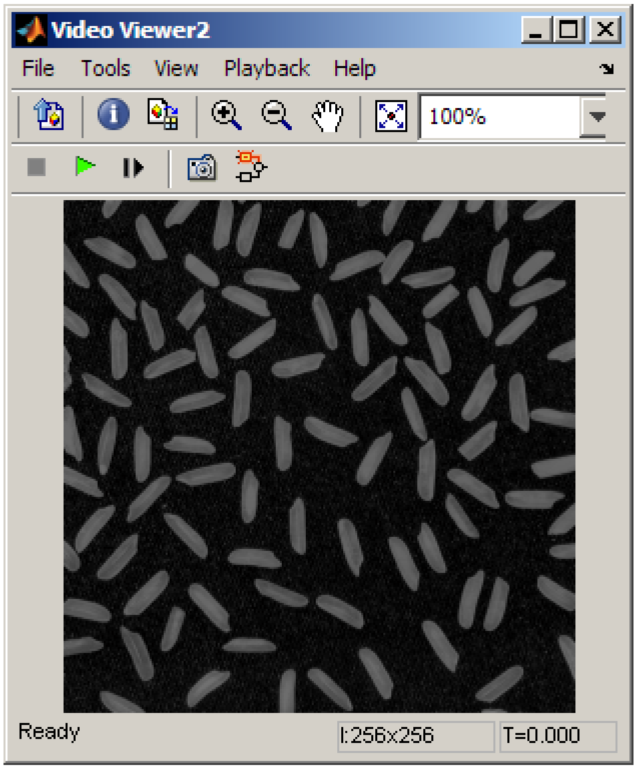
The original image appears in the Video Viewer window.



The estimated background appears in the Video Viewer1 window.

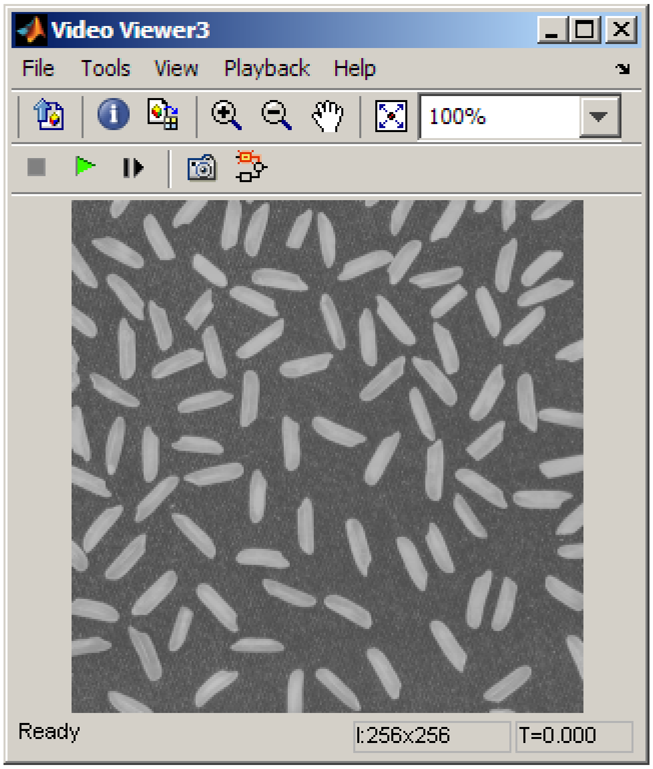


The image without the estimated background appears in the Video Viewer2 window.



The preceding image is too dark. The Constant block provides an offset value that you used to brighten the image.

The corrected image, which has even lighting, appears in the Video Viewer3 window. The following image is shown at its true size.



In this section, you have used the Opening block to remove irregular illumination from an image. For more information about this block, see the Opening reference page. For related information, see the Top-hat block reference page. For more information about STREL objects, see the `strel` function in the Image Processing Toolbox documentation.

