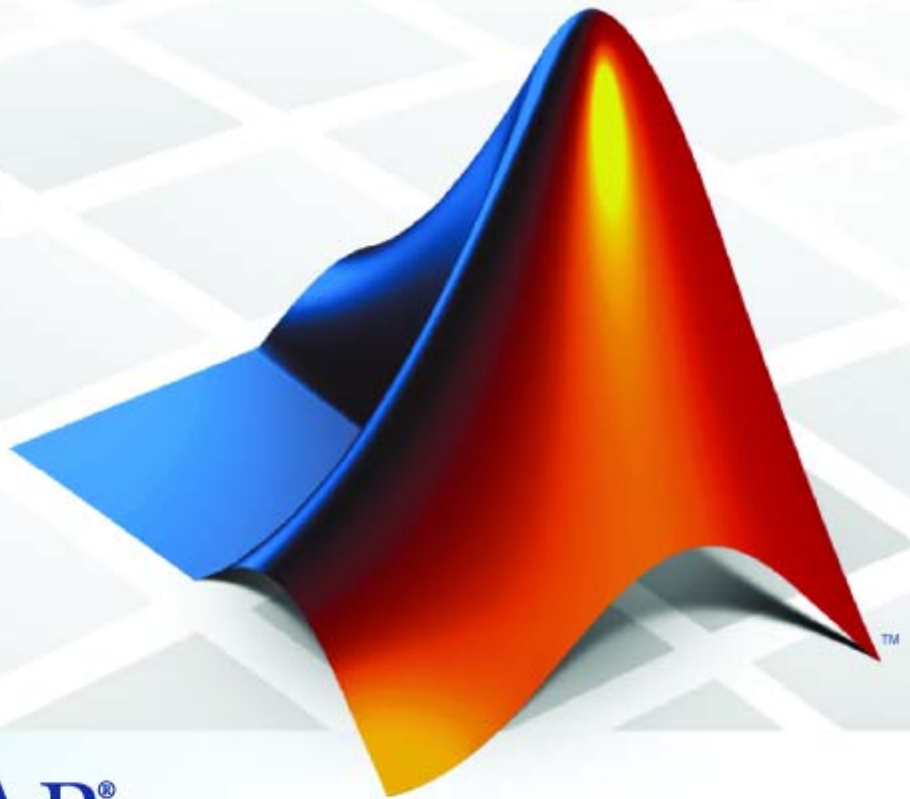


Video and Image Processing Blockset™ 3

User's Guide



MATLAB®
& **SIMULINK®**

How to Contact The 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.

Video and Image Processing Blockset™ User's Guide

© COPYRIGHT 2004–2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

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)

Getting Started

1

Product Overview	1-2
Installation	1-3
Installing the Video and Image Processing Blockset	
Software	1-3
Required Products	1-4
Related Products	1-4
Product Demos	1-5
Demos in the Help Browser	1-5
Demos on the Web	1-9
Demos on MATLAB Central	1-10
Working with the Documentation	1-11
Viewing the Documentation	1-11
Printing the Documentation	1-12
Using This Guide	1-12
Key Blockset Concepts	1-15
Image Types	1-15
Video in the Video and Image Processing Blockset	
Blocks	1-16
Defining Intensity and Color	1-16
Color Image Processing	1-17
Coordinate Systems	1-24
Image Data Stored in Column-Major Format	1-26
Sample Time	1-26
Video Duration and Simulation Time	1-27
Acceleration Modes	1-28
Strategies for Real-Time Video Processing	1-29
Code Generation	1-31
Block Data Type Support	1-33

Image Credits	1-34
----------------------------	-------------

Importing and Exporting Images and Video

2

Batch Processing Image Files	2-2
Working with Live Video	2-7
Working with Multimedia Files	2-8
Blocks That Support Multimedia Files	2-8
Importing and Viewing Multimedia Files	2-8
Exporting to Multimedia Files	2-11
Working with AVI Files	2-14
Working with Audio	2-38
Working with MATLAB Workspace Variables	2-43
How to Import MATLAB Workspace Variables	2-43

Viewing Video

3

Viewing Video Files	3-2
Viewing Video Signals in Simulink	3-3
Using the Video Viewer Block	3-3
Using the To Video Display Block	3-3
Using the MPlay GUI	3-3
Viewing Video File Frames	3-22

4

Feature Extraction	4-2
Finding Edges in Images	4-2
Finding Lines in Images	4-9
Measuring an Angle Between Lines	4-18
Image Enhancement	4-30
Sharpening and Blurring an Image	4-30
Removing Salt and Pepper Noise from Images	4-39
Removing Periodic Noise from Video	4-45
Adjusting the Contrast in Intensity Images	4-54
Adjusting the Contrast in Color Images	4-61
Template Matching	4-67
Using the Template Matching Block	4-67
Video Stabilization	4-71
Panorama Creation	4-72
Pixel Statistics	4-73
Blocks That Compute Pixel Statistics	4-73
Finding the Histogram of an Image	4-73

Conversions

5

Intensity to Binary Conversion	5-2
Overview of Intensity and Binary Images	5-2
Thresholding Intensity Images Using Relational Operators	5-2
Thresholding Intensity Images Using the Autothreshold Block	5-7
Color Space Conversion	5-14
Overview of Color Space Conversion Block	5-14
Converting Color Information from R'G'B' to Intensity ...	5-14

Chroma Resampling	5-19
--------------------------------	-------------

Geometric Transformation

6

Geometric Transformation Interpolation Methods	6-2
Overview of Interpolation Methods	6-2
Nearest Neighbor Interpolation	6-2
Bilinear Interpolation	6-3
Bicubic Interpolation	6-4
Rotating an Image	6-6
Resizing an Image	6-14
Cropping an Image	6-20

Morphological Operations

7

Overview of Morphology	7-2
Counting Objects in an Image	7-3
Correcting for Nonuniform Illumination	7-10

Example Applications

8

Pattern Matching	8-2
Overview of Pattern Matching	8-2

Tracking an Object Using Correlation	8-2
Motion Compensation	8-10
Image Compression	8-12
Overview of Image Compression	8-12
Compressing an Image	8-12
Viewing the Compressed Image	8-18

Getting Started with System Objects

9

What Are System Objects?	9-2
Setting Up and Running System Objects	9-3
Creating an Instance of a System Object	9-3
Using Methods to Run System Objects	9-6
Finding Help and Demos for System Objects	9-8
Using System Objects with the Embedded MATLAB Subset	9-9
Considerations for Using System Objects with the Embedded MATLAB Subset	9-9
Using System Objects with Embedded MATLAB Coder ..	9-11
Using System Objects with the Embedded MATLAB Function Block	9-12
Using System Objects with Embedded MATLAB MEX ...	9-12

Using Video and Image Processing System Objects

10

What Are Video and Image Processing System Objects?	10-2
--	-------------

Generating Code for Video and Image Processing	
System Objects	10-3
Working with Fixed-Point Data	10-5
Working with Fixed-Point Data	10-5
Example: Using System Objects in Video and Image	
Processing Applications: Marking a Region of	
Interest	10-9

Index

Getting Started

The Video and Image Processing Blockset™ software is a tool for processing images and video in the Simulink® environment. This chapter provides an introduction to the Video and Image Processing Blockset software, its product requirements, and its documentation.

- “Product Overview” on page 1-2
- “Installation” on page 1-3
- “Product Demos” on page 1-5
- “Working with the Documentation” on page 1-11
- “Key Blockset Concepts” on page 1-15
- “Block Data Type Support” on page 1-33
- “Image Credits” on page 1-34

Product Overview

Video and Image Processing Blockset provides algorithms and tools for the design and simulation of video processing, image processing, and computer vision systems. You can process video and image data to solve problems such as noise, low contrast, out-of-focus optics, and artifacts resulting from interlaced video. You can then perform tasks such as motion analysis, object detection and tracking, video stabilization, and disparity estimation for stereo vision. Most algorithms and tools are available as both System objects (for use in MATLAB®) and blocks (for use in Simulink®).

Tools for multimedia file I/O, video display, drawing graphics, and compositing enable you to visualize, simulate, and evaluate design alternatives. For embedded system design and rapid prototyping, the blockset supports fixed-point arithmetic, C-code generation, and implementation on embedded hardware.

Installation

In this section...

“Installing the Video and Image Processing Blockset Software” on page 1-3

“Required Products” on page 1-4

“Related Products” on page 1-4

Installing the Video and Image Processing Blockset Software

Before you begin working with the Video and Image Processing Blockset software, you need to install the product on your computer.

Installation from a DVD

Video and Image Processing Blockset blocks follow the same installation procedure as the MATLAB® toolboxes:

- 1 Start the MathWorks™ installer.
- 2 When prompted, select the **Product** check boxes for the products you want to install.

The documentation is installed along with the products.

Installation from a Web Download

You can use your MathWorks Account to download products from the MathWorks Web site:

- 1 Navigate to http://www.mathworks.com/web_downloads/.
- 2 Click **Download products**.
- 3 Log in to the system using your MathWorks Account e-mail and password. If you do not have a MathWorks Account, you can create one from this Web page.
- 4 Select your platform and the products you want to install.

- 5 Follow the instructions on the **Download and Install** screen, which describe how to download the product(s) and the installer.
- 6 Double-click the `Installer.exe` file to run the installer.
- 7 When prompted, enter your Personal License Password.
- 8 Select the **Product** check boxes for the products you want to install.

The documentation is installed along with the products.

Required Products

The Video and Image Processing Blockset software is part of a family of products from The MathWorks™. You need to install several products to use the Video and Image Processing Blockset software. For more information, see the MathWorks Web site at <http://www.mathworks.com/products/viprocessing/requirements.jsp>.

Related Products

The MathWorks provide several products that are relevant to the kinds of tasks you can perform with the Video and Image Processing Blockset software.

For more information about any of these products, see either

- The online documentation for that product if it is installed on your system
- The MathWorks Web site, at <http://www.mathworks.com/products/viprocessing/related.jsp>

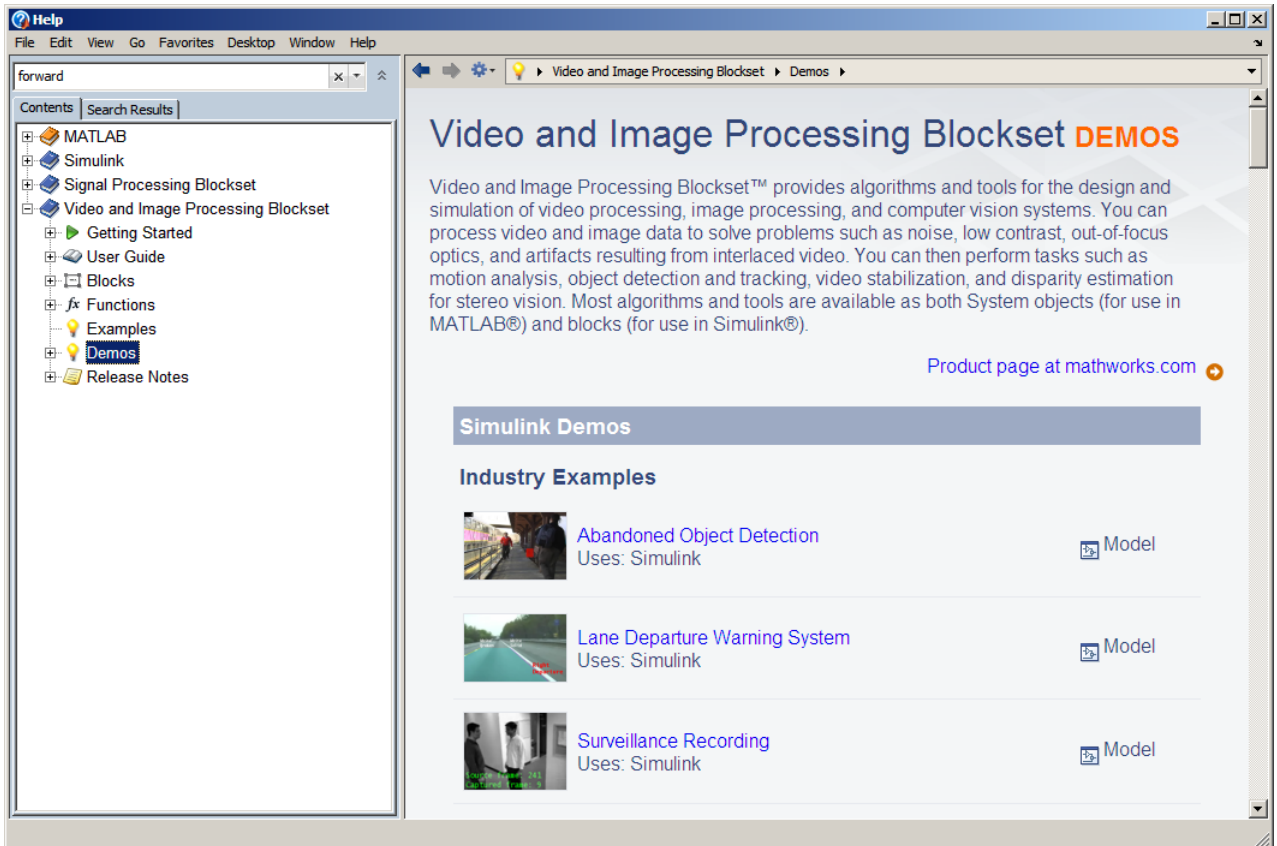
Product Demos

In this section...
“Demos in the Help Browser” on page 1-5
“Demos on the Web” on page 1-9
“Demos on MATLAB Central” on page 1-10

Demos in the Help Browser

You can find interactive Video and Image Processing Blockset demos in the MATLAB Help browser. This example shows you how to locate and open a typical demo:

- 1** To open the Help browser to the **Demos** tab, type `doc` at the MATLAB command line.
- 2** Expand the **Video and Image Processing** node in the Help browser, then the **Demos** node.

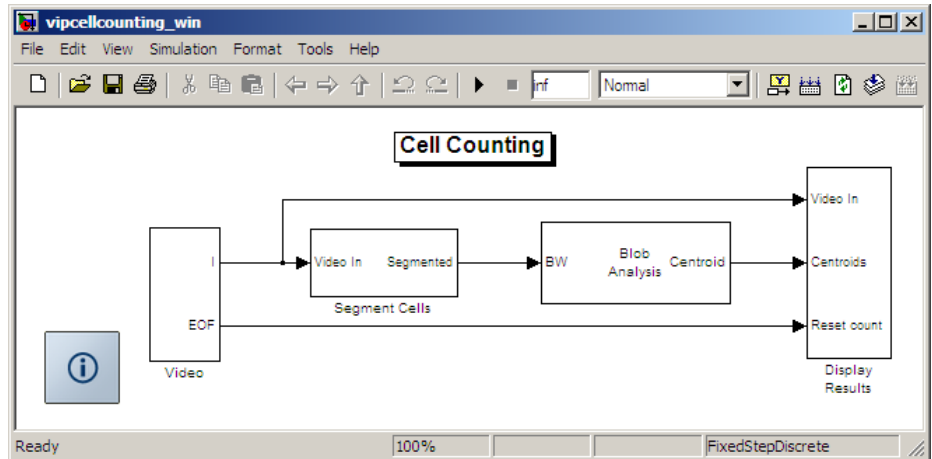


There are two entries under the Video and Image Processing Blockset **Demos** node:

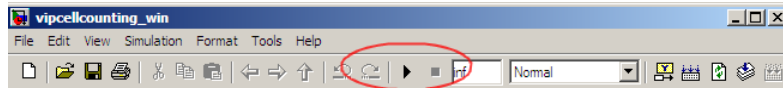
- **Simulink Demos** — Expand this entry to see a categorical list of block-based Video and Image Processing Blockset demos.
 - **MATLAB Demos** — Expand this entry to see a categorical list of Video and Image Processing Blockset System object demos.
- 3** To view the description of the Simulink-based Cell Counting demo, which demonstrates how to extract information from a video stream, expand the **Simulink Demos** and **Analysis**, and then click **Cell Counting**.

The screenshot shows the MATLAB Help interface. On the left, a navigation tree lists various topics, with 'Cell Counting' highlighted under 'Analysis' in the 'Simulink Demos' section. The main content area displays the 'vipcellcounting' demo page. The page title is 'vipcellcounting' with a link to 'Open this model'. The main heading is 'Cell Counting'. Below the heading is a paragraph describing the demo: 'This demo illustrates how to use a combination of basic morphological operators and blob analysis to extract information from a video stream. In this case, the demo counts the number of E. Coli bacteria in each video frame. Note that the cells are of varying brightness, which makes the task of segmentation more challenging.' Below this is a 'Contents' section with links to 'Demo Model', 'Segment Cells Subsystem', 'Cell Counting Results', 'Data Set Credits', and 'Available Demo Versions'. A 'Demo Model' section follows, stating 'The following figure shows the Cell Counting demo model.' Below the text is a block diagram of the 'Cell Counting' model. The diagram shows a 'Video' input block connected to a 'Segment Cells' block. The 'Segment Cells' block outputs 'Video In Segmented' to a 'Blob Analysis Centroid' block. The 'Blob Analysis Centroid' block outputs 'Centroids' and 'Reset count' to a 'Display Results' block. There is also a feedback loop from 'Display Results' back to 'Segment Cells'.

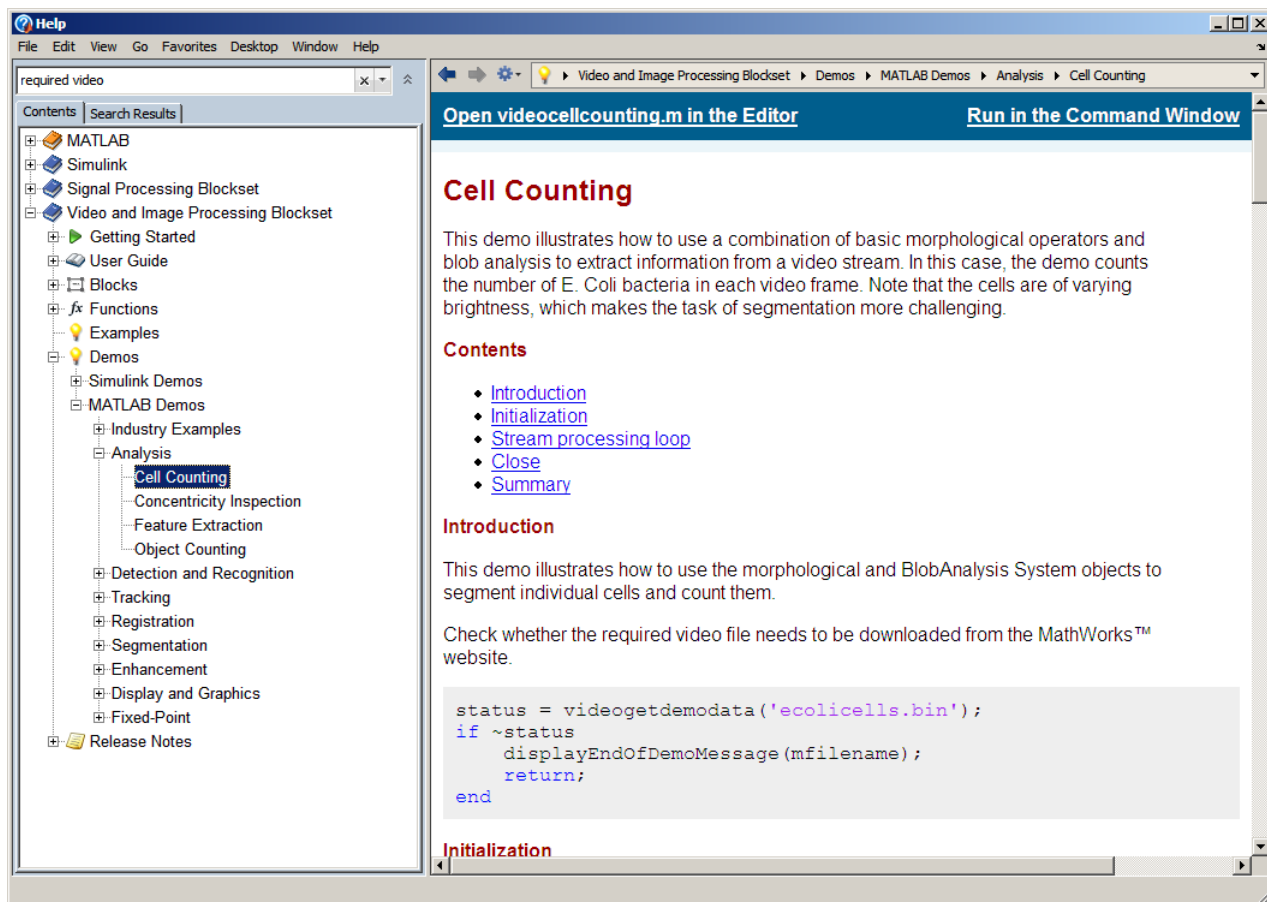
- a Click **Open this model** to display the Simulink model for Cell Counting demo. The model window opens.