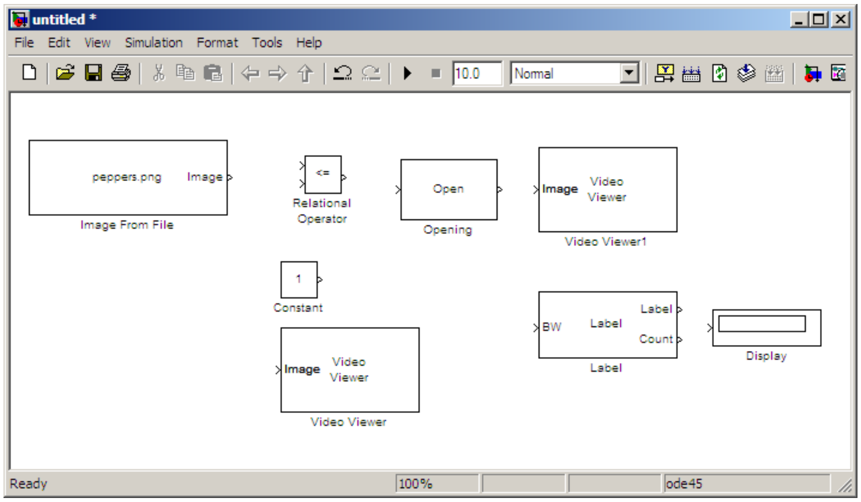
Count Objects in an Image

In this example, you import an intensity image of a wheel from the MATLAB workspace and convert it to binary. Then, using the Opening and Label blocks, you count the number of spokes in the wheel. You can use similar techniques to count objects in other intensity images. However, you might need to use additional morphological operators and different structuring elements:

1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	**Library**	**Quantity**
Image From File	Computer Vision System Toolbox > Sources	1
Opening	Computer Vision System Toolbox> Morphological Operations	1
Label	Computer Vision System Toolbox > Morphological Operations	1
Video Viewer	Computer Vision System Toolbox > Sinks	2
Constant	Simulink > Sources	1
Relational Operator	Simulink > Logic and Bit Operations	1
Display	DSP System Toolbox > Signal Processing Sinks	1

2 Position the blocks as shown in the following figure. The unconnected ports disappear when you set block parameters.



You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

3 Use the Image From File block to import your image. Set the **File name** parameter to `testpat1.png`. This is a 256-by-256 matrix image of 8-bit unsigned integers.

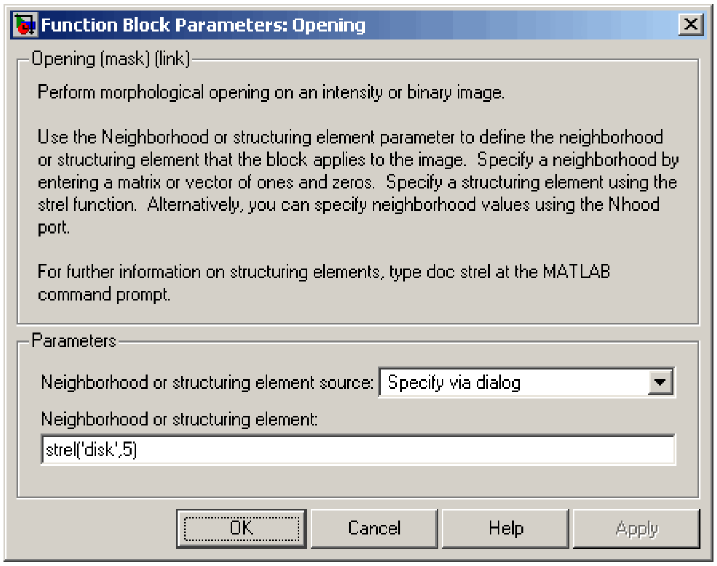
4 Use the Constant block to define a threshold value for the Relational Operator block. Set the **Constant value** parameter to `200`.

5 Use the Video Viewer block to view the original image. Accept the default parameters.

6 Use the Relational Operator block to perform a thresholding operation that converts your intensity image to a binary image. Set the **Relational Operator** parameter to `<`.

If the input to the Relational Operator block is less than 200, its output is 1; otherwise, its output is 0. You must threshold your intensity image because the Label block expects binary input. Also, the objects it counts must be white.

7 Use the Opening block to separate the spokes from the rim and from each other at the center of the wheel. Use the default parameters.



Perform morphological opening on an intensity or binary image.

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 ones and zeros. Specify a structuring element using the strel function. Alternatively, you can specify neighborhood values using the Nhood port.

For further information on structuring elements, type doc strel at the MATLAB command prompt.