- b** Run the model by selecting **Start** from the **Simulation** menu in the model window, or by clicking the start button from the toolbar.



- c** The Cell Counting demo analyzes a binary video file. You will be prompted to download the file, if it is unavailable in your folder system.
 - d** The results of the demo appear in the **Results** graphical window.
- 4** To view a version of the Cell Counting demo that uses System objects in MATLAB, expand the **MATLAB Demos** and **Analysis**, and then click **Cell Counting**



- a Click **Open videocellcounting.m in the Editor** to display the System object Cell Counting demo in the MATLAB editor.
- b Run the demo by clicking the **Run** toolbar button . The results of the demo appear in the **Results** graphical window.

Demos on the Web

The MathWorks Web site contains demos that show you how to use the Video and Image Processing Blockset software. You can find these demos at <http://www.mathworks.com/products/viprocessing/demos.jsp>.

You can run these demos without having MATLAB or Video and Image Processing Blockset software installed on your system.

Demos on MATLAB Central

The MATLAB Central website contains files, including demos, contributed by users and developers of Video and Image Processing Blockset software, MATLAB, Simulink and other products. Contributors submit their files to one of a list of categories. You can browse these categories to find submissions that pertain to Video and Image Processing Blockset software or a specific problem that you would like to solve. MATLAB Central is located at <http://www.mathworks.com/matlabcentral/>.

Working with the Documentation

In this section...
“Viewing the Documentation” on page 1-11
“Printing the Documentation” on page 1-12
“Using This Guide” on page 1-12

Viewing the Documentation

You can access the Video and Image Processing Blockset documentation using files you installed on your system or from the Web using the MathWorks Web site.

Documentation in the Help Browser

This procedure shows you how to use the MATLAB Help browser to view the Video and Image Processing Blockset documentation installed on your system:

- 1** In the MATLAB window, from the **Help** menu, click **Product Help**. The Help browser opens.
- 2** From the list of products in the left pane, click **Video and Image Processing Blockset**. In the right pane, the Help browser displays the Video and Image Processing Blockset Roadmap page.
- 3** Under the section titled **Documentation Set**, select **User’s Guide**. The Help browser displays the chapters of this manual.

The Help browser also has a **Demos** tab where you can view product demos. For more information, see “Product Demos” on page 1-5.

Documentation on the Web

You can also view the documentation from the MathWorks Web site. The documentation available on these Web pages is for the latest release, regardless of whether the release was distributed on a DVD or as a Web download:

- 1 Navigate to the Video and Image Processing Blockset Product page at <http://www.mathworks.com/products/viprocessing/>.
- 2 On the right side of the page, click the **Documentation** link. The Video and Image Processing Blockset documentation is displayed.

Printing the Documentation

The documentation for the Video and Image Processing Blockset software is also available in printable PDF format. You need to install Adobe Acrobat Reader 4.0 or later to open and read these files. To download a free copy of Acrobat Reader, see <http://www.adobe.com/products/acrobat/main.html>.

The following procedure shows you how to view the documentation in PDF format:

- 1 In the MATLAB window, from the **Help** menu, click **Product Help**. The Help browser opens.
- 2 From the list of products in the left pane, click **Video and Image Processing Blockset**. In the right pane, the Help browser displays the Video and Image Processing Blockset Roadmap page.
- 3 Under the **Printing the Documentation Set** heading, click the links to view PDF versions of the Video and Image Processing Blockset documentation.

Using This Guide

To help you effectively read and use this guide, here is a brief description of the chapters and a suggested reading path.

Expected Background

This manual assumes that you are familiar with the following:

- The MATLAB language, to write scripts and functions with MATLAB code, and to use functions with the command-line interface
- The Simulink environment, to create simple models as block diagrams and simulate those models

What Chapter Should I Read?

Follow the procedures in this guide to become familiar with the blockset's functionality. The User's Guide contains tutorial sections that are designed to help you become familiar with using the Simulink and Video and Image Processing Blockset software:

- Read Chapter 1, “Getting Started” to learn about the installation process, the products required to run Video and Image Processing Blockset blocks, and to view the Video and Image Processing Blockset demos.
- Read Chapter 2, “Importing and Exporting Images and Video” to understand how video is interpreted by the Simulink blocks. Also, learn how to bring video data into a model, display it on your monitor, and export it to an AVI file.
- Read Chapter 3, “Viewing Video” to learn how to use the MPlay GUI to view videos that are represented as variables in the MATLAB workspace. You can also learn how to use it to view video files or video signals in Simulink models.
- Read Chapter 5, “Conversions” to learn how to convert an intensity image to a binary image, how to convert color information between color spaces, and how to downsample the chroma components of an image.
- Read Chapter 6, “Geometric Transformation” to understand how blocks in the Geometric Transformations library interpolate values. You also learn how to rotate, resize, and crop images.
- Read Chapter 7, “Morphological Operations” to learn about morphological operations and which blocks can be used to perform them. For example, you learn how to count objects in an image and correct for nonuniform illumination.
- Read Chapter 4, “Analysis and Enhancement” to learn how to sharpen, blur, and remove noise from images. You also learn how to find object boundaries and calculate the histogram of the R, G, and B values in an image.
- Read Chapter 8, “Example Applications” to learn how to track the motion of an object in a video stream. Also, learn more about motion compensation and image compression.

For a description of each block's operation, parameters, and characteristics, see the Block Reference in the Video and Image Processing Blockset documentation on the Web at <http://www.mathworks.com/products/viprocessing/> or in the Help browser.

Key Blockset Concepts

In this section...

“Image Types” on page 1-15

“Video in the Video and Image Processing Blockset Blocks” on page 1-16

“Defining Intensity and Color” on page 1-16

“Color Image Processing” on page 1-17

“Coordinate Systems” on page 1-24

“Image Data Stored in Column-Major Format” on page 1-26

“Sample Time” on page 1-26

“Video Duration and Simulation Time” on page 1-27

“Acceleration Modes” on page 1-28

“Strategies for Real-Time Video Processing” on page 1-29

“Code Generation” on page 1-31

Image Types

In the Video and Image Processing Blockset 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 Video and Image Processing Blockset 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 Video and Image Processing Blockset 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.

Video in the Video and Image Processing Blockset Blocks

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

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

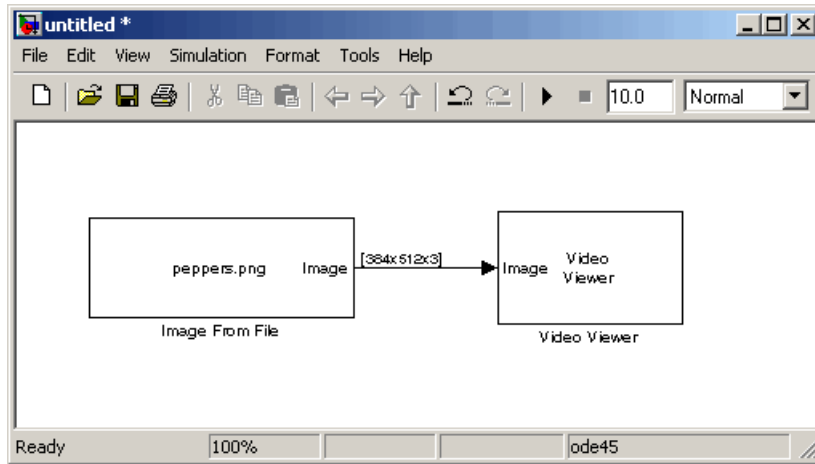
Note The Video and Image Processing Blockset software considers any data type other than double-precision floating point and single-precision floating point to be fixed point.

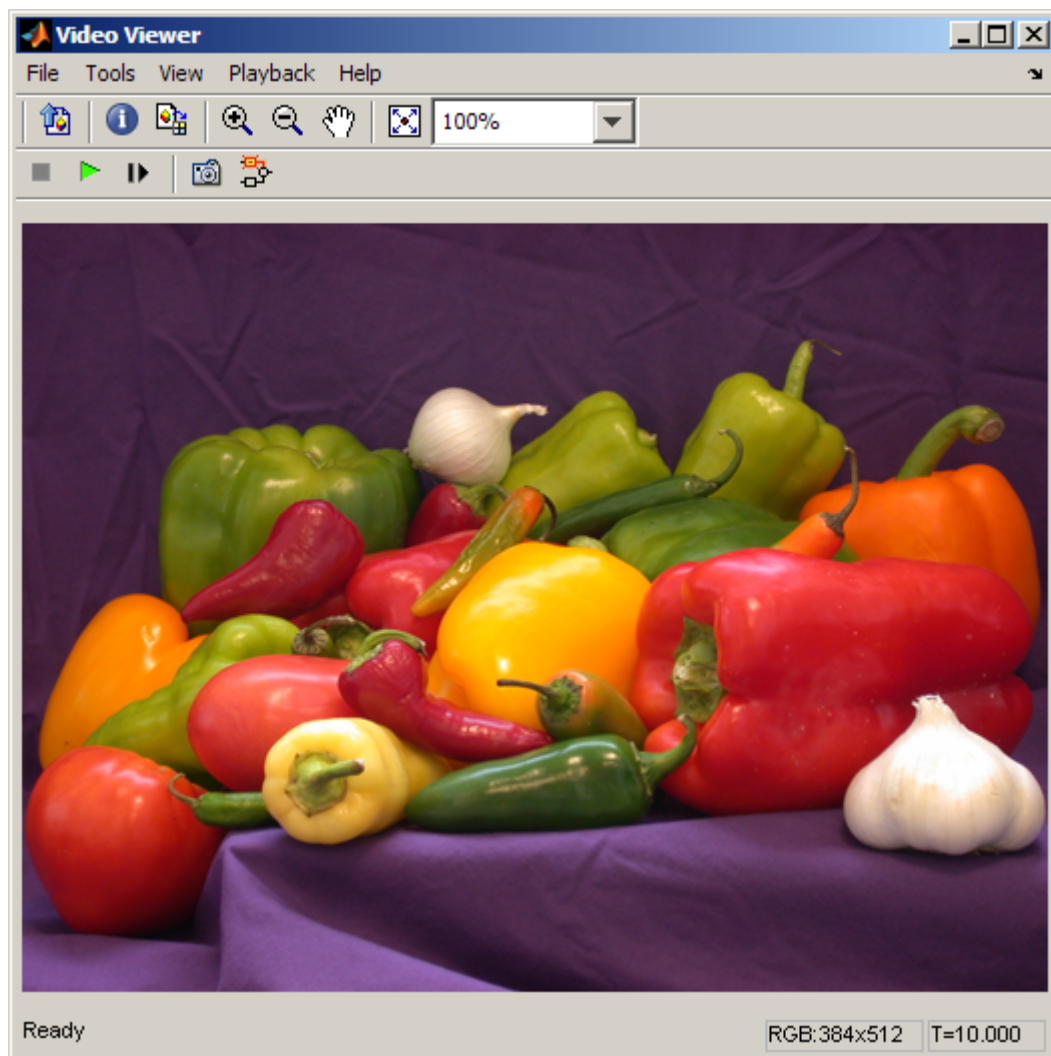
For example, for an intensity image whose image values are 8-bit unsigned integers, 0 is black and 255 is white. For an intensity image whose image values are double-precision floating point, 0 is black and 1 is white. For an intensity image whose image values are 16-bit signed integers, -32768 is black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

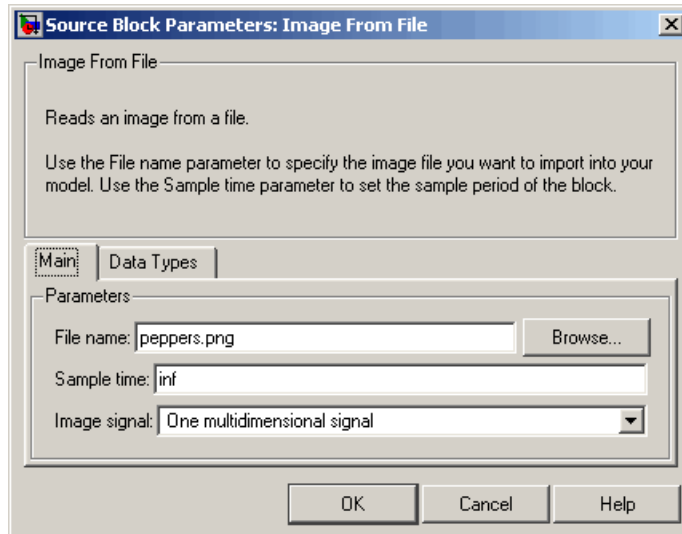
Color Image Processing

The Video and Image Processing Blockset 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.





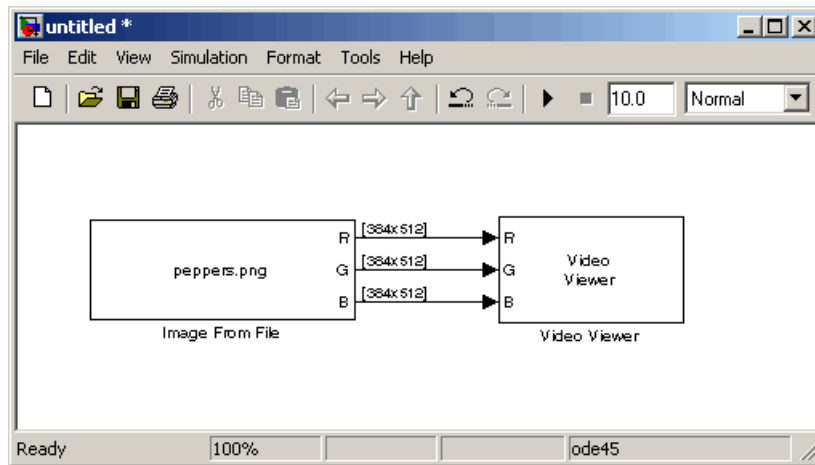
You can choose to process the image as a multidimensional array by setting the **Image signal** parameter to `One multidimensional signal` in the Image From File block dialog box.

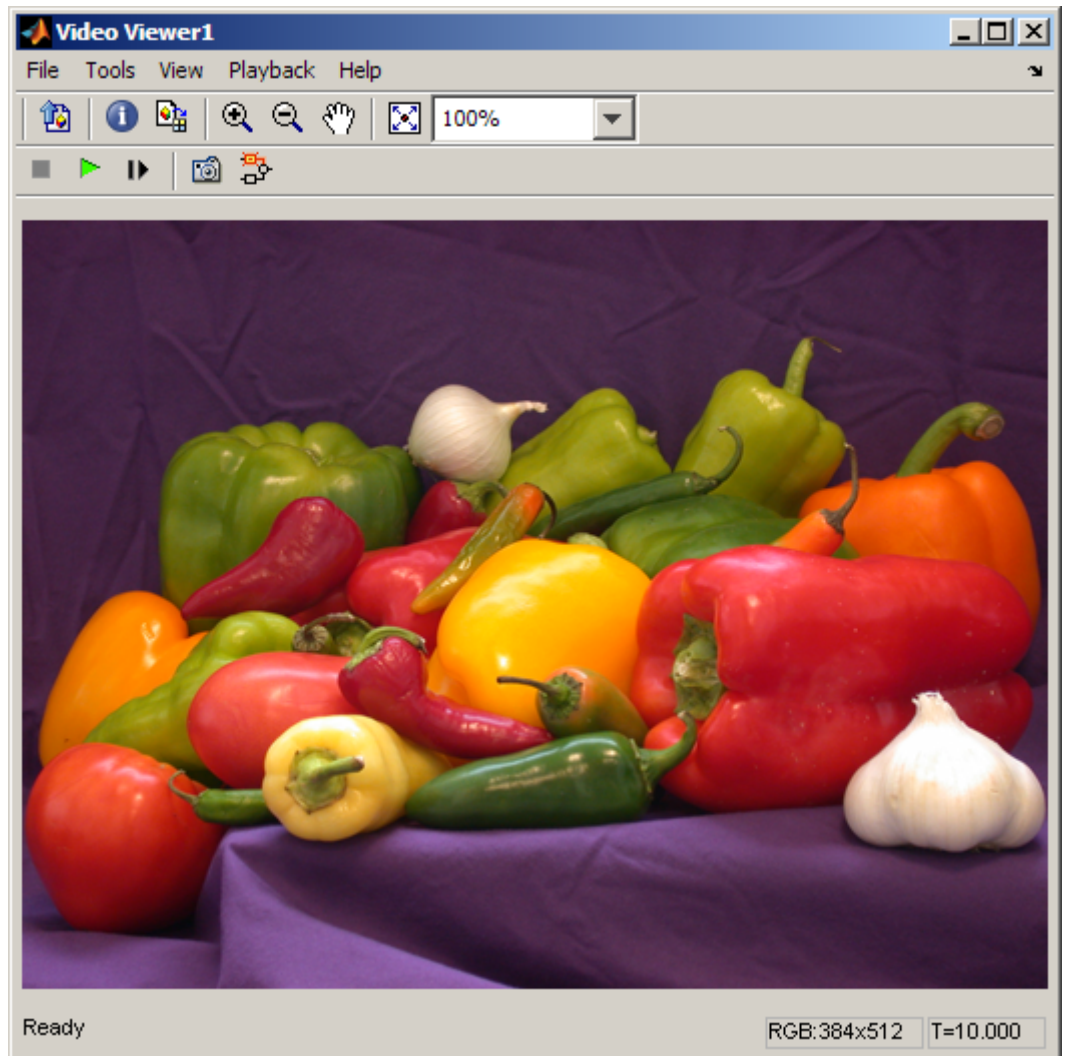


The blocks that support multidimensional arrays meet at least one of the following criteria:

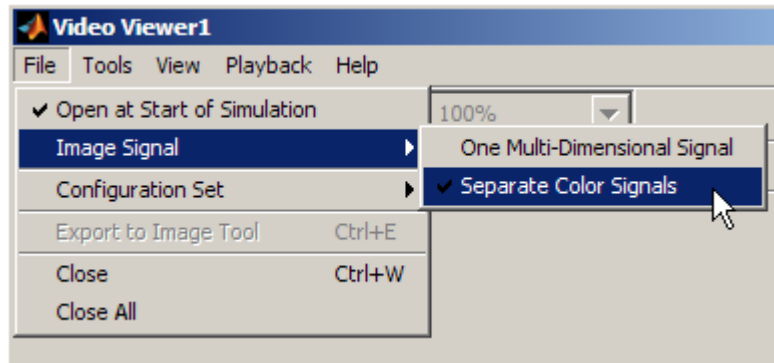
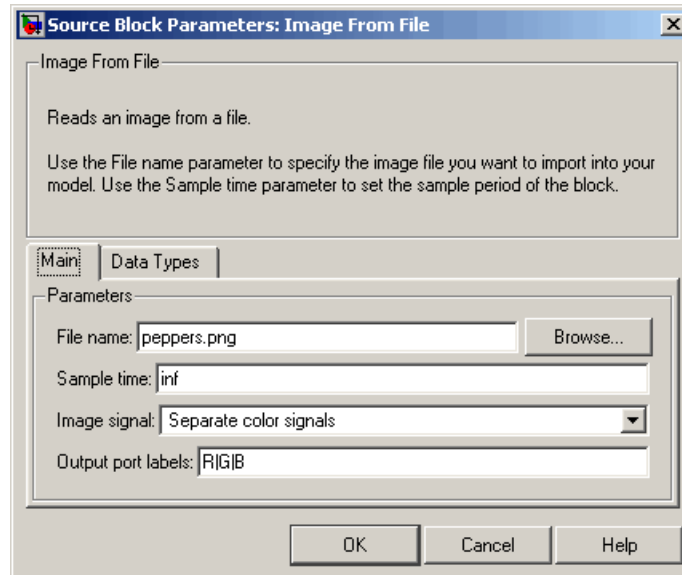
- They have the **Image signal** parameter on their block mask.
- They have a note in their block reference pages that says, “This block supports intensity and color images on its ports.”
- Their input and output ports are labeled “Image”.

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.





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.



Note The ability to output separate color signals is a legacy option. It is recommended 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 “Measuring an Angle Between Lines” on page 4-18. If you are

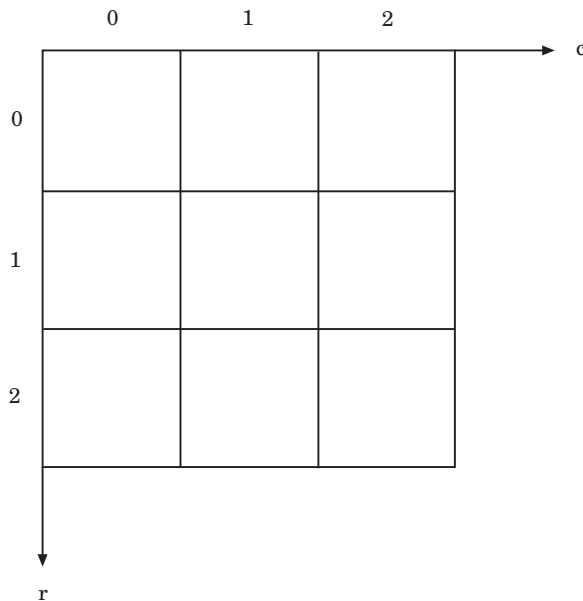
working with a block that only accepts multidimensional arrays, you can use the Matrix Concatenation block to create a multidimensional array. For an example of this process, see “Finding the Histogram of an Image” on page 4-73.

Coordinate Systems

You can specify locations in images using various coordinate systems. This topic discusses pixel coordinates and spatial coordinates, which are the two main coordinate systems used in the Video and Image Processing Blockset software.

Pixel Coordinates

Pixel coordinates enable you to specify locations in images. In this coordinate system, the image is treated as a grid of discrete elements, ordered from top to bottom and left to right, as shown in the following figure:

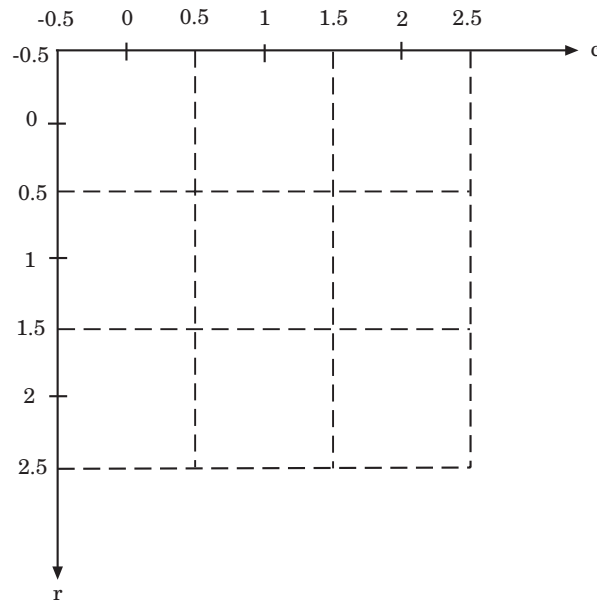


For pixel coordinates, the first component r (the row) increases downward, while the second component c (the column) increases to the right. Pixel coordinates are integer values and range from 0 to the length of the row or

column. The pixel coordinates used in Video and Image Processing Blockset software are zero based, while the pixel coordinates used by Image Processing Toolbox and MATLAB are one based. For more information on the pixel coordinate system used by Image Processing Toolbox, see “Pixel Coordinates” in the Image Processing Toolbox documentation.

Spatial Coordinates

Spatial coordinates enable you to specify a location in an image with greater granularity than pixel coordinates. For example, in the pixel coordinate system, a pixel is treated as a discrete unit, uniquely identified by an integer row and column pair, such as (3,4). In a spatial coordinate system, locations in an image can be represented in terms of partial pixels, such as (3.3, 4.7). The following figure illustrates the spatial coordinate system used for images:



This spatial coordinate system corresponds to the pixel coordinate system in the following ways. First, both are defined in terms of row and column positions. Second, the spatial coordinates of the center point of any pixel are identical to the pixel coordinates for that pixel. However, the pixel coordinate system is discrete, while the spatial coordinate system is continuous. This

means that, in pixel coordinates, the upper-left corner of an image is (0,0), while in spatial coordinates, this location is (-0.5,-0.5). The spatial coordinate system used by the Video and Image Processing Blockset software differs from the one used by Image Processing Toolbox. For more information on this spatial coordinate system, see “Spatial Coordinates” in the Image Processing Toolbox documentation.

Image Data Stored in Column-Major Format

The MATLAB technical computing software and Video and Image Processing Blockset 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 Video and Image Processing Blockset 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.

Sample Time

Because the Video and Image Processing blocks calculate values directly rather than solving differential equations, you must configure the Simulink Solver to behave like a scheduler that uses each block’s sample time to determine when the code behind the block is executed. The following steps show you how to do this:

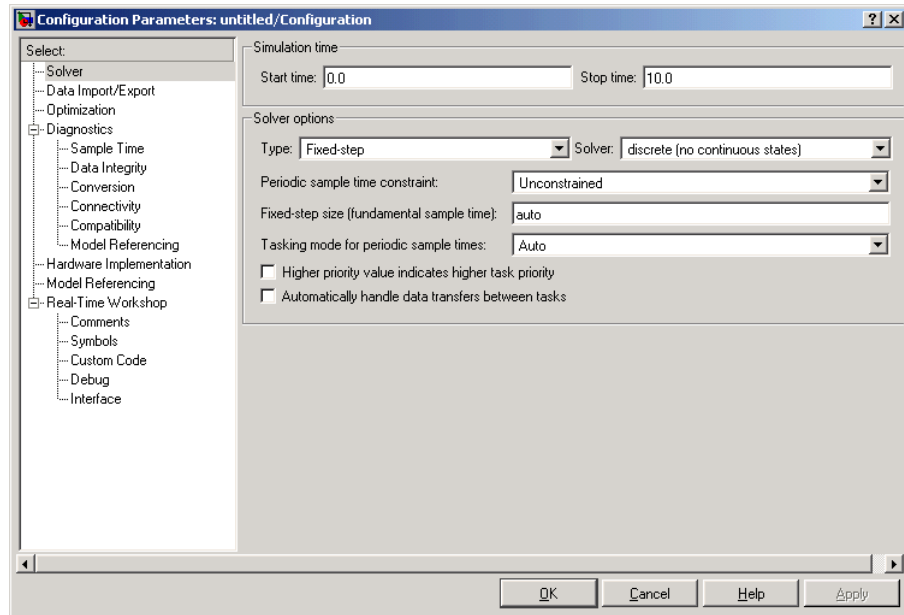
- 1** From the model’s **Simulation** menu, select **Configuration Parameters**.

The Configuration dialog box opens.

- 2** From the **Type** list, choose **Fixed-step**.

- 3** From the **Solver** list, choose **Discrete (no continuous states)**.

The following figure shows the correctly configured Configuration dialog box.

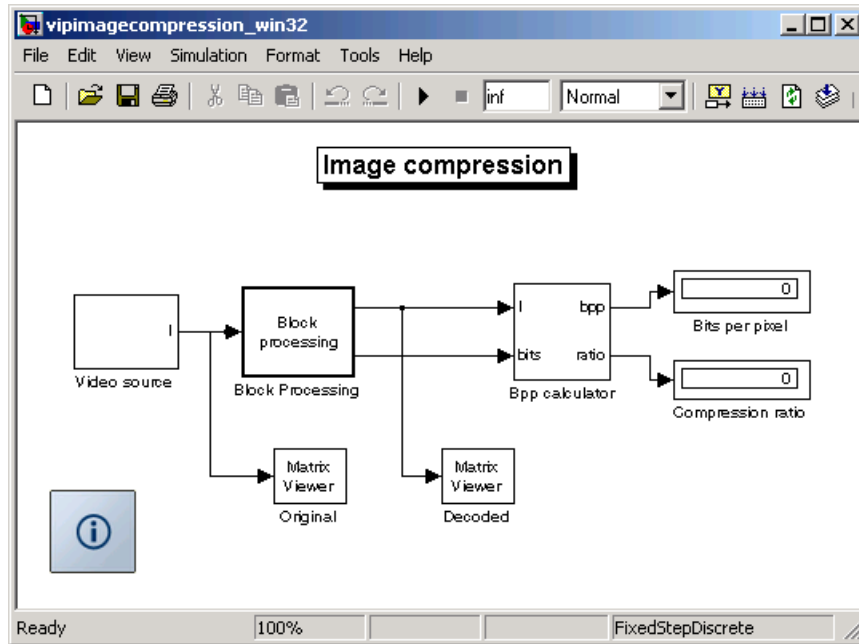


The Solver, while in scheduler mode, uses a block's sample time to determine when the code behind each block is executed. For example, if the sample time of a Video From Workspace block is 0.05, the Solver executes the code behind this block, and every other block with this sample time, once every 0.05 second.

Video Duration and Simulation Time

The duration of the simulation is controlled by the **Stop time** parameter — not the input video. If you want the simulation to run for the duration of the input video, you must adjust the **Stop time** parameter. If your video is being cropped, increase the parameter value. If your video is complete and the display window is black, decrease the parameter value. To view the first N frames of your video, set the **Stop time** parameter to $(N-1)*T_s$, where T_s is the sample time of your source block.

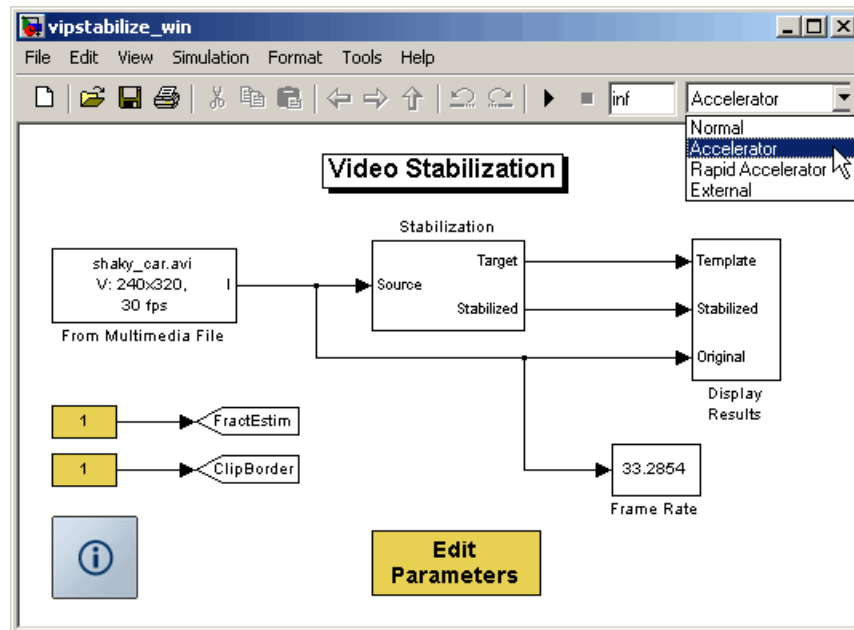
You can access the **Stop time** parameter in the model window, as shown in the following figure, or on the **Solver** pane of the Configuration dialog box.



Acceleration Modes

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 Video and Image Processing Blockset 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.

Strategies for Real-Time Video Processing

Video processing is computationally intensive, and the ability to perform real-time video processing is affected by the following factors:

- Hardware capability
- Model complexity
- Model implementation
- Input data size

Optimizing Your Implementation

Optimizing your implementation is a crucial step toward real-time video processing. The following tips can help improve the performance of your model:

- Minimize the number of blocks in your model.
- Process only the regions of interest to reduce the input data size.
- Use efficient algorithms or the simplest version of an algorithm that achieves the desired result.
- Use efficient block parameter settings. However, you need to decide whether these settings best suit your algorithm. For example, the most efficient block parameter settings might not yield the most accurate results. You can find out more about individual block parameters and their effect on performance by reviewing specific block reference pages.

The two following examples show settings that make each block's operation the least computationally expensive:

- Resize block — **Interpolation method** = Nearest neighbor
- Blocks that support fixed point — On the **Fixed-Point** tab, **Overflow mode** = Wrap
- Choose data types carefully.
 - Avoid data type conversions.
 - Use the smallest data type necessary to represent your data to reduce memory usage and accelerate data processing.

In simulation mode, models with floating-point data types run faster than models with fixed-point data types. To speed up fixed-point models, you must run them in accelerator mode. Simulink contains additional code to process all fixed-point data types. This code affects simulation performance. After you run your model in accelerator mode or generate code for your target using Real-Time Workshop, the fixed-point data types are specific to the choices you made for the fixed-point parameters. Therefore, the fixed-point model and generated code run faster.

Developing Your Models

Use the following general process guidelines to develop real-time video processing models to run on embedded targets. By optimizing the model at each step, you improve its final performance.

- 1** Create the initial model and optimize the implementation algorithm. Use floating-point data types so that the model runs faster in simulation mode. If you are working with a floating-point processor, go to step 3.
- 2** If you are working with a fixed-point processor, gradually change the model data types to fixed point, and run the model after every modification.

During this process, you can use data type conversion blocks to isolate the floating point sections of the model from the fixed-point sections. You should see a performance improvement if you run the model in accelerator mode.

- 3** Remove unnecessary sink blocks, including scopes, and blocks that log data to files.
- 4** Compile the model for deployment on the embedded target.

Code Generation

The Video and Image Processing Blockset, Real-Time Workshop®, and Real-Time Workshop® Embedded Coder™ software enable you to generate code that you can use to implement your model for a practical application. For instance, you can create an executable from your Simulink model to run on a target chip. For more information, see “Understanding Code Generation” in *Signal Processing Blockset Getting Started Guide*.

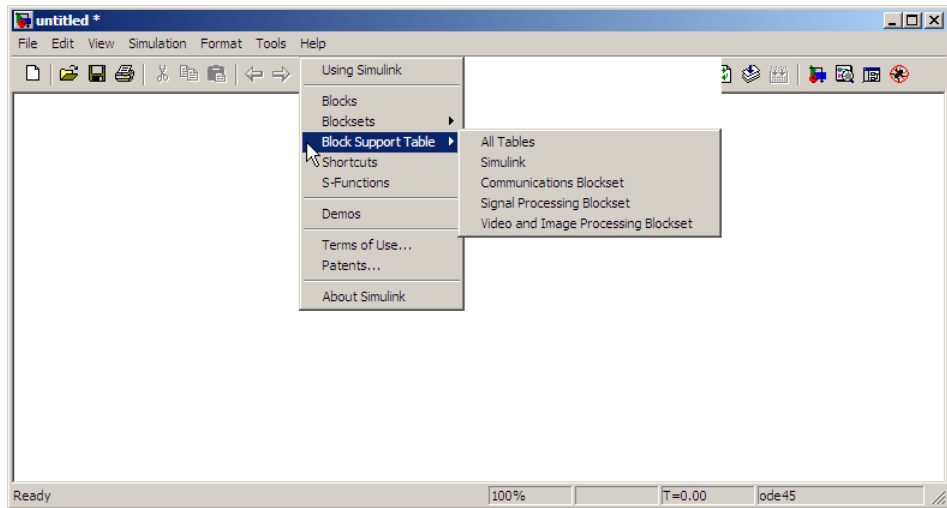
Shared Library Dependencies

For the blocks listed in the table below, copy the shared library files from the machine where the blockset software is installed to a folder on the system path of the destination machine.

Block	Dependent library file	Product
To Multimedia File	tommfile.dll SldirectShow.dll	Signal Processing Blockset™
From Multimedia File	frommmfile.dll SldirectShow.dll	Signal Processing Blockset
To Video Display	tovideodevice.dll SldirectShow.dll	Video and Image Processing Blockset

Block Data Type Support

The Video and Image Processing Blockset Data Type Support Table is now available through the Simulink model Help menu. The table provides information about data type support and code generation coverage for all Video and Image Processing Blockset blocks. Select **Help > Block Support Table > Video and Image Processing Blockset** or **Help > Block Support Table > All Tables**.



You can also type `showvipblockdatatypetable` at the MATLAB command line to bring up the table.

Image Credits

This table lists the copyright owners of the images used in the Video and Image Processing Blockset documentation.

Image	Source
cameraman	Copyright Massachusetts Institute of Technology. Used with permission.
cell	Cancer cell from a rat's prostate, courtesy of Alan W. Partin, M.D., Ph.D., Johns Hopkins University School of Medicine.
circuit	Micrograph of 16-bit A/D converter circuit, courtesy of Steve Decker and Shujaat Nadeem, MIT, 1993.
moon	Copyright Michael Myers. Used with permission.

Importing and Exporting Images and Video

- “Batch Processing Image Files” on page 2-2
- “Working with Live Video” on page 2-7
- “Working with Multimedia Files” on page 2-8
- “Working with MATLAB Workspace Variables” on page 2-43

Batch Processing 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 and display the sequence using the Video and Image Processing Blockset software.

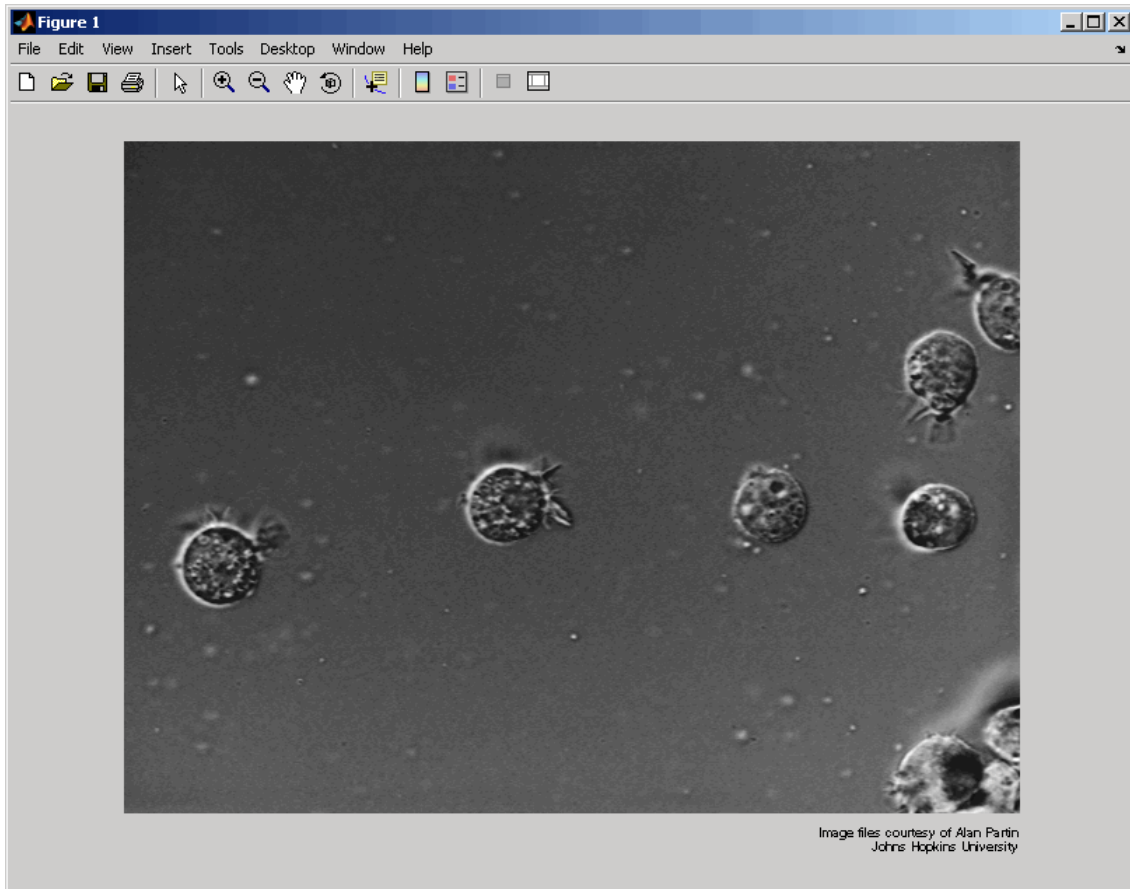
Note In this example, the image files are a set of 10 microscope images of rat prostate cancer cells. These files are only the first 10 of 100 images acquired.