Parameters

Neighborhood or structuring element source: Specify via dialog

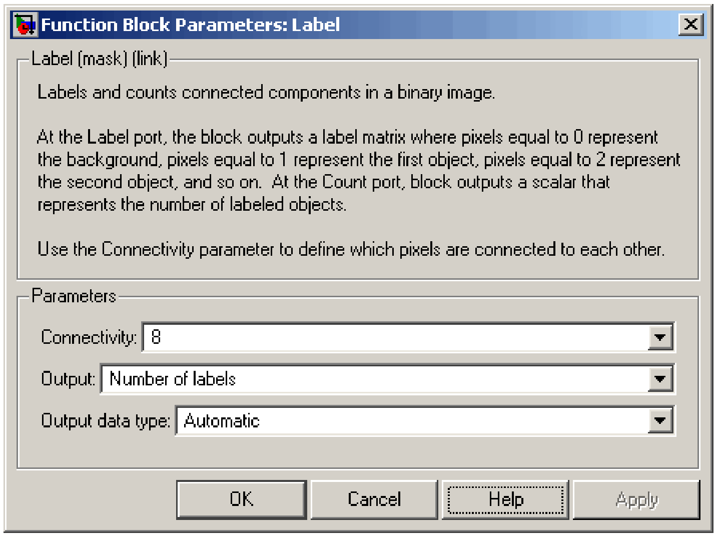
Neighborhood or structuring element:

```
strel('disk',5)
```

The `strel` function creates a circular STREL object with a radius of 5 pixels. When working with the Opening block, pick a STREL object that fits within the objects you want to keep. It often takes experimentation to find the neighborhood or STREL object that best suits your application.

8 Use the Video Viewer1 block to view the opened image. Accept the default parameters.

9 Use the Label block to count the number of spokes in the input image. Set the **Output** parameter to `Number of labels`.


Function Block Parameters: Label

Label (mask) (link)

Labels and counts connected components in a binary image.

At the Label port, the block outputs 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. At the Count port, block outputs a scalar that represents the number of labeled objects.

Use the Connectivity parameter to define which pixels are connected to each other.