- 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');
```



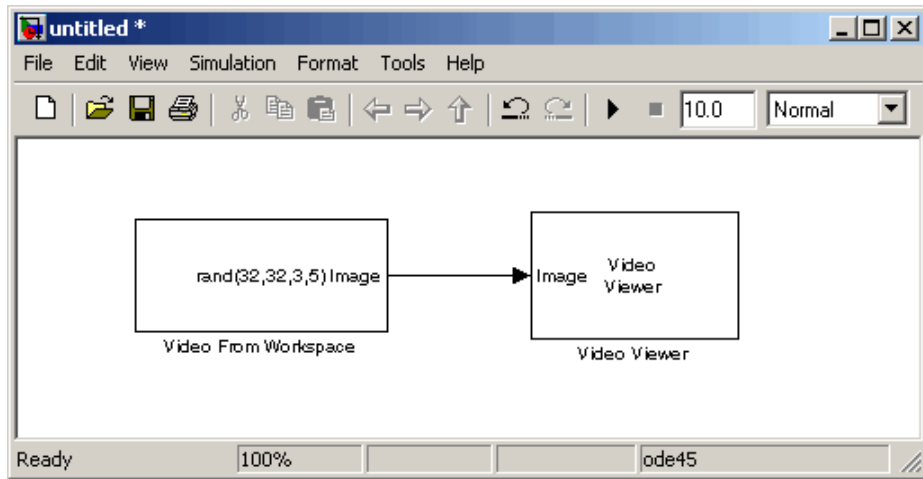
- 3** Use a for loop to create a variable that stores the entire image sequence. You are going to use this variable to import the sequence into Simulink.

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

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

Block	Library	Quantity
Video From Workspace	Video and Image Processing Blockset > Sources	1
Video Viewer	Video and Image Processing Blockset > Sinks	1

5 Connect the blocks so your model looks similar to the following figure.

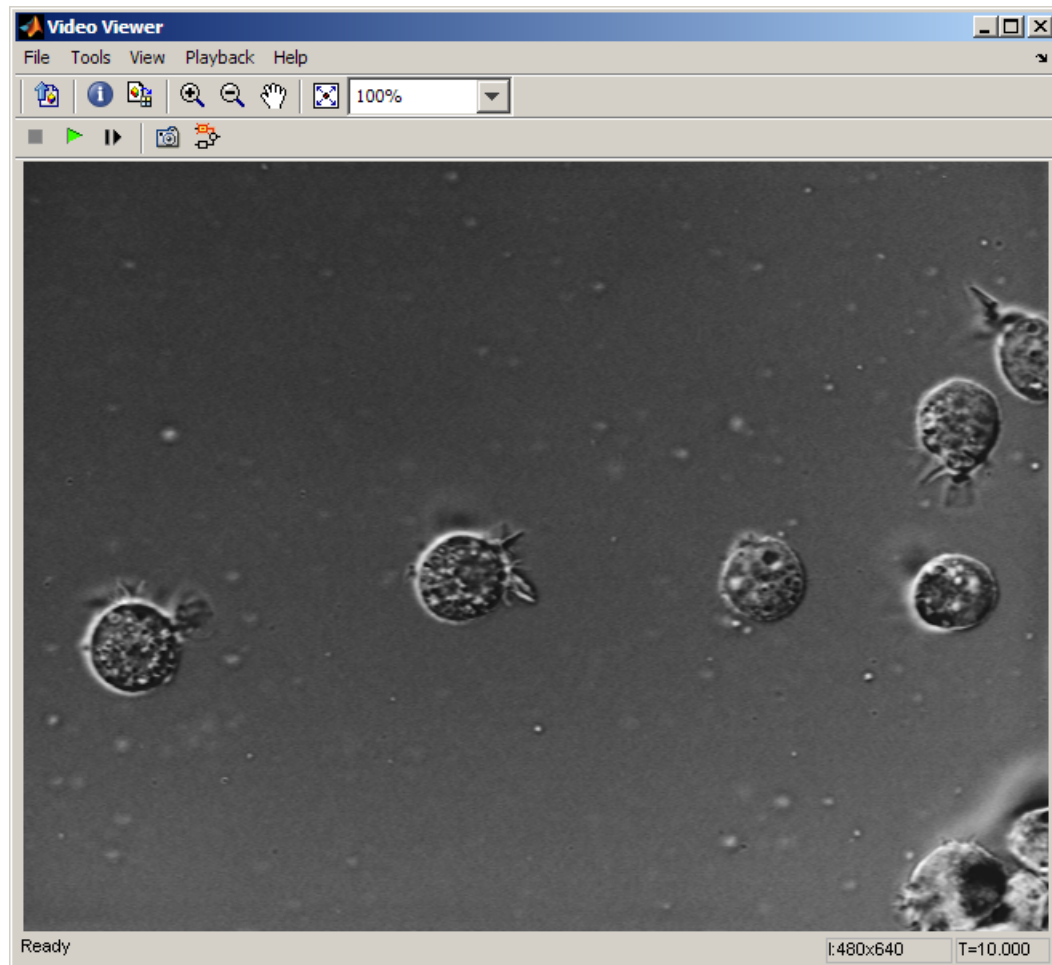


- 6 Use the Video From Workspace block to import the image sequence into Simulink. Set the **Signal** parameter to `my_video`.
- 7 Use the Video Viewer block to view the image sequence. Accept the default parameters.
- 8 Set the configuration parameters. Open the Configuration dialog box by selecting **Simulation > Configuration Parameters**. On the **Solver** pane, set the parameters as follows:
 - **Stop time** = 10
 - **Type** = Fixed-step

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

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

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



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.

Working with Live Video

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 Video and Image Processing blocks. To see the full list of Video and Image Processing demos, type `vipdemos` at the MATLAB command prompt.

Working with Multimedia Files

In this section...
“Blocks That Support Multimedia Files” on page 2-8
“Importing and Viewing Multimedia Files” on page 2-8
“Exporting to Multimedia Files” on page 2-11
“Working with AVI Files” on page 2-14
“Working with Audio” on page 2-38

Blocks That Support Multimedia Files

The Video and Image Processing Blockset software contain blocks that you can use to import and export multimedia files. These blocks include the From Multimedia File block and the To Multimedia File block. If you are working on a Windows platform, these blocks perform best on platforms with DirectX Version 9.0 or later and Windows Media Player Version 11 or later. These blocks also support code generation.

Importing and Viewing Multimedia Files

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

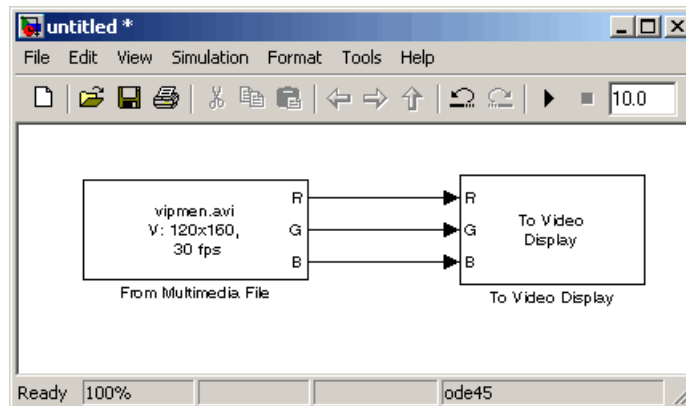
- 1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
To Video Display	Video and Image Processing Blockset > Sinks	1

- 2 Locate a multimedia file that you want to import into Simulink. If you do not have access to a multimedia file, the Video and Image Processing Blockset software has sample multimedia files you can use to complete this procedure.
- 3 Use the From Multimedia File block to import the multimedia file into the model. Double-click the From Multimedia File block:
 - If you do not have your own multimedia file, enter `vipmen.avi` 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.

- 4 Use the To Video Display block to view the multimedia file. Set the **Image signal** parameter to `Separate color signals`.
- 5 Connect the blocks so your model looks similar to the following figure.



- 6 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. On the **Solver** pane, set the parameters as follows:
 - **Stop time** = 20
 - **Type** = Fixed-step
 - **Solver** = Discrete (no continuous states)
- 7 Run your model.

View your video in the To Video Display window that automatically appears when you start your simulation. This window closes as soon as the simulation stops.



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 your Simulink model. In “Exporting to Multimedia Files” on page 2-11, you 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 in the *Video and Image Processing Blockset Reference*. To listen to audio associated with an AVI file, use the To Audio Device block in Signal Processing Blockset software.

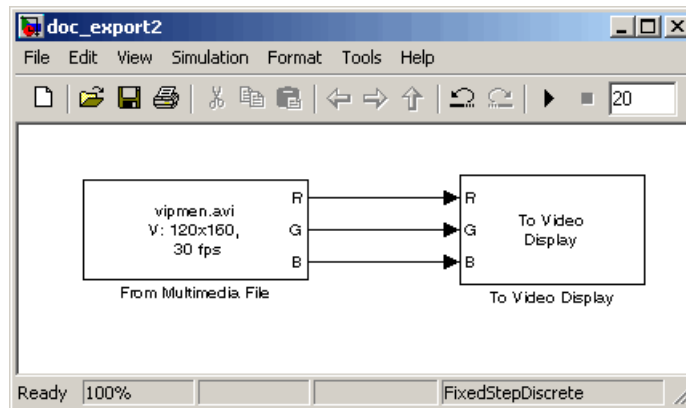
Exporting to Multimedia Files

The Video and Image Processing Blockset blocks enable you to export video data from your Simulink model. In this section, you use the To Multimedia File block to export an multimedia file from your model.

- 1 If the model you created in “Importing and Viewing Multimedia Files” on page 2-8 is not open on your desktop, you can open an equivalent model by typing

```
doc_export2
```

at the MATLAB command prompt.

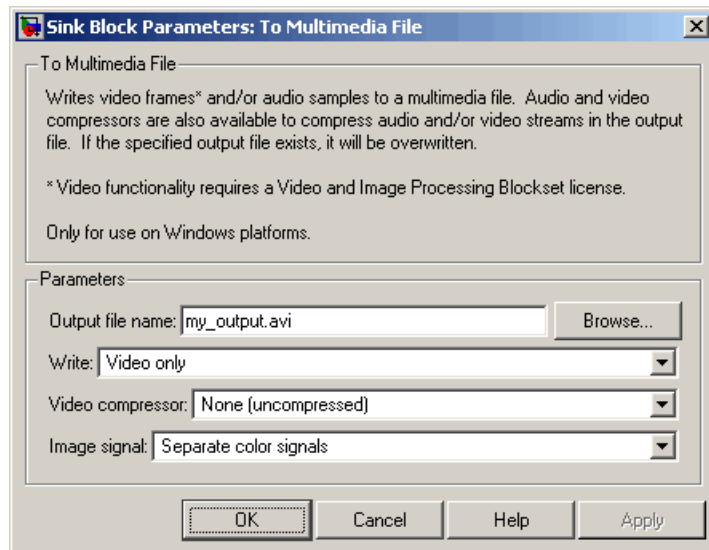


- 2 Click-and-drag the following blocks into your model.

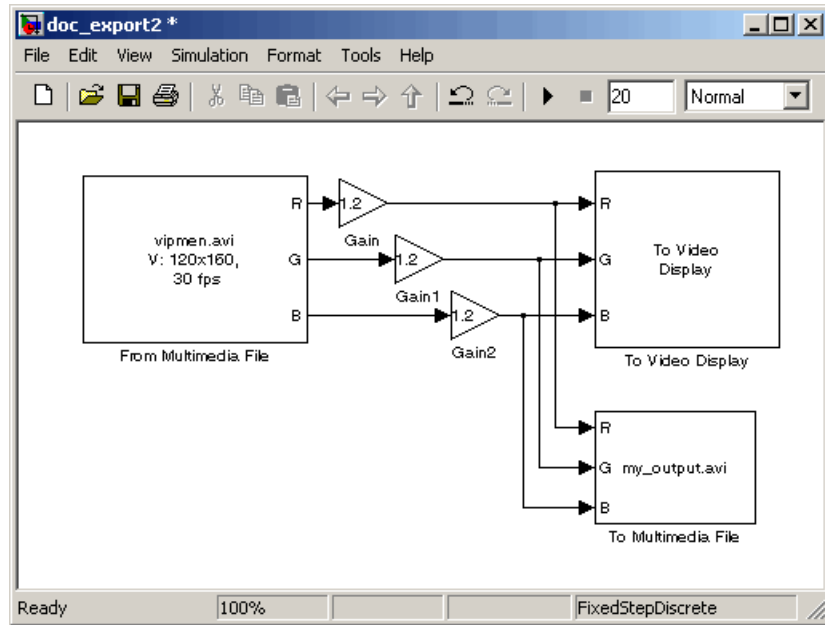
Block	Library	Quantity
To Multimedia File	Video and Image Processing Blockset > Sinks	1
Gain	Simulink > Math Operations	3

- 3 Use the Gain blocks to increase the red, green, and blue values of the video stream. This increases the contrast of the video. Set the block parameters as follows:

- **Main** pane, **Gain** = 1.2
 - **Signal Attributes** pane, **Output data type** = Inherit: Same as input
- 4 Use the To Multimedia File block to export the video to a multimedia file. Set the block parameters as follows:
- **Output file name** = my_output.avi
 - **Write** = Video only
 - **Image signal** = Separate color signals



- 5 Connect the blocks as shown in the following figure. You might need to resize some blocks to do so.



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

- 6 If you have not already done so, set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. On the **Solver** pane, set the parameters as follows:

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

- 7 Run your model.

You can view your video in the To Video Display window. By increasing the red, green, and blue color values, you increased 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.



You have now manipulated your video stream and exported it from a Simulink model to a multimedia file. For more information, see the To Multimedia File block reference page in the *Video and Image Processing Blockset Reference*.

Working with AVI Files

- “Importing and Viewing AVI Files” on page 2-14
- “Exporting to AVI Files” on page 2-18
- “Annotating AVI Files with Video Frame Numbers” on page 2-22
- “Annotating AVI Files at Two Separate Locations” on page 2-26
- “Saving Portions of an AVI File to Separate Files” on page 2-30

Importing and Viewing AVI Files

Before you can analyze or operate on your data, you must import it into your Simulink model. Blocks from the Sources library, such as the From Multimedia File block, can help you with this type of task.

In this section, you use the From Multimedia File block to import video from an AVI file into your model and the Video Viewer block to view it:

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

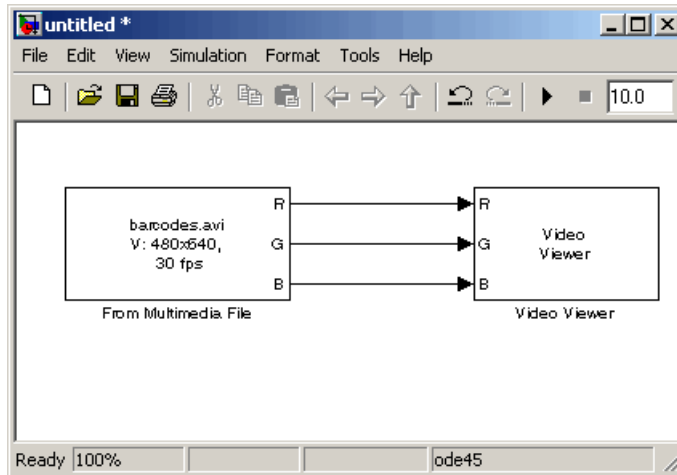
Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
Video Viewer	Video and Image Processing Blockset > Sinks	1

2 Use the From Multimedia File block to import an AVI file into the model. Double-click the From Multimedia File block. The Video and Image Processing Blockset software has sample AVI files you can use to complete this procedure.

- If you do not have your own AVI file, enter `barcodes.avi` for the **File name** parameter.
- If the AVI file is on your MATLAB path, enter the AVI filename for the **File name** parameter.
- If the file is not on your MATLAB path, use the **Browse** button to locate the AVI filename.
- **Image signal** = 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.

- 3** Use the Video Viewer block to view the AVI file. Click the **File** menu of the Video Viewer GUI to set the **Image signal** parameter to `Separate color signals`.
- 4** Connect the blocks so your model looks similar to the following figure.

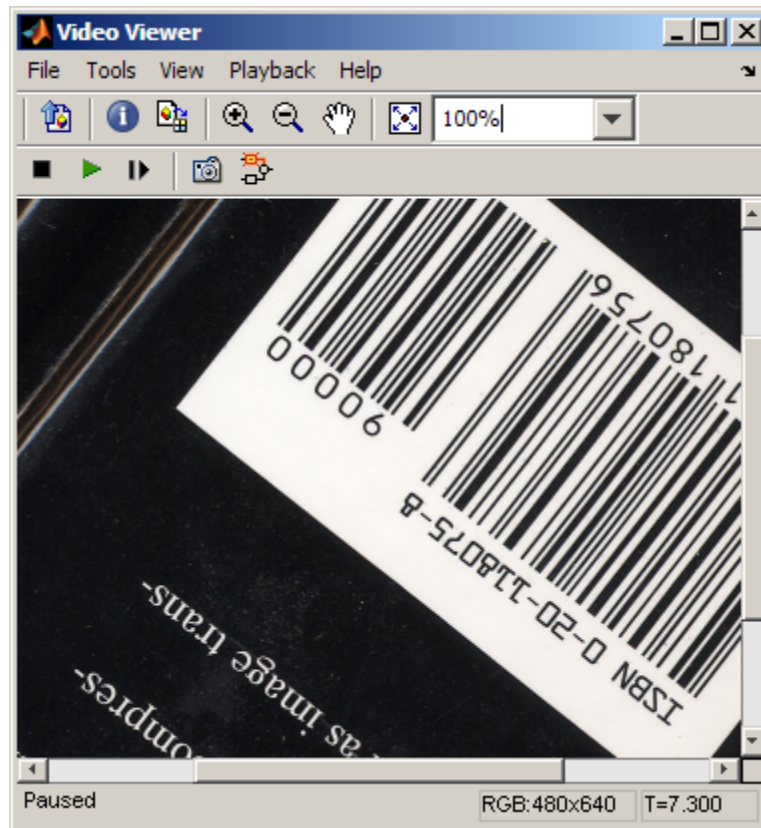


5 Set the configuration parameters. Open the Configuration dialog box by selecting **Simulation > Configuration Parameters**. Set the parameters as follows:

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

6 Run your model.

View your video in the Video Viewer window that automatically appears when you start your simulation.



Note The video that is displayed in the Video Viewer window runs as fast as Simulink processes the video frames. If you are on a Windows platform and you want to run the video at the frame rate that corresponds to the input sample time, use the To Video Display block.

You have now imported and displayed video data in your Simulink model. In “Exporting to AVI Files” on page 2-18, you manipulate your video stream and export it to an AVI file. For more information on the blocks used in this example, see the From Multimedia File and Video Viewer block reference pages in the *Video and Image Processing Blockset Reference*. To listen to

audio associated with an AVI file, use the To Audio Device block in Signal Processing Blockset software.

Note The Video Viewer block is supported on all platforms, but it does not support code generation. If you are on a Windows platform, you can use the To Video Display block to display video data. This block supports code generation. For more information, see the To Video Display block reference page in the *Video and Image Processing Blockset Reference*.

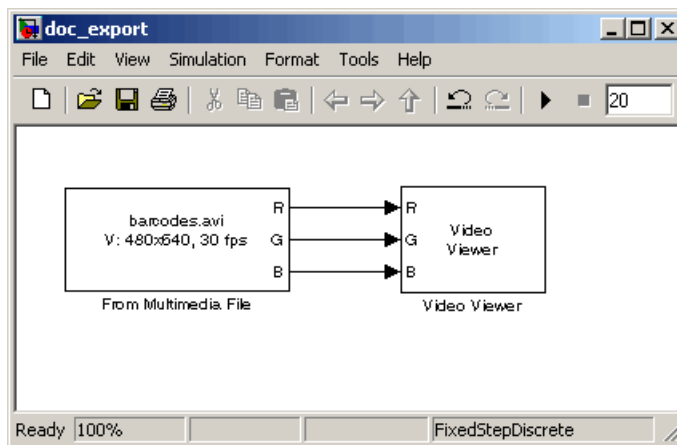
Exporting to AVI Files

The Video and Image Processing Blockset blocks enables you to export video data from your Simulink model. In the following procedure, you use the To Multimedia File block to export video data from your model into an AVI file:

- 1 If the model you created in “Importing and Viewing AVI Files” on page 2-14 is not open on your desktop, open an equivalent model by typing

`doc_export`

at the MATLAB command prompt.



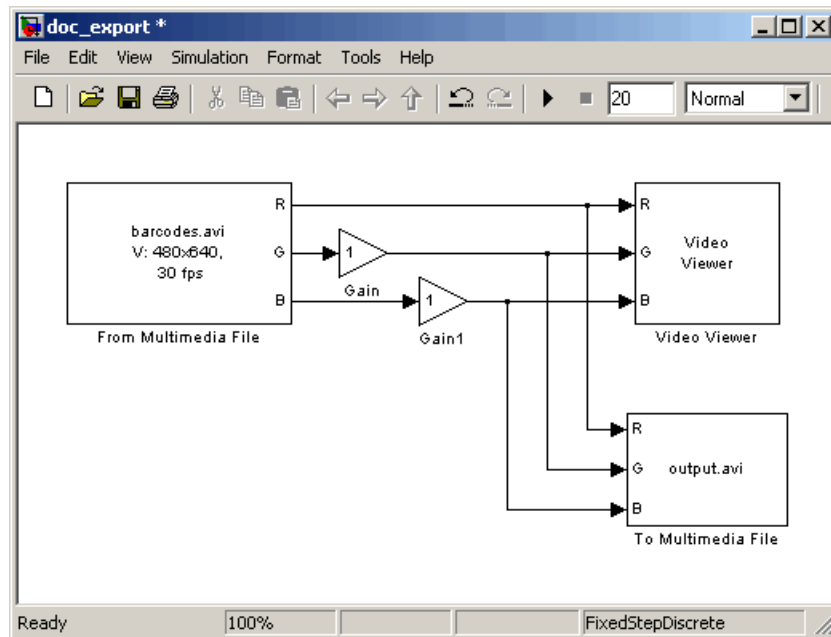
- 2 Click-and-drag the blocks shown on the following table into your model.

Block	Library	Quantity
To Multimedia File	Video and Image Processing Blockset > Sinks	1
Gain	Simulink > Math Operations	2

3 Change the inputs to the To Multimedia File block. Set the block parameters as follows:

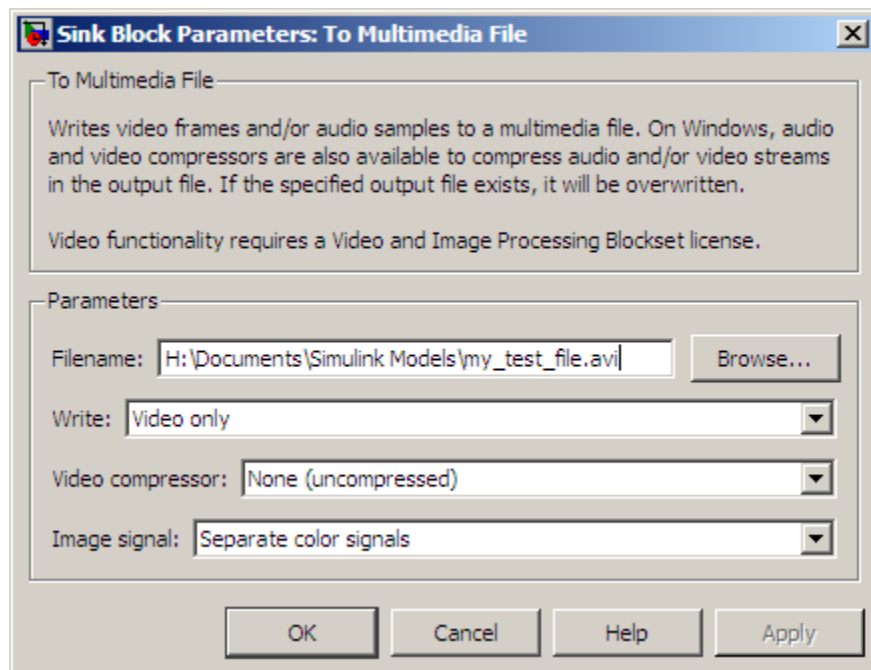
- **Write** = Video only
- **Image signal** = Separate color signals

4 Connect the blocks as shown in the following figure. You might need to resize some blocks to do so.



5 Use the Gain block to change the green values of the video stream. Set the block parameters as follows:

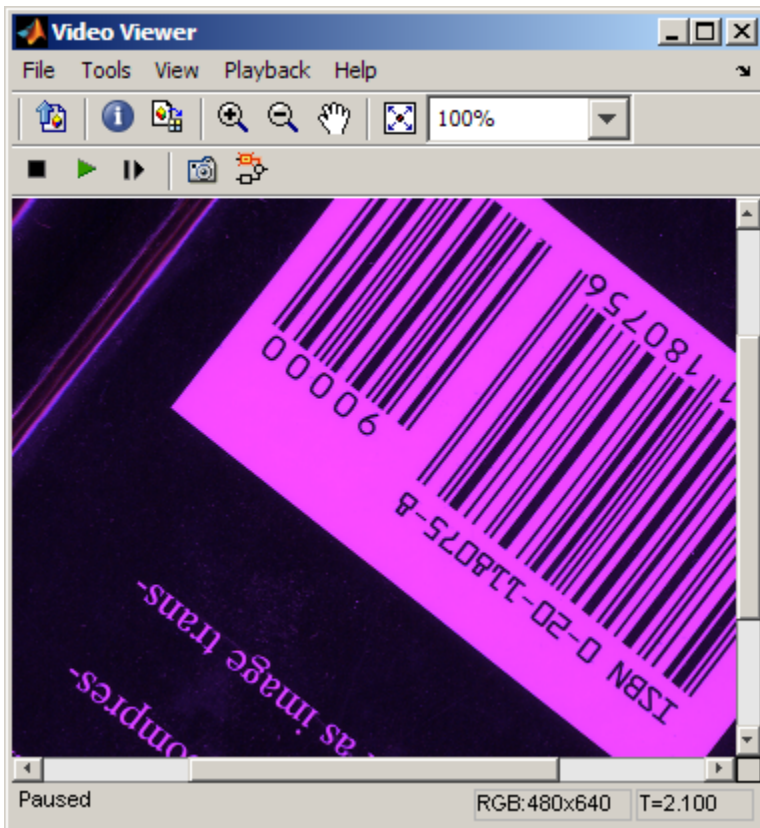
- **Main** pane, **Gain** = 0.3
 - **Signal Attributes** pane, **Output data type** = Inherit:Same as input
- 6 Use the Gain1 block to change the blue values of the video stream. Set the block parameters as follows:
- **Main** pane, **Gain** = 1.5
 - **Signal Attributes** pane, **Output data type** = Inherit:Same as input
- 7 Use the To Multimedia File block to export the video to an AVI file. Set the **File name** parameter to `my_test_file.avi`.



- 8 If you have not already done so, set the configuration parameters. Open the Configuration dialog box by selecting **Simulation > Configuration Parameters**. Set the parameters as follows:
- **Solver** pane, **Stop time** = 20
 - **Solver** pane, **Type** = Fixed-step

- **Solver** pane, **Solver** = Discrete (no continuous states)
- 9 Run your model.

You can view your video in the Video Viewer window. The To Multimedia File block exports the video data from the Simulink model to an AVI file that it creates in your current folder.



You have now manipulated your video stream and exported it from a Simulink model to an AVI file. For more information, see the To Multimedia File block reference page in the *Video and Image Processing Blockset Reference*.

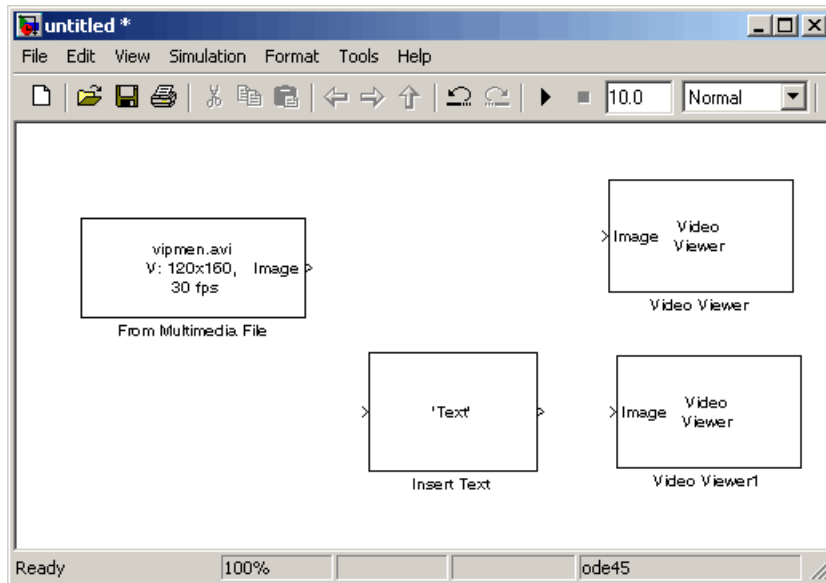
Annotating AVI Files with Video Frame Numbers

You can use the Insert Text block to overlay text on video stream. In this example, you add a running count of the number of video frames to a video.

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

Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
Insert Text	Video and Image Processing Blockset > Text & Graphics	1
Video Viewer	Video and Image Processing Blockset > Sinks	2

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



- 3 Use the From Multimedia File block to import the video into the Simulink model. Set the **Image color space** parameter to Intensity.

- 4 Open the Surveillance Recording demo by typing