Parameters

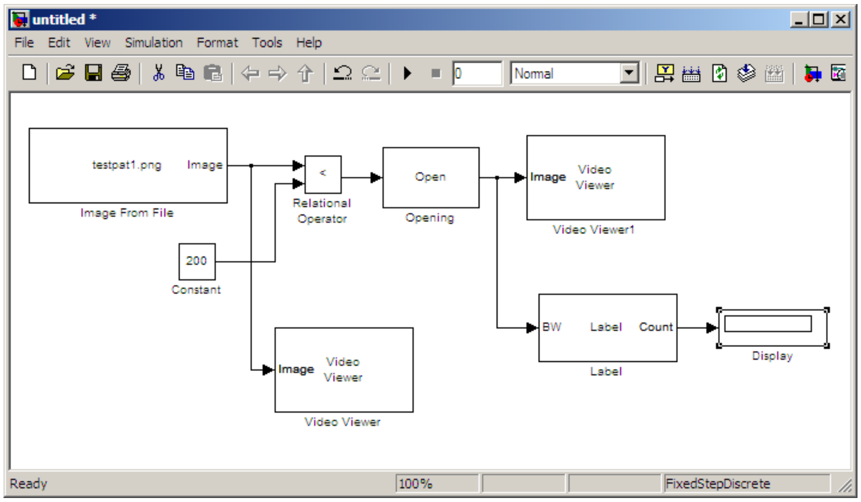
Connectivity: 8

Output: Number of labels

Output data type: Automatic

10 The Display block displays the number of spokes in the input image. Use the default parameters.

11 Connect the block as shown in the following figure.

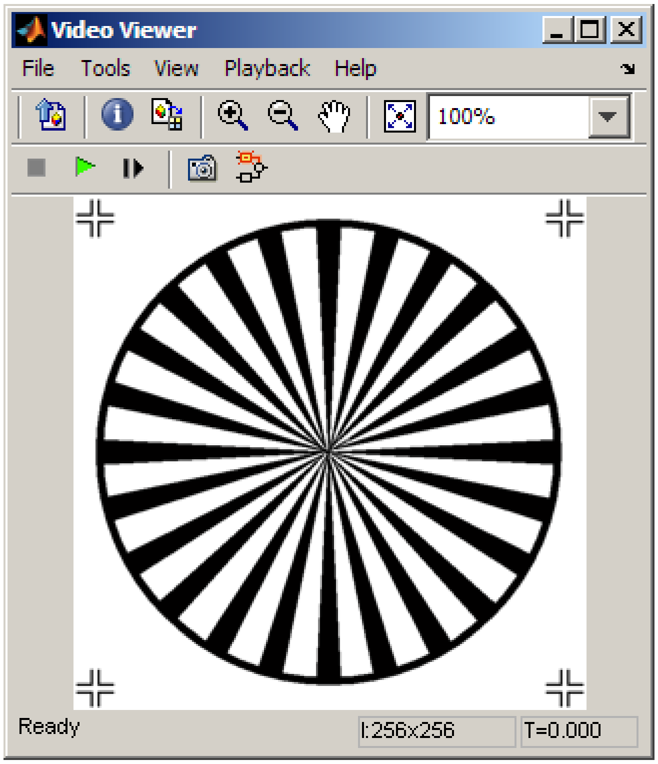


12 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

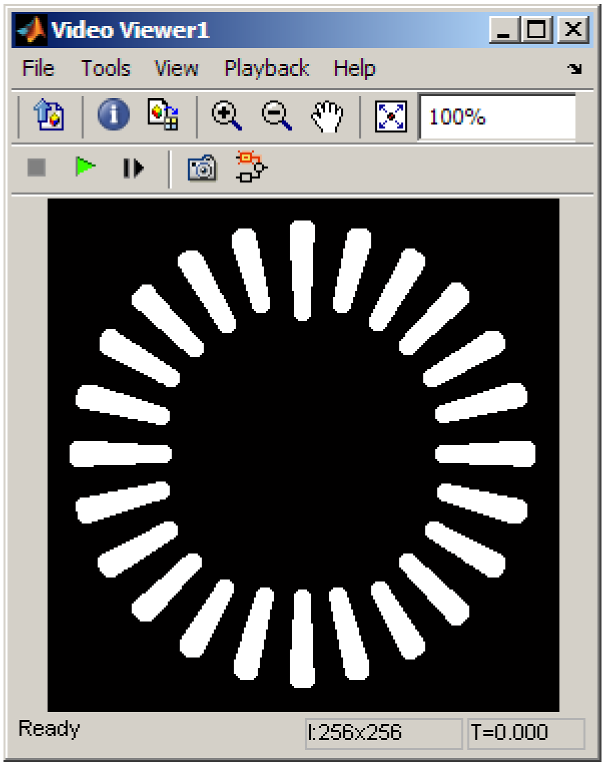
- **Solver** pane, **Stop time** = `0`
- **Solver** pane, **Type** = `Fixed-step`
- **Solver** pane, **Solver** = `discrete (no continuous states)`

13 Run the model.

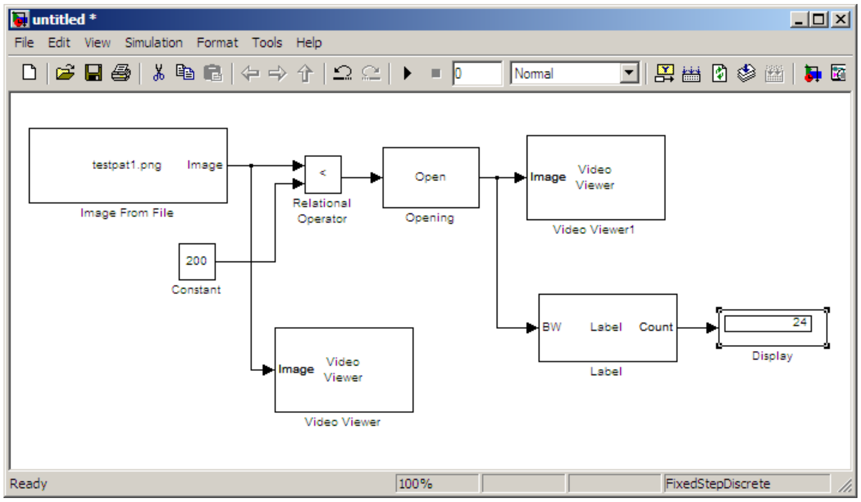
The original image appears in the Video Viewer1 window. To view the image at its true size, right-click the window and select **Set Display To True Size**.



The opened image appears in the Video Viewer window. The following image is shown at its true size.



As you can see in the preceding figure, the spokes are now separate white objects. In the model, the Display block correctly indicates that there are 24 distinct spokes.



You have used the Opening and Label blocks to count the number of spokes in an image. For more information about these blocks, see the Opening and Label block reference pages in the *Computer Vision System Toolbox Reference*. If you want to send the number of spokes to the MATLAB workspace, use the To Workspace block in Simulink or the Signal to Workspace block in DSP System Toolbox. For more information about STREL objects, see `strel` in the Image Processing Toolbox documentation.