```
vipsurveillance
```

at the MATLAB command prompt.

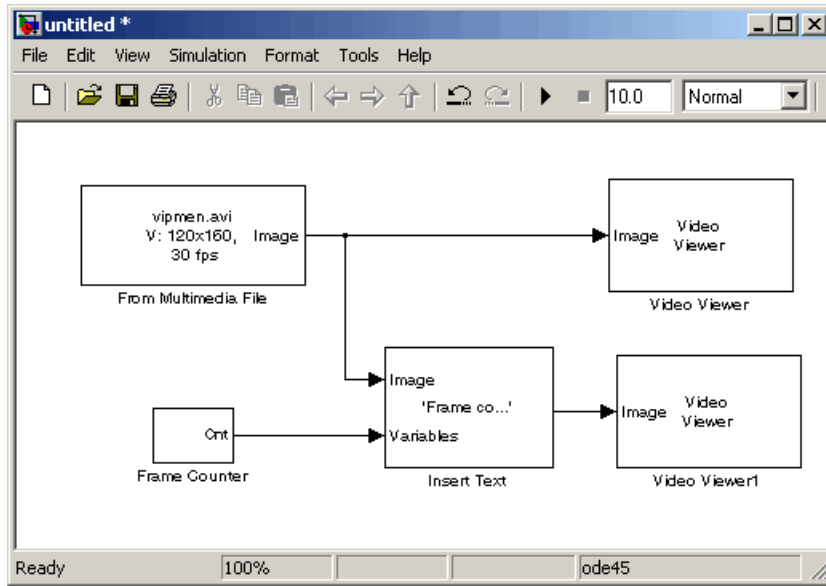
- 5 Click-and-drag the Frame Counter block from the demo model into your model. This block counts the number of frames in an input video.

- 6 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, **Location** = [85 2]
- **Main** pane, **Color value** = 1
- **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 (it is expecting a decimal) and substitutes it for the `%d` in the string. You used the **Location** 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.

- 7 Use the Video Viewer blocks to view the original and annotated videos. Accept the default parameters.
- 8 Connect the blocks as shown in the following figure.

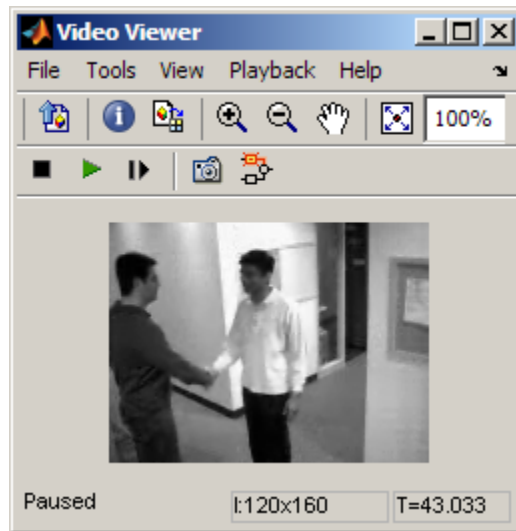


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

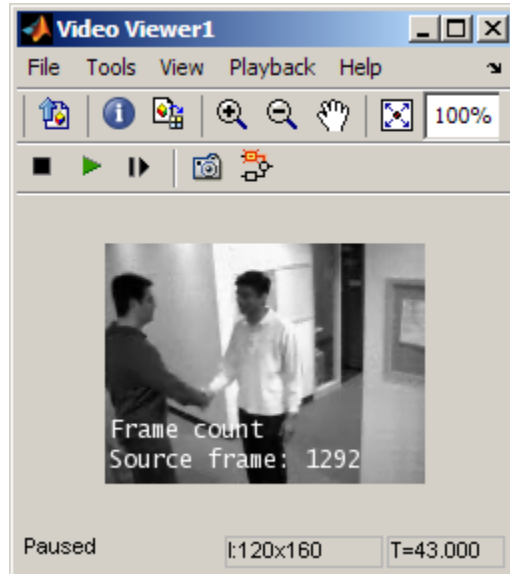
- **Solver** pane, **Stop time** = inf
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

10 Run the model.

The original video appears in the Video Viewer window.



The annotated video appears in the Video Viewer1 window.



You have now added descriptive text to a video stream. For more information, see the Insert Text block reference page in the *Video and Image Processing Blockset Reference*. For related information, see the Draw Shapes and Draw Markers block reference pages.

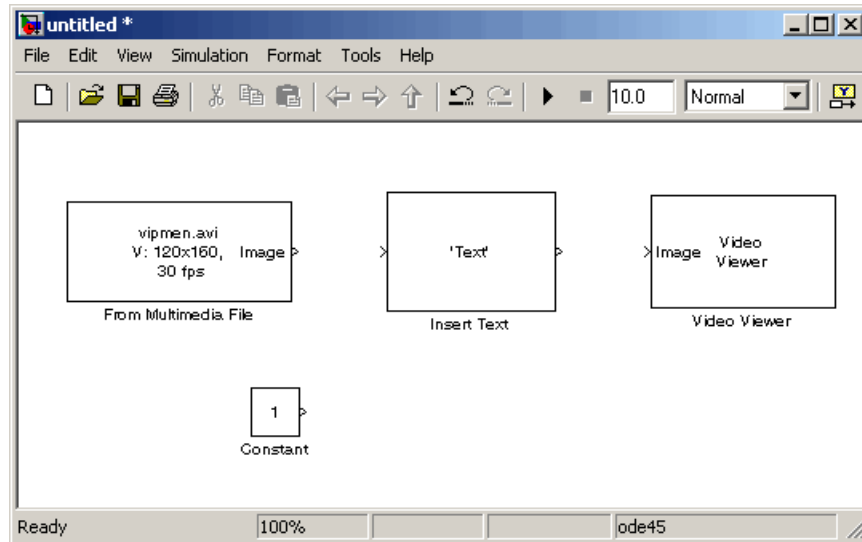
Annotating AVI Files at Two Separate Locations

You can use the Insert Text block to overlay text on a video stream at two separate locations in the video frame.

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

Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
Insert Text	Video and Image Processing Blockset > Text & Graphics	1
Video Viewer	Video and Image Processing Blockset > Sinks	1
Constant	Simulink > Sources	1

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



- 3 Use the From Multimedia File block to import the video stream into the Simulink model. Accept the default parameters.
- 4 Use the Insert Text block to annotate the video with two text strings. Set the block parameters as follows:
 - **Main pane, Text** = 'Text position: Row %d and Column %d'
 - **Main pane, Location** = `[[5 10]' [80 10]'`

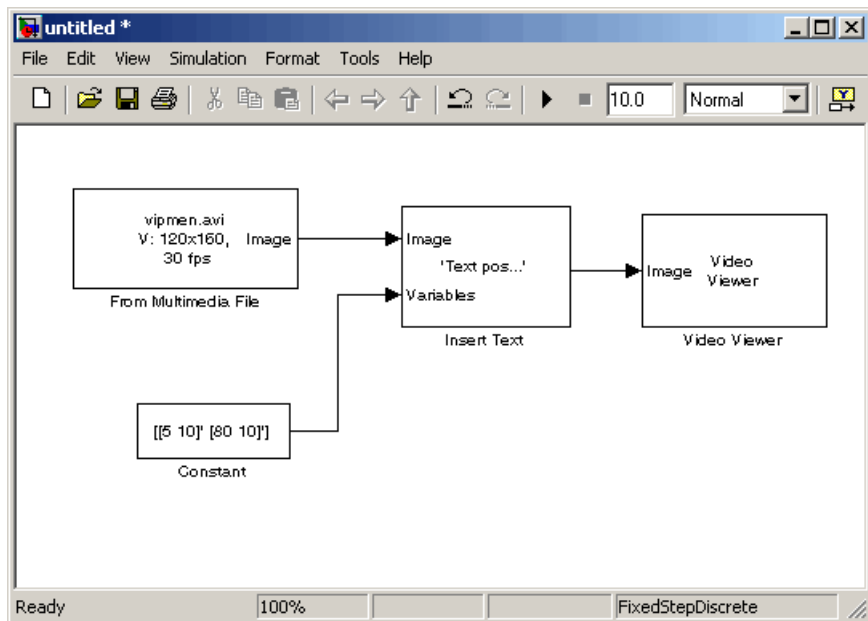
By setting the **Text** parameter to 'Text position: Row %d and Column %d', you are asking the block to replace each conversion specification (%d) with a decimal input to the Variables port. You used the **Location** parameter to specify where to print each text string. In this case, the block places the top-left corner of the text box that surrounds the first text string 5 rows down and 10 rows over from the top left corner of the image. The block places the second text string 80 rows down and 10 rows over.

- 5 Use the Constant block to specify the decimal values input into the Insert Text block's Variables port. Because the conversion specification is %d, the values must be an integer data type. Set the block parameters as follows:
 - **Main pane, Constant value** = `[[5 10]' [80 10]'`

- **Main** pane, clear the **Interpret vector parameters as 1-D** check box.
- **Signal Attributes** pane, **Output data type** = uint8

The Insert Text block substitutes the values from the first column of the **Constant value** parameter into the first text string and the values from the second column into the second text string.

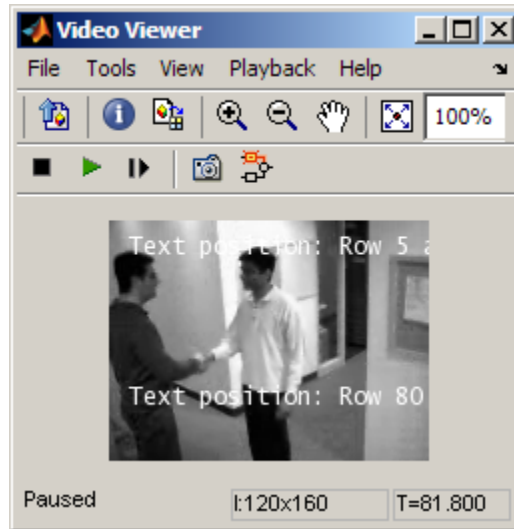
- 6 Use the Video Viewer blocks to view the annotated image. Accept the default parameters.
- 7 Connect the blocks as shown in the following figure.



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

- 9 Run the model.

The annotated video appears in the Video Viewer window.



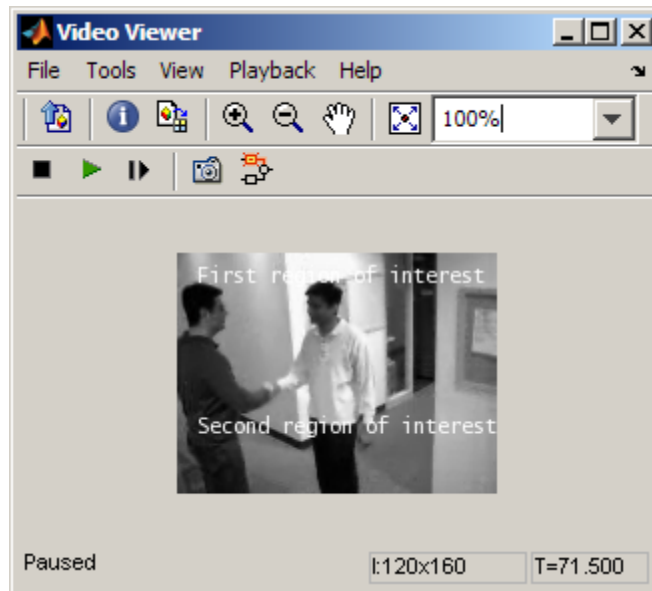
Alternatively, you can input two string values at the Variables port.

- 10 On the Insert Text block dialog box, set the **Text** parameter to '%s region of interest'.

You are asking the block to replace the %s conversion specification with a string input to the Variables port.

- 11 Use the Constant block to specify the strings to substitute into the first and second text strings. Because the conversion specification is %s, the values must be 8-bit unsigned integer data types. Set the **Constant value** parameter to `[uint8('First') 0 uint8('Second')]`.
- 12 Run the model.

The annotated video appears in the Video Viewer window.



You have now added descriptive text to a video stream. For more information, see the Insert Text block reference page in the *Video and Image Processing Blockset Reference*.

Saving Portions of an AVI File to Separate Files

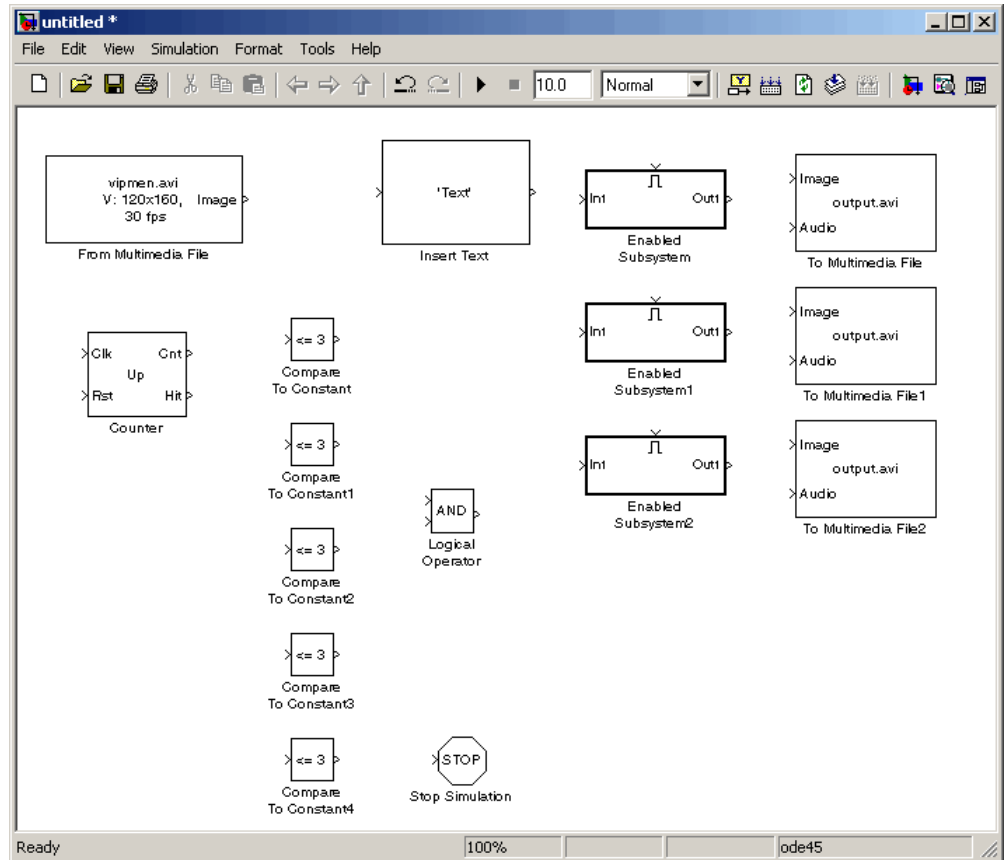
In this section, you use To Multimedia File and Enabled Subsystem blocks to save portions of one AVI file to three separate AVI files.

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

Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
Insert Text	Video and Image Processing Blockset > Text & Graphics	1
Enabled Subsystem	Simulink > Ports & Subsystems	3

Block	Library	Quantity
To Multimedia File	Video and Image Processing Blockset > Sinks	3
Counter	Signal Processing Blockset > Signal Management > Switches and Counters	1
Compare To Constant	Simulink > Logic and Bit Operations	5
Logical Operator	Simulink > Logic and Bit Operations	1
Stop Simulation	Simulink > Sinks	1

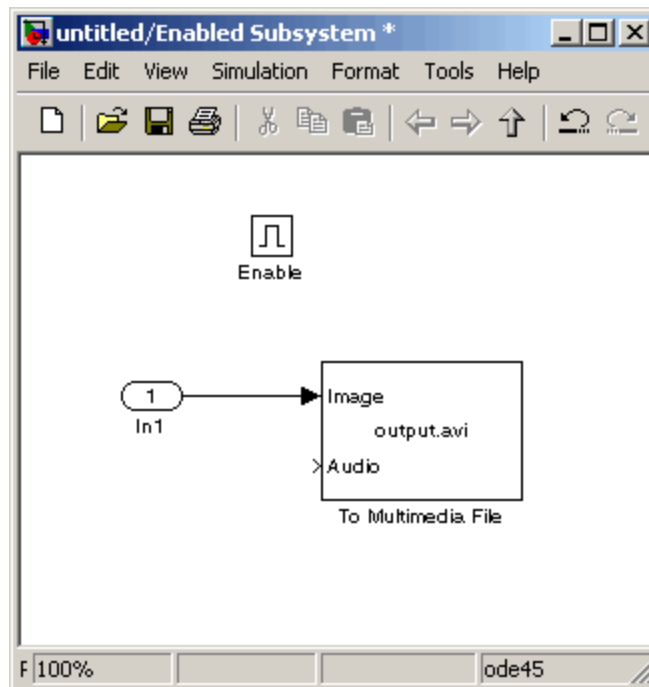
- 2** Place the blocks so that your model looks similar to the one in the following figure.



- 3 Use the From Multimedia File block to import an AVI file into your model. Set the parameters as follows:
 - Uncheck **Inherit sample time from file** checkbox
 - Set **Desired sample time** parameter = 1/30
- 4 Use the Insert Text block to annotate the video stream with the frame numbers. Set the parameters as follows:
 - **Text** = 'Frame %d'
 - **Location** = [10 10]
 - **Color** = [0 1 0]

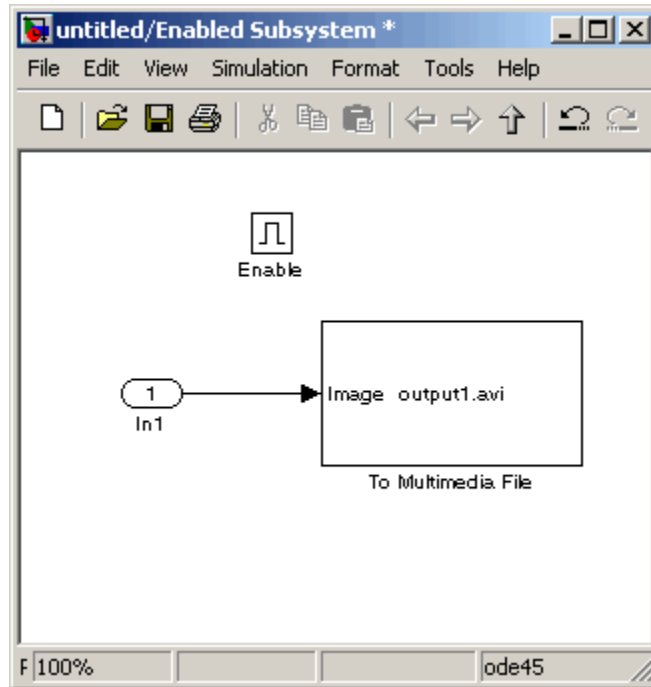
The block writes the frame number in green in the upper-left corner of the output video stream.

- 5 Double-click each Enabled Subsystem block, and click-and-drag one of the To Multimedia File blocks into it.
- 6 Inside each Enabled Subsystem, connect the blocks so that your subsystem looks similar to the one in the following figure.



- 7 Use the To Multimedia File blocks to send the video stream to three separate AVI files. Set the block parameters as follows:
 - **Output file name** = output1.avi, output2.avi, and output3.avi, respectively
 - **Write** = Video only

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



- 8 Use the Counter block to count the number of video frames. You use this information to specify which frames are sent to which file. Set the block parameters as follows:
 - **Count event** = Free running
 - **Initial count** = 1
 - **Output** = Count
 - Clear the **Reset input** check box.
 - **Sample time** = 1/30
 - **Count data type** = uint16