Code Generation

System Objects that Generate Code

The following Computer Vision System Toolbox System objects support code generation in MATLAB. See "Use System Objects for Code Generation from MATLAB" and "Use System Objects for Code Generation from MATLAB".

Supported Computer Vision System Toolbox System Objects

Object	Description
Analysis & Enhancement	
`vision.BoundaryTracer`	Trace object boundaries in binary images
`vision.ContrastAdjuster`	Adjust image contrast by linear scaling
`vision.Deinterlacer`	Remove motion artifacts by deinterlacing input video signal
`vision.EdgeDetector`	Find edges of objects in images
`vision.ForegroundDetector`	Detect foreground using Gaussian Mixture Models
`vision.HistogramEqualizer`	Enhance contrast of images using histogram equalization
`vision.TemplateMatcher`	Perform template matching by shifting template over image
Conversions	
`vision.Autothresholder`	Convert intensity image to binary image
`vision.ChromaResampler`	Downsample or upsample chrominance components of images
`vision.ColorSpaceConverter`	Convert color information between color spaces
`vision.DemosaicInterpolator`	Demosaic Bayer's format images
`vision.GammaCorrector`	Apply or remove gamma correction from images or video streams
`vision.ImageComplementer`	Compute complement of pixel values in binary, intensity, or RGB images

Supported Computer Vision System Toolbox System Objects (Continued)

Object	Description
`vision.ImageDataTypeConverter`	Convert and scale input image to specified output data type
Filtering	
`vision.Convolver`	Compute 2-D discrete convolution of two input matrices
`vision.ImageFilter`	Perform 2-D FIR filtering of input matrix
`vision.MedianFilter`	2D median filtering
Geometric Transformations	
`vision.GeometricRotator`	Rotate image by specified angle
`vision.GeometricRotator`	Enlarge or shrink image size
`vision.GeometricScaler`	Shift rows or columns of image by linearly varying offset
`vision.GeometricTransformer`	Apply projective or affine transformation to an image
`vision.GeometricTransformEstimator`	Estimate geometric transformation from matching point pairs
`vision.GeometricTranslator`	Translate image in two-dimensional plane using displacement vector
Morphological Operations	
`vision.ConnectedComponentLabeler`	Label and count the connected regions in a binary image
`vision.MorphologicalClose`	Perform morphological closing on image
`vision.MorphologicalDilate`	Perform morphological dilation on an image
`vision.MorphologicalErode`	Perform morphological erosion on an image

Supported Computer Vision System Toolbox System Objects (Continued)

Object	Description
`vision.MorphologicalOpen`	Perform morphological opening on an image
Sinks	
`vision.DeployableVideoPlayer`	Send video data to computer screen
`vision.VideoFileWriter`	Write video frames and audio samples to multimedia file
Sources	
`vision.VideoFileReader`	Read video frames and audio samples from compressed multimedia file
Statistics	
`vision.Autocorrelator`	Compute 2-D autocorrelation of input matrix
`vision.BlobAnalysis`	Compute statistics for connected regions in a binary image
`vision.Crosscorrelator`	Compute 2-D cross-correlation of two input matrices
`vision.Histogram`	Generate histogram of each input matrix
`vision.LocalMaximaFinder`	Find local maxima in matrices
`vision.Maximum`	Find maximum values in input or sequence of inputs
`vision.Mean`	Find mean value of input or sequence of inputs
`vision.Median`	Find median values in an input
`vision.Minimum`	Find minimum values in input or sequence of inputs
`vision.PSNR`	Compute peak signal-to-noise ratio (PSNR) between images

Supported Computer Vision System Toolbox System Objects (Continued)

Object	Description
`vision.StandardDeviation`	Find standard deviation of input or sequence of inputs
`vision.Variance`	Find variance values in an input or sequence of inputs
Text & Graphics	
`vision.AlphaBlender`	Combine images, overlay images, or highlight selected pixels
`vision.MarkerInserter`	Draw markers on output image
`vision.ShapeInserter`	Draw rectangles, lines, polygons, or circles on images
`vision.TextInserter`	Draw text on image or video stream
Transforms	
`vision.DCT`	Compute 2-D discrete cosine transform
`vision.FFT`	Two-dimensional discrete Fourier transform
`vision.HoughLines`	Find Cartesian coordinates of lines that are described by rho and theta pairs
`vision.HoughTransform`	Find lines in images via Hough transform
`vision.IDCT`	Compute 2-D inverse discrete cosine transform
`vision.IFFT`	Two–dimensional inverse discrete Fourier transform
`vision.Pyramid`	Perform Gaussian pyramid decomposition
Utilities	
`vision.ImagePadder`	Pad or crop input image along its rows, columns, or both

Functions that Generate Code

The following Computer Vision System Toolbox functions support code generation in MATLAB. See "Use System Objects for Code Generation from MATLAB" for more information.

Function	Description
`epipolarLine`	Compute epipolar lines for stereo images
`estimateFundamentalMatrix`	Estimate fundamental matrix from corresponding points in stereo image
`estimateUncalibratedRectification`	Uncalibrated stereo rectification
`extractFeatures`	Extract interest point descriptors
`isEpipoleInImage`	Determine whether image contains epipole
`lineToBorderPoints`	Intersection points of lines in image and image border
`matchFeatures`	Find matching image features

Shared Library Dependencies

In general, the code you generate from Computer Vision System Toolbox blocks is portable ANSI® C code. After you generate the code, you can deploy it on another machine. For more information on how to do so, see “Relocating Code to Another Development Environment” in the Simulink Coder documentation.

There are a few Computer Vision System Toolbox blocks that generate code with limited portability. These blocks use precompiled shared libraries, such as DLLs, to support I/O for specific types of devices and file formats. To find out which blocks use precompiled shared libraries, open the Computer Vision System Toolbox Block Support Table. You can identify blocks that use precompiled shared libraries by checking the footnotes listed in the **Code Generation Support** column of the table. All blocks that use shared libraries have the following footnote:

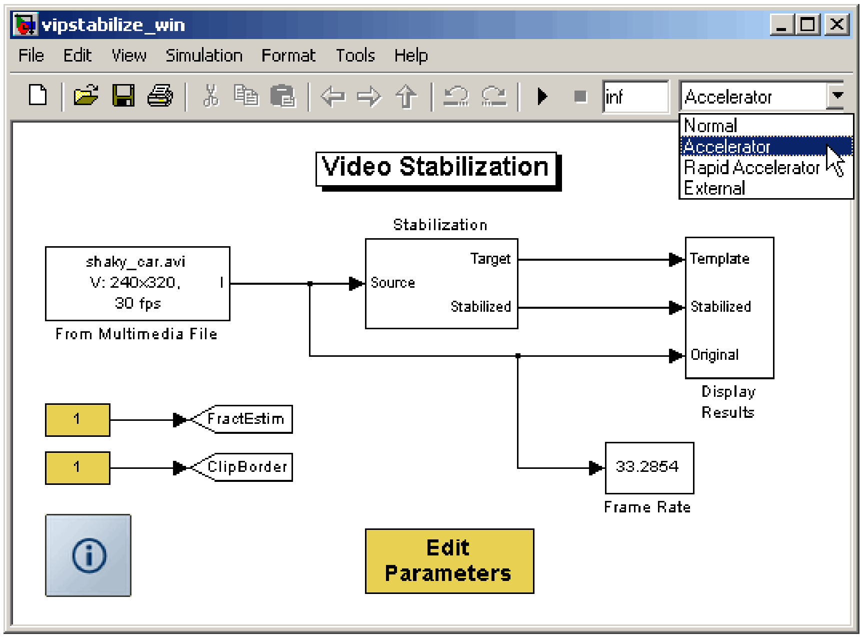
```
Host computer only.  Excludes Real-Time Windows (RTWIN) target.
```

Simulink Coder provides functions to help you set up and manage the build information for your models. For example, one of the “Build Information ” functions that Simulink Coder provides is `getNonBuildFiles`. This function allows you to identify the shared libraries required by blocks in your model. If your model contains any blocks that use precompiled shared libraries, you can install those libraries on the target system. The folder that you install the shared libraries in must be on the system path. The target system does not need to have MATLAB installed, but it does need to be supported by MATLAB.

Accelerating Simulink Models

The Simulink software offer `Accelerator` and `Rapid Accelerator` simulation modes that remove much of the computational overhead required by Simulink models. These modes compile target code of your model. Through this method, the Simulink environment can achieve substantial performance improvements for larger models. The performance gains are tied to the size and complexity of your model. Therefore, large models that contain Computer Vision System Toolbox blocks run faster in `Rapid Accelerator` or `Accelerator` mode.

To change between `Rapid Accelerator`, `Accelerator`, and `Normal` mode, use the drop-down list at the top of the model window.



For more information on the accelerator modes in Simulink, see "Accelerating Models" in the Simulink User's Guide.

Index