- 9 Use the Compare to Constant block to send frames 1 to 9 to the first AVI file. Set the block parameters as follows:
 - **Operator** = <
 - **Constant value** = 10

- 10** Use the Compare to Constant1 and Compare to Constant2 blocks to send frames 10 to 19 to the second AVI file. Set the Compare to Constant1 block parameters as follows:

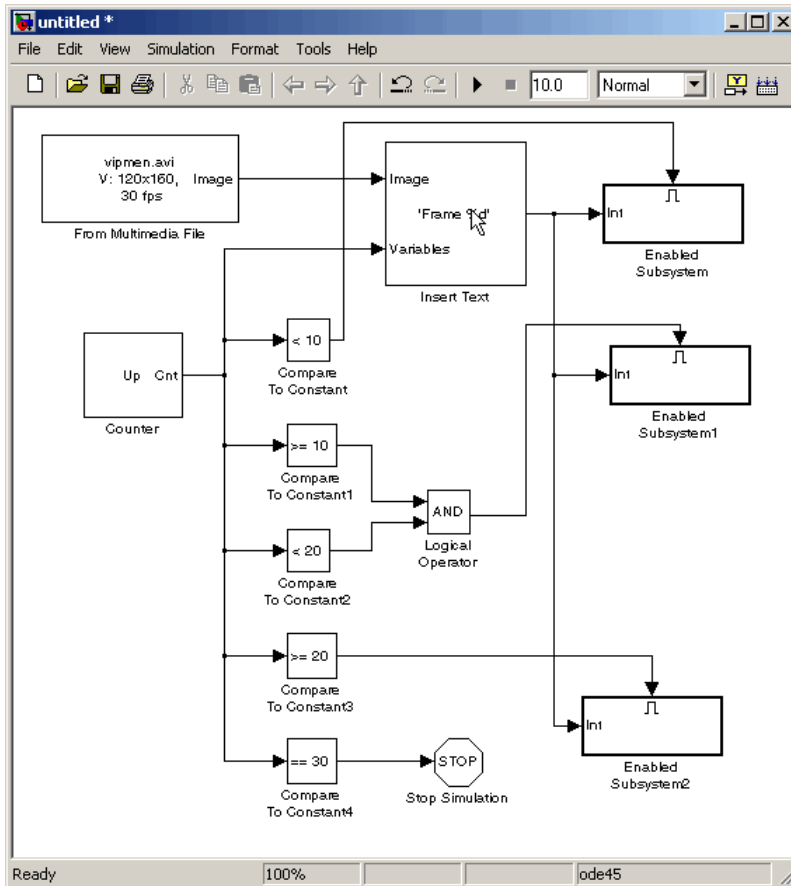
 - **Operator** = >=
 - **Constant value** = 10

Set the Compare to Constant2 block parameters as follows:

 - **Operator** = <
 - **Constant value** = 20
- 11** Use the Compare to Constant3 block to send frames 20 to 30 to the third AVI file. Set the block parameters as follows:

 - **Operator** = >=
 - **Constant value** = 20
- 12** Use the Compare to Constant4 block to stop the simulation when the video reaches frame 30. Set the block parameters as follows:

 - **Operator** = ==
 - **Constant value** = 30
 - **Output data type mode** = boolean
- 13** Connect the blocks so that your model resembles the one in the following figure.



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

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

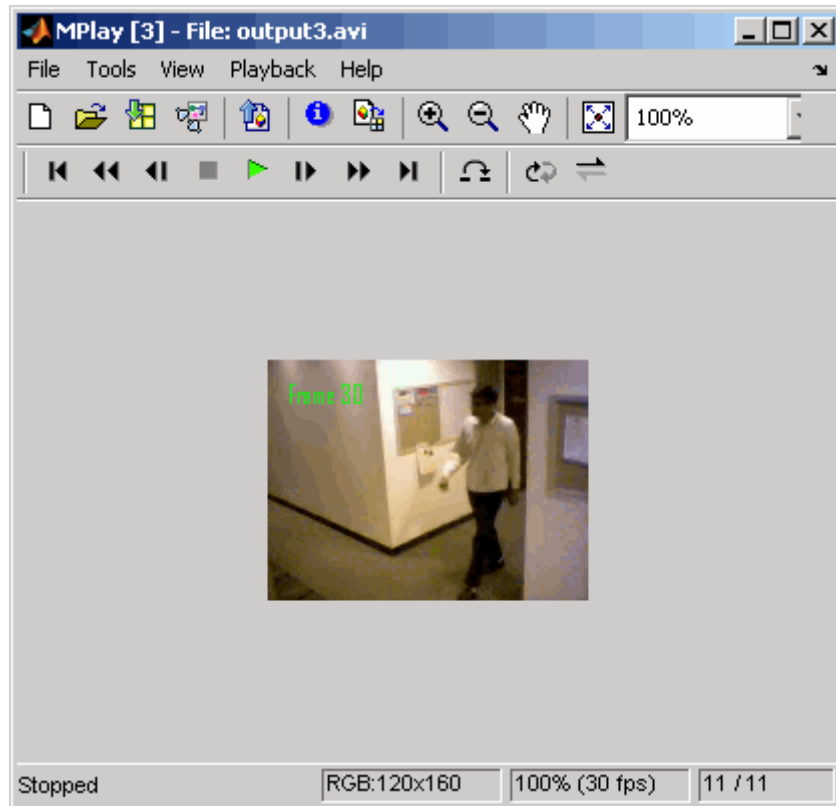
15 Run your model.

The model saves the three output AVI files in your current folder.

- 16** View the resulting files by typing the following commands at the MATLAB command prompt:

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

Then, press the **Play** button on the MPlay GUI.



You have now sent portions of an AVI file to three separate AVI files using an Enabled Subsystem block, a To Multimedia File block, and a trigger signal. 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.

Working with Audio

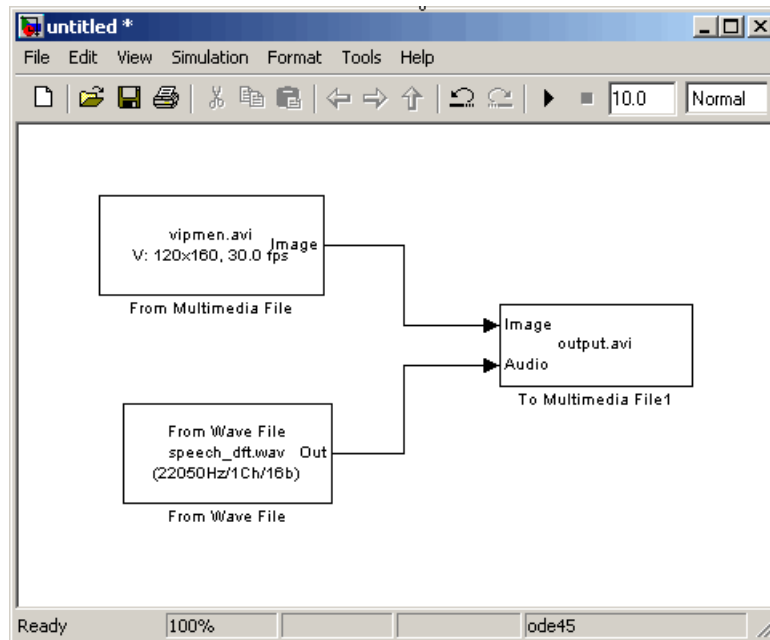
In this example, you use the From Multimedia File block to import a video stream into a Simulink model. You also use Signal Processing Blockset software From Wave File block to import an audio stream into the model. Then you write this audio and video to a single file using the To Multimedia File block.

This procedure assumes you are working on a Windows platform:

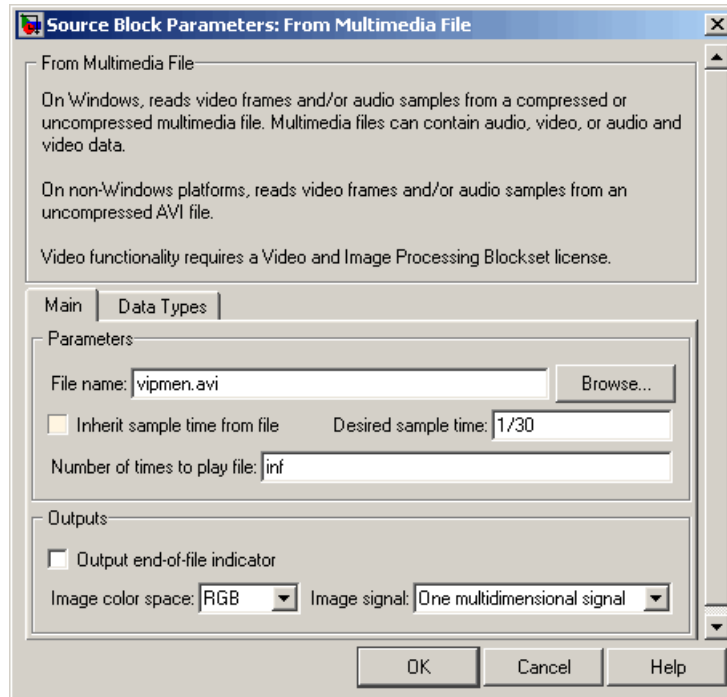
- 1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
From Multimedia File	Video and Image Processing Blockset > Sources	1
From Wave File	Signal Processing Blockset > Signal Processing Sources	1
To Multimedia File	Video and Image Processing Blockset > Sinks	1

- 2 Connect the blocks so your model looks similar to the following figure.

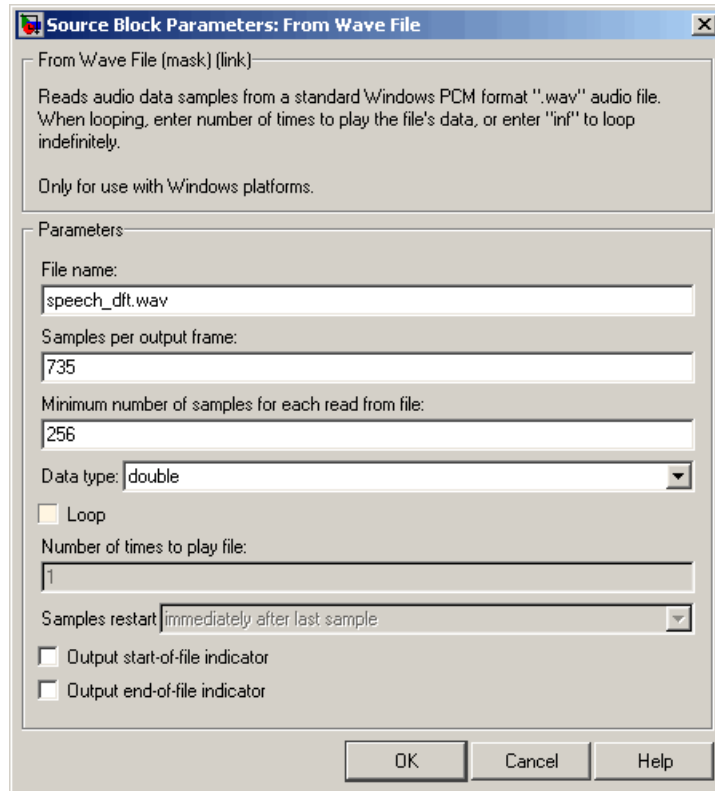


- 3 Use the From Multimedia File block to import a multimedia file into the model. Deselect the **Inherit sample time from file** check box. (Deselecting the checkbox enables **Desired sample time** parameter.) Accept the following default parameters.



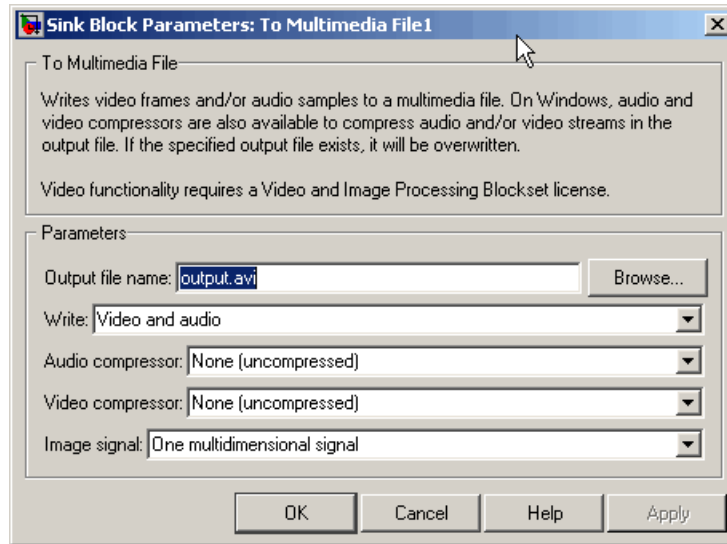
The From Multimedia File block inherits its sample time from `vipmen.avi`. For video signals, the sample time is equivalent to the frame period. Because this file's frame rate is 30 frames per second (fps) and the frame period is defined as $1/\text{frame rate}$, the frame period of this block is 0.0333 seconds per frame.

- 4 Use the From Wave File block to import an audio file into the model. To calculate the output frame size, divide the frequency of the audio signal (22050 samples per second) by the frame rate (which is approximately 30 frames per second) to get 735 samples per frame. Set the **Samples per output frame** parameter to 735.



The frame period of the audio signal must match the frame period of the video signals, which is 0.0333 seconds per frame. Since 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. Alternatively, you can verify that the frame period of the audio and video signals is the same using a Simulink Probe block.

- 5 Use the To Multimedia File to output the audio and video signals to a single multimedia file. Select **Video** and **audio** for the **Write** parameter and **One multidimensional signal** for the **Image signal** parameter. Accept the other default parameters.



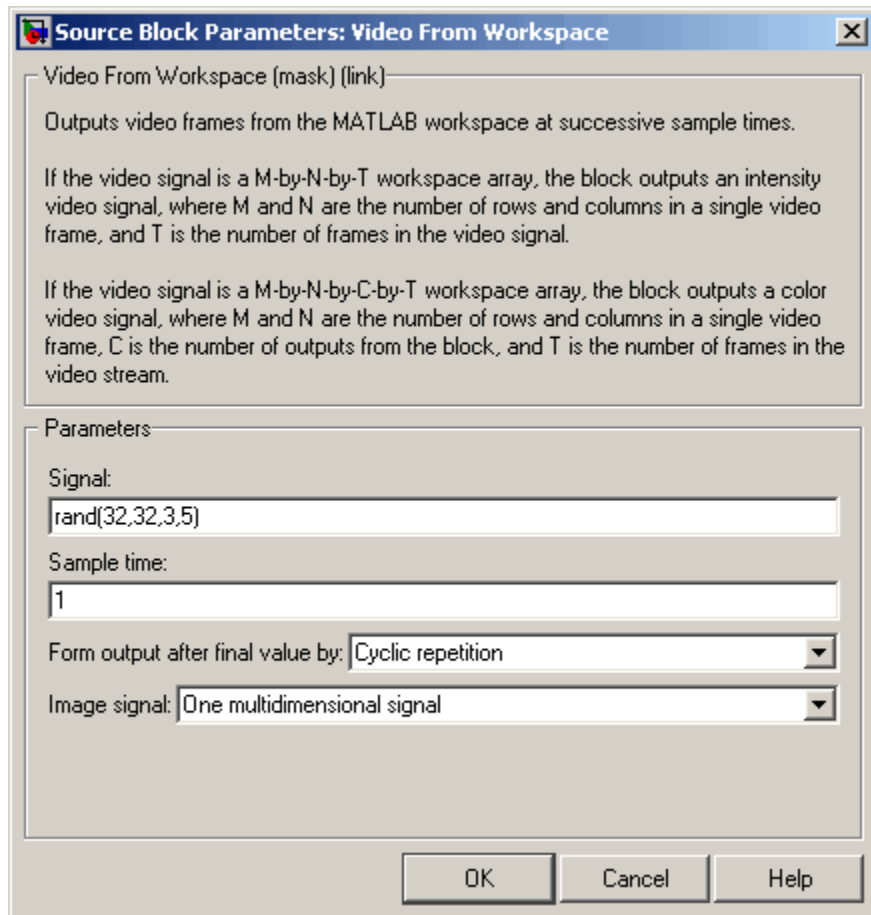
- 6 Set the configuration parameters. Open the Configuration dialog box by selecting **Simulation > Configuration Parameters**. On the **Solver** pane, set the parameters as follows:
 - **Stop time** = 10
 - **Type** = Fixed-step
 - **Solver** = Discrete (no continuous states)
- 7 Run your model. The model creates a multimedia file called `output.avi` in your current folder.
- 8 Play the multimedia file using a media player. The original video file now has an audio component to it.

You have now combined audio and video information into a single file using the To Multimedia File block. For more information, see the To Multimedia File block reference page in the *Video and Image Processing Blockset Reference*.

Working with MATLAB Workspace Variables

How to 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.



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.

Viewing Video

- “Viewing Video Files” on page 3-2
- “Viewing Video Signals in Simulink” on page 3-3
- “Viewing Video File Frames” on page 3-22

Viewing Video Files

The Video and Image Processing Blockset provides three video viewer applications to accommodate your specific requirements. This table summarizes the intended use for these viewers.

Viewer	Intended Use
Video Viewer	A full featured viewer for your Simulink model. The Video Viewer includes simulation controls and analysis tools.
To Video Display	A Windows® only lightweight, higher performance simple display. This block also generates code.
mp1ay	View video signals in Simulink models without adding blocks to your model or view signal from the MATLAB workspace or directly from a file.

Viewing Video Signals in Simulink

In this section...

“Using the Video Viewer Block” on page 3-3

“Using the To Video Display Block” on page 3-3

“Using the MPlay GUI” on page 3-3

Using the 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 in the *Video and Image Processing Blockset Reference*.

Using the 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 in the *Video and Image Processing Blockset Reference*.

Using the MPlay GUI

The MPlay GUI enables you to view video signals in Simulink models without adding blocks to your model. You can also view videos represented as variables in the MATLAB workspace.

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.

For further information on using the MPlay GUI, see the following topics:

- “Connecting MPlay to Your Simulink Model” on page 3-4
- “MPlay GUI Interface” on page 3-6

For more information about the MPlay GUI, see the `mplay` function reference page.

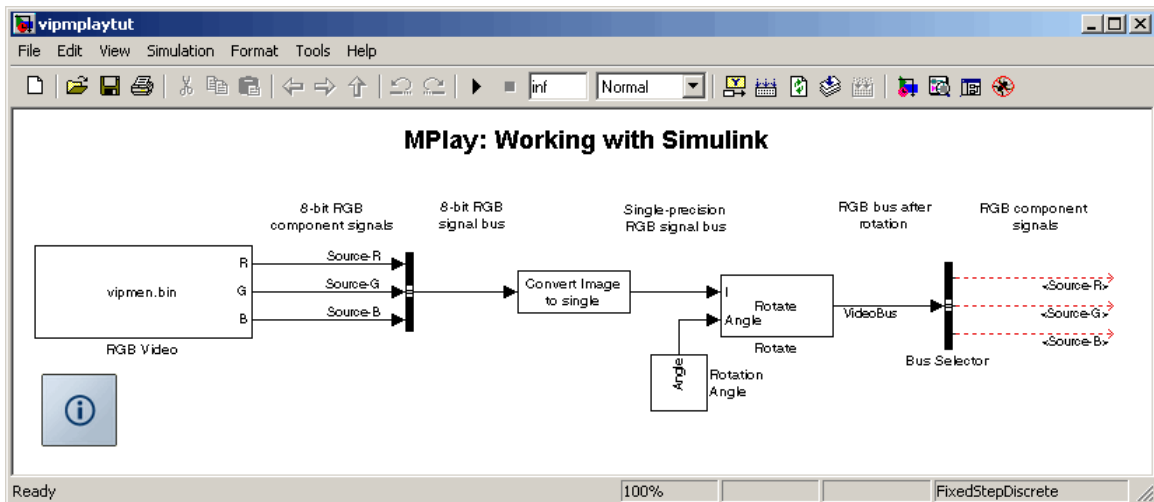
Connecting MPlay to Your Simulink Model

Set Simulink simulation mode to Normal to use `mplay`. MPlay does not work when you use “Acceleration Modes” on page 1-28.


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

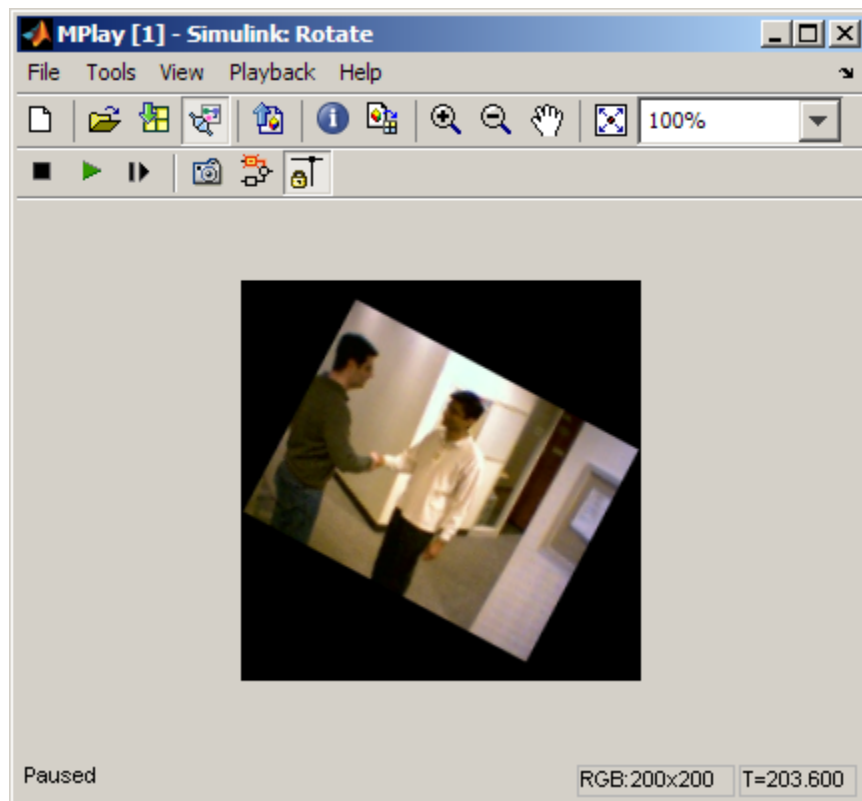
- 1 Open a Simulink model. At the MATLAB command prompt, type


```
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.



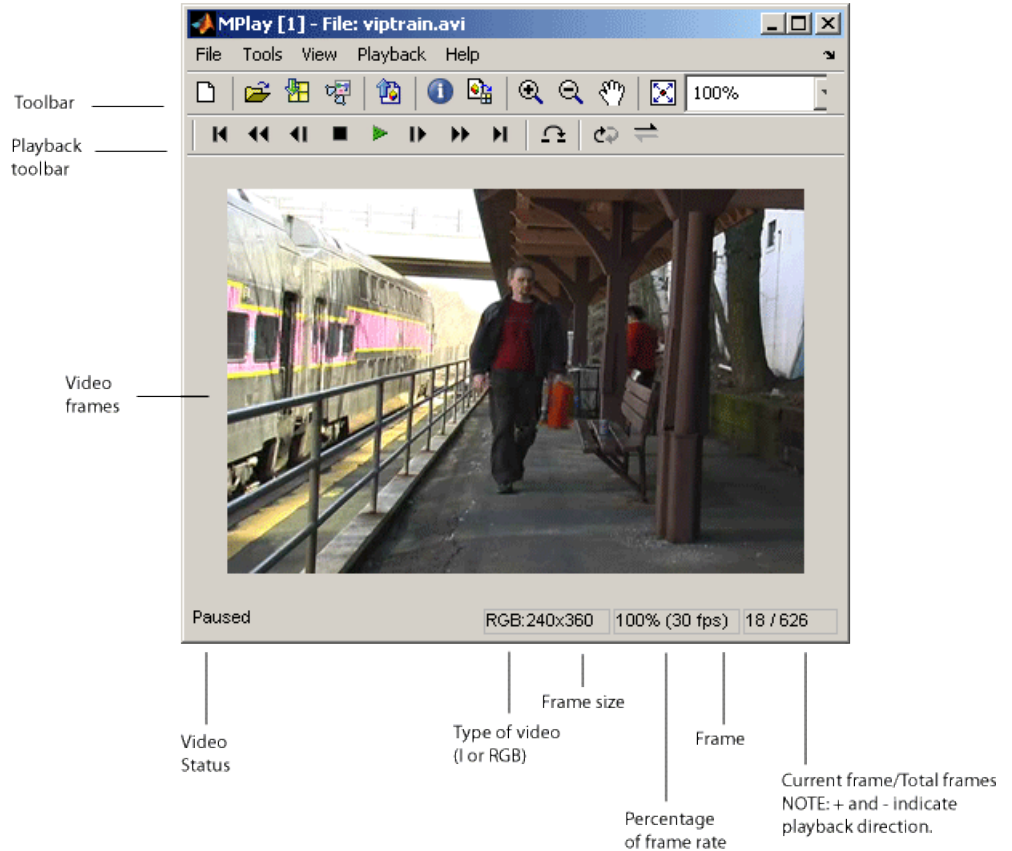
- 6 Change to floating-scope mode by clicking the *persistent connect* GUI element,  button.

- 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, Real-Time Workshop does not generate code for the MPlay GUI.







MPlay GUI Interface







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 > 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.
	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.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	Tools > Pixel Region	N/A	Open the Pixel Region tool. For more information about this tool, see the Image Processing Toolbox documentation.
	Tools > Zoom In	N/A	Zoom in on the video display.
	Tools > Zoom Out	N/A	Zoom out of the video display.
	Tools > Pan	N/A	Move the image displayed in the GUI.
	Tools > Maintain Fit to Window	N/A	Scale video to fit GUI size automatically. Toggle the button on or off.
	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.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	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.
	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.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	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
	Playback > Stop	S	Stop the video. This button also controls the Simulink model.
	Playback > Start	P, Space bar	Play the video. This button also controls the Simulink model.
	Playback > Pause	P, Space bar	Pause the video. This button also controls the Simulink model and appears only when the video is playing.
	Playback > Step Forward	Right arrow, Page Down	Step forward one frame. This button also controls the Simulink model.
	Playback > Simulink Snapshot	N/A	Click this button to freeze the display in the MPlay window.

GUI	Menu Equivalent	Shortcut Keys and Accelerators	Description
	Playback > Highlight Simulink Signal	Ctrl+L	In the model window, highlight the Simulink signal the MPlay GUI is displaying.
	Playback > 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.
	Playback > Floating Signal Connection (selected)	N/A	Indicates floating Simulink connection. In this mode, you can click different signals in 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.

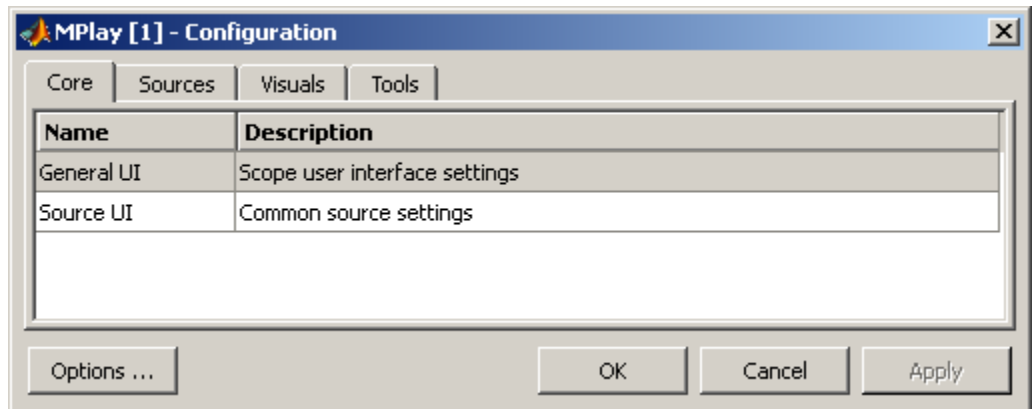
- To open the Configuration dialog box, select **File > Configuration Set > Edit**.
- To save the configuration settings for future use, select **File > Configuration Set > Save as**.

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**.

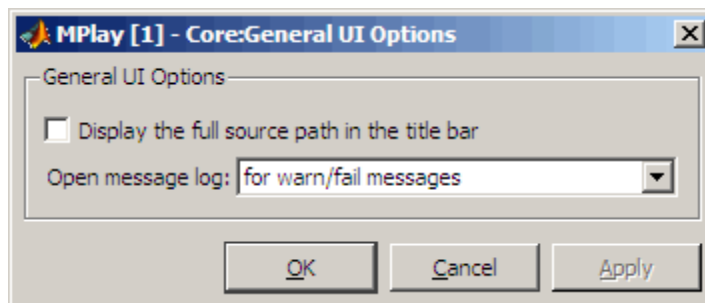
Configuration Core Pane

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



General UI

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

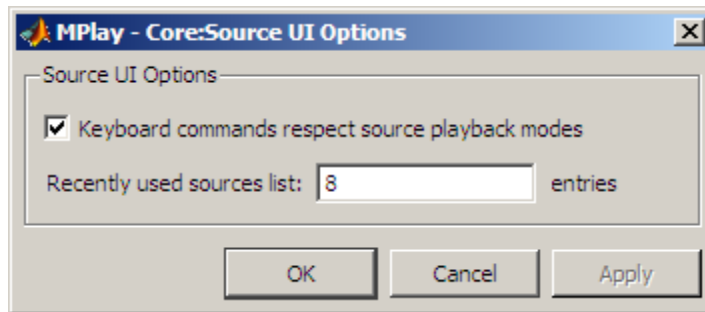


If you select the **Display the full source path in the title bar** check box, the full Simulink path appears in the title bar. Otherwise, the title bar displays a shortened name.

Use the **Message log opens when** 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.



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.

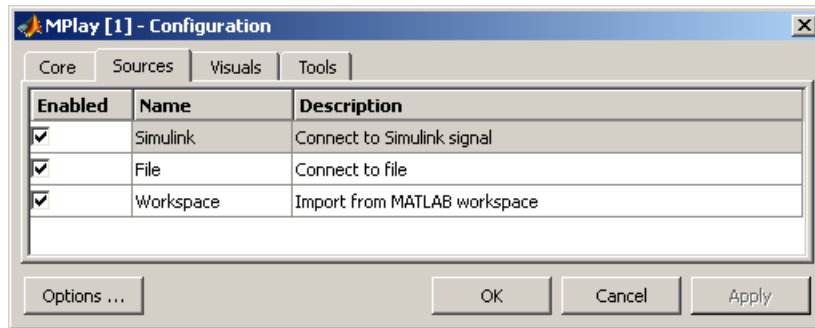
- 1 Select the **Keyboard commands respect playback modes** check box.
- 2 Select the **Backward playback** button.
 - Using the right keyboard arrow key moves the video backward, and using the left keyboard arrow key moves the video forward.
 - With MPlay set to play backwards, the keyboard “forward”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

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

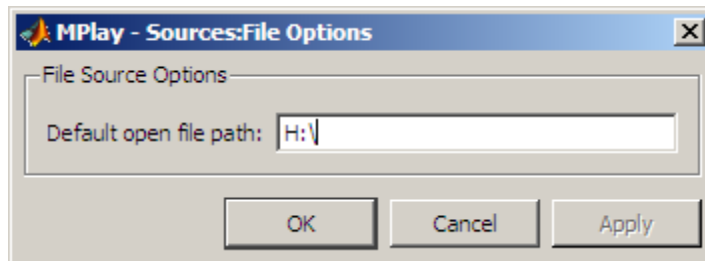
Use the **Recently used sources list** parameter to control the number of sources you see in the **File** menu.

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.

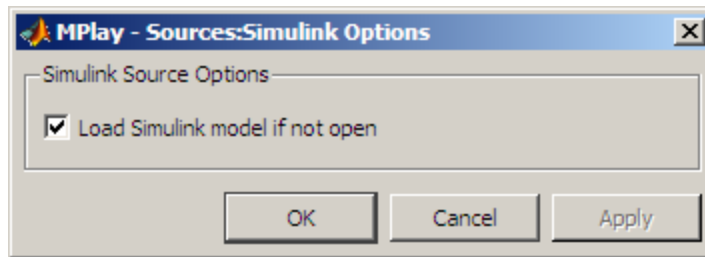


Click **File**, and then click the **Options** button to open the **Sources:File Options** dialog box.



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**.

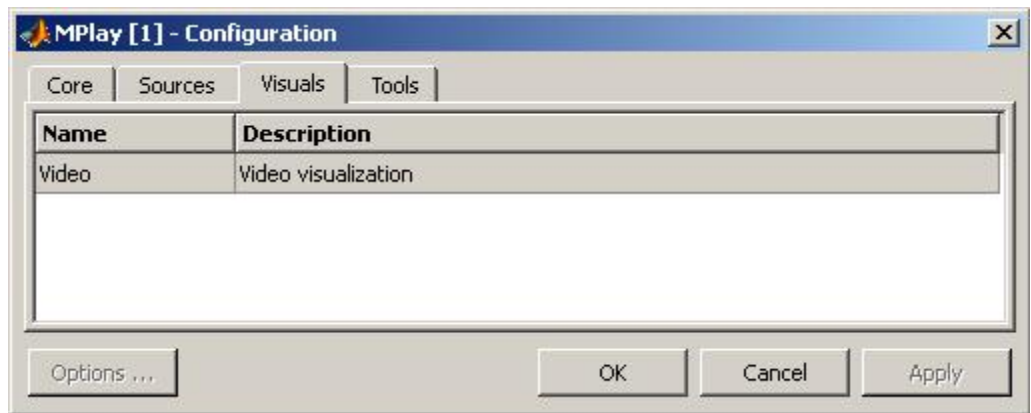
Click **Simulink**, and then click the **Options** button to open the **Sources:Simulink Options** dialog box.



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.

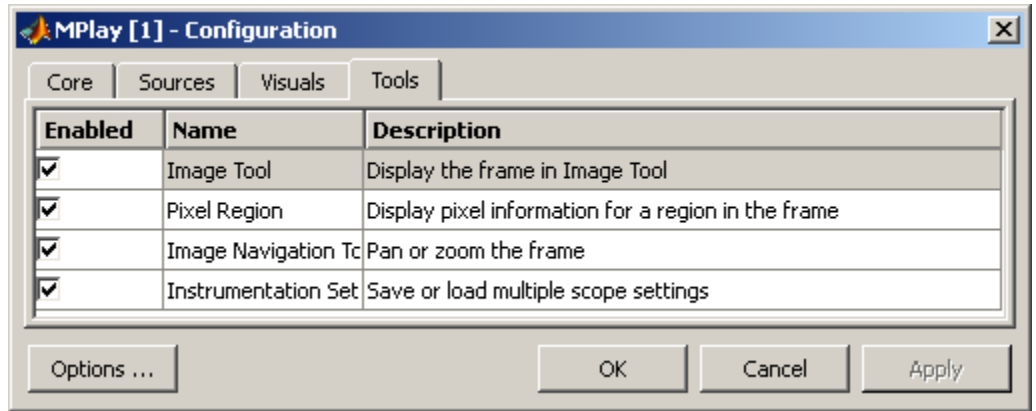
Configuration Visuals Pane

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

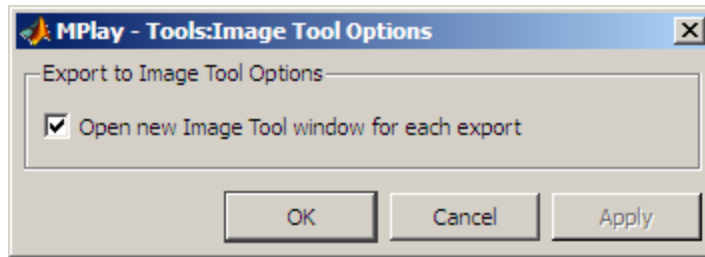


Configuration Tools Pane

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.



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



Select the **Open new Image Tool window for export** check box if you want to send each video frame to a different session of Image Tool.

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

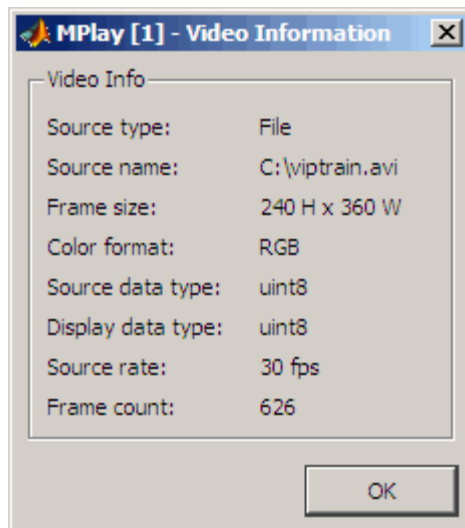
Select the **Image Navigation Tools** check box to enable the pan-and-zoom GUI button.

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.

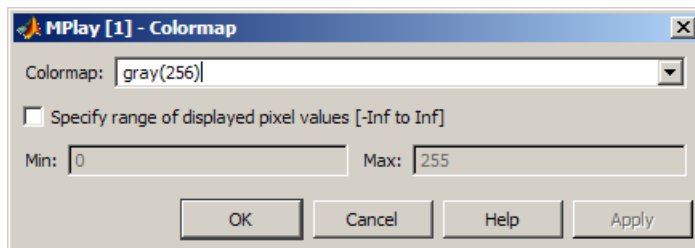
Video Information

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  .



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**.



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

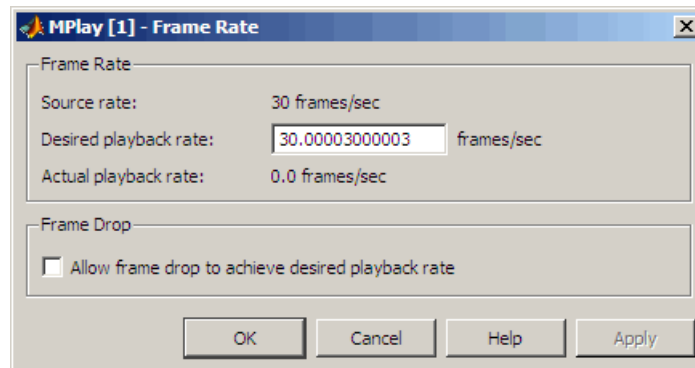
Sometimes, the pixel values do not use the entire data type range. In such cases, you can select the **Specify range of displayed pixel values** check box. You can then enter the range for your data. The dialog box automatically displays the range based on the data type of the pixel values.

Frame Rate

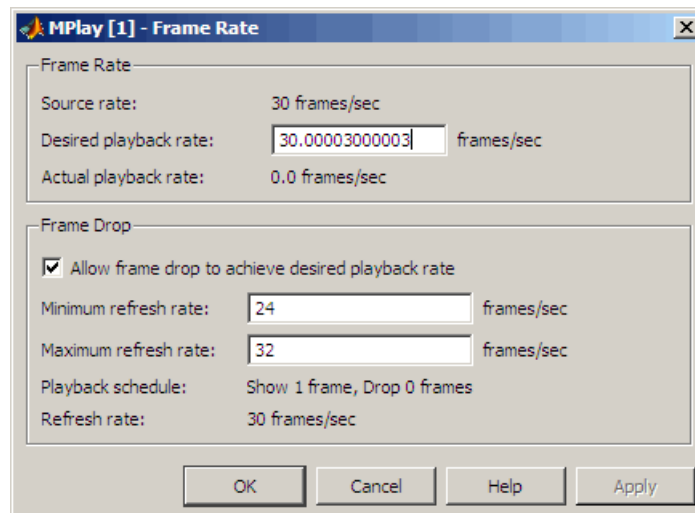
The Frame Rate dialog box displays the frame rate of the source. It also lets you change the rate at which the MPlay GUI plays the video and displays the actual playback rate.

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

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**.



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.



You can refine further both the quality of playback versus the 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 both of the refresh rates (which is how often the GUI updates the screen), to 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.

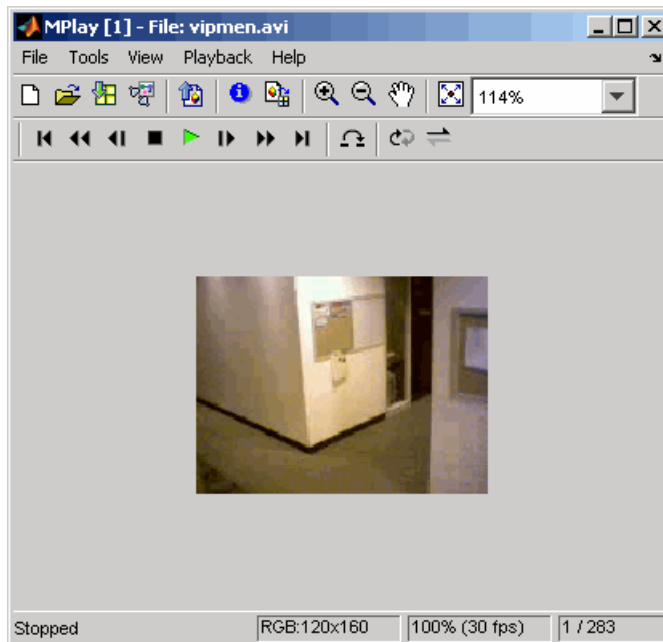
Viewing Video File Frames

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\vipblks\vipdemos\vipmen.avi`.

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



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

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

Analysis and Enhancement

- “Feature Extraction” on page 4-2
- “Image Enhancement” on page 4-30
- “Template Matching” on page 4-67
- “Pixel Statistics” on page 4-73

Feature Extraction

In this section...
“Finding Edges in Images” on page 4-2
“Finding Lines in Images” on page 4-9
“Measuring an Angle Between Lines” on page 4-18

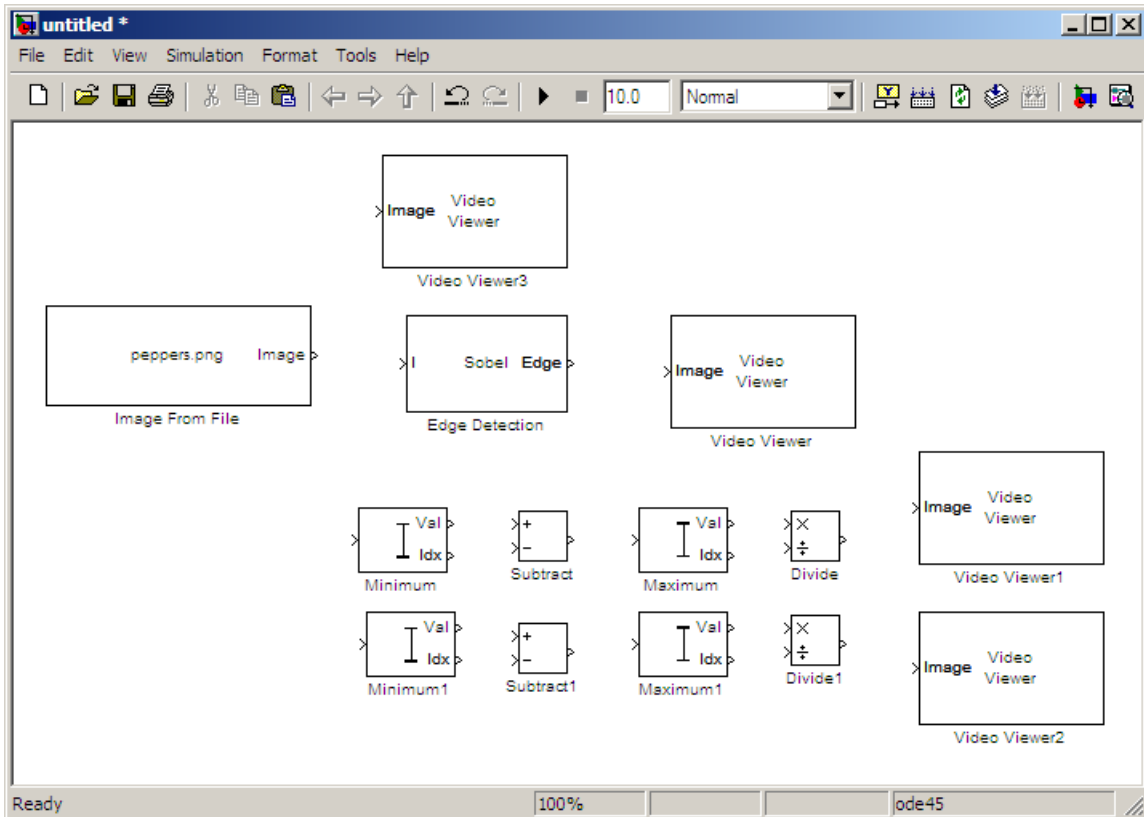
Finding Edges in Images

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.

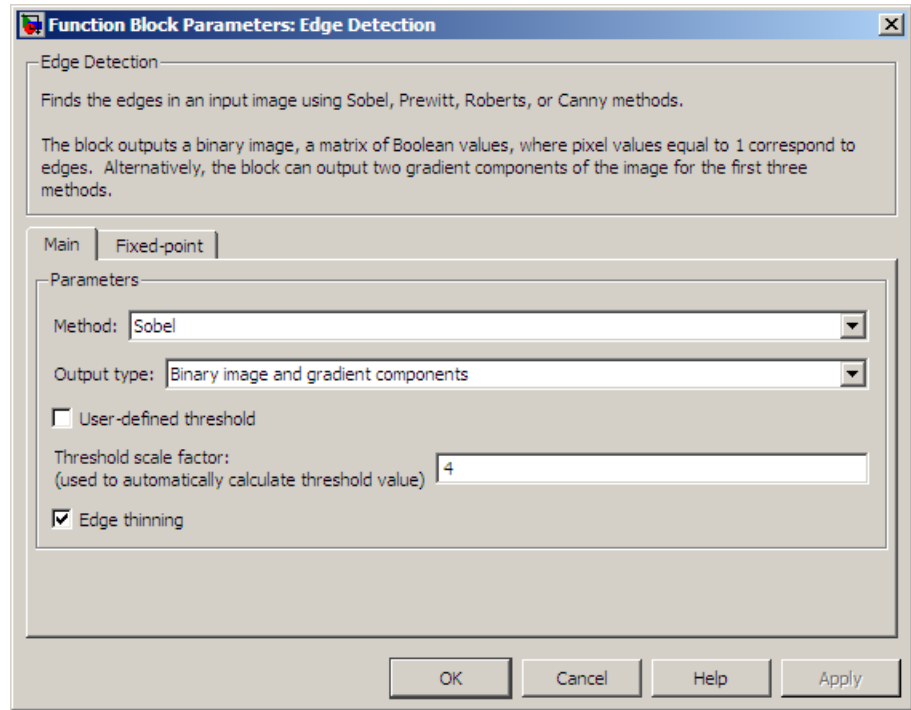
Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Edge Detection	Video and Image Processing Blockset > Analysis & Enhancement	1
Minimum	Video and Image Processing Blockset > Statistics	2
Maximum	Video and Image Processing Blockset > Statistics	2
Video Viewer	Video and Image Processing Blockset > Sinks	4
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.

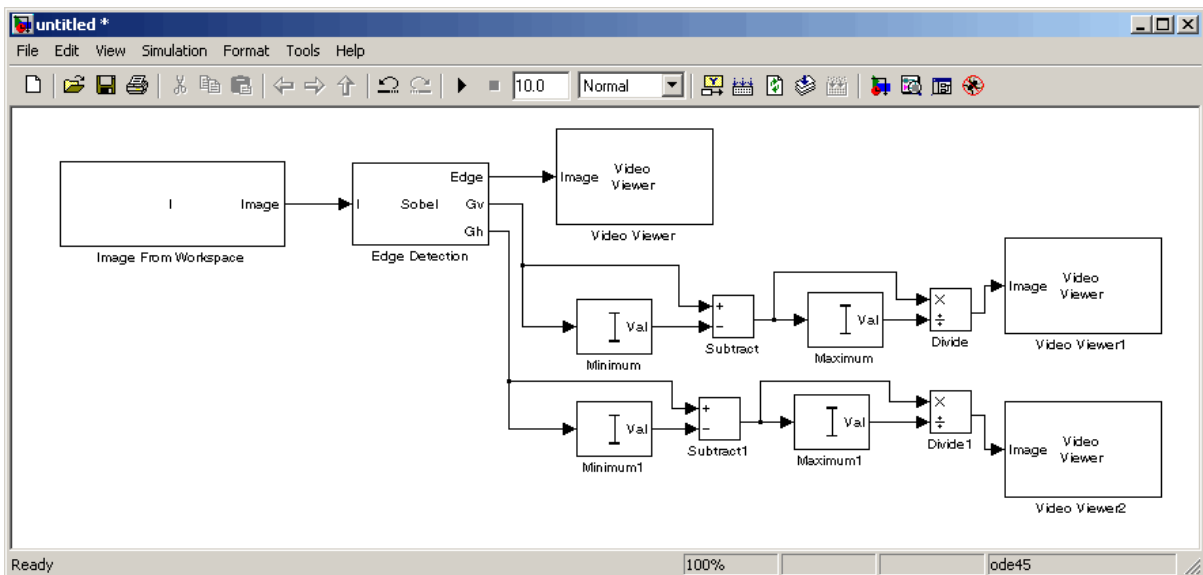


The Edge Detection block convolves the input matrix with the Sobel kernel. This calculates the gradient components of the image that correspond to the horizontal and vertical edge responses. The block outputs these components at the **Gh** and **Gv** ports, respectively. Then the block performs a thresholding 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

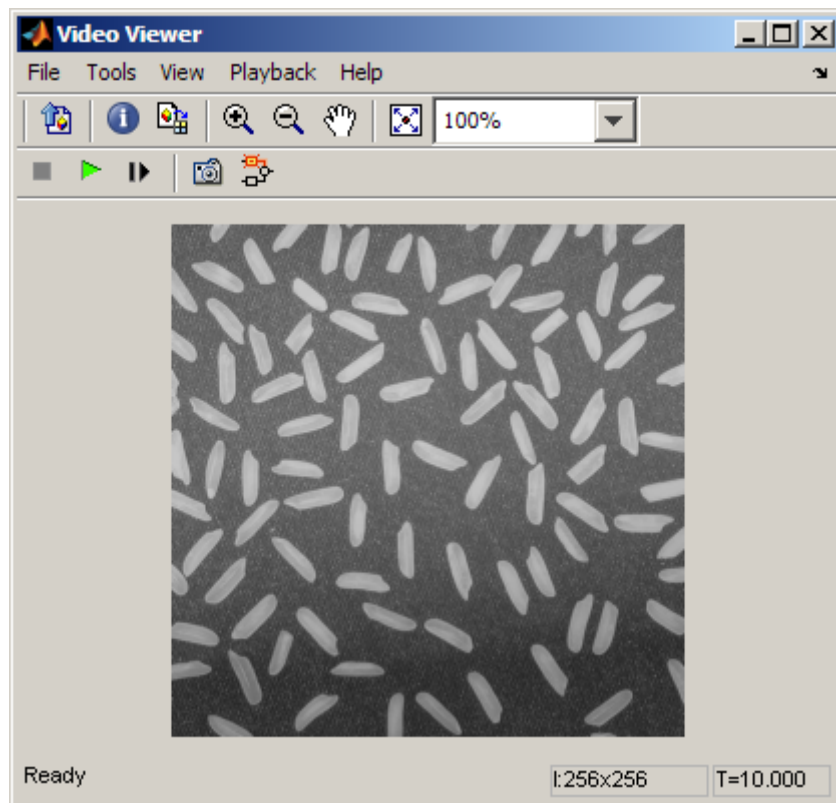
need to be scaled between 0 and 1 in order to display them using the Video Viewer blocks. This is done with the Statistics and Math Operation blocks.

- 7 Use the Minimum blocks to find the minimum value of Gv and Gh matrices. Set the **Mode** parameters to Value.
- 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.
- 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 Viewer1 and Video Viewer2 blocks. Accept the default parameters.
- 12 Connect the blocks as shown in the following figure.

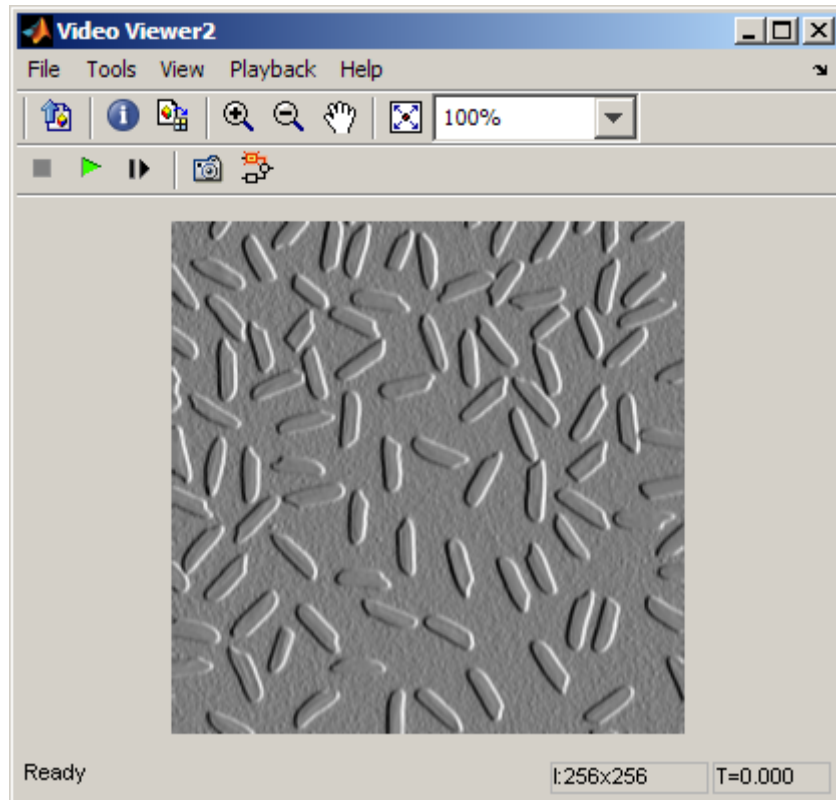


- 13 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
 - **Solver** pane, **Stop time** = 0
 - **Solver** pane, **Type** = Fixed-step
 - **Solver** pane, **Solver** = Discrete (no continuous states)
- 14 Run your model.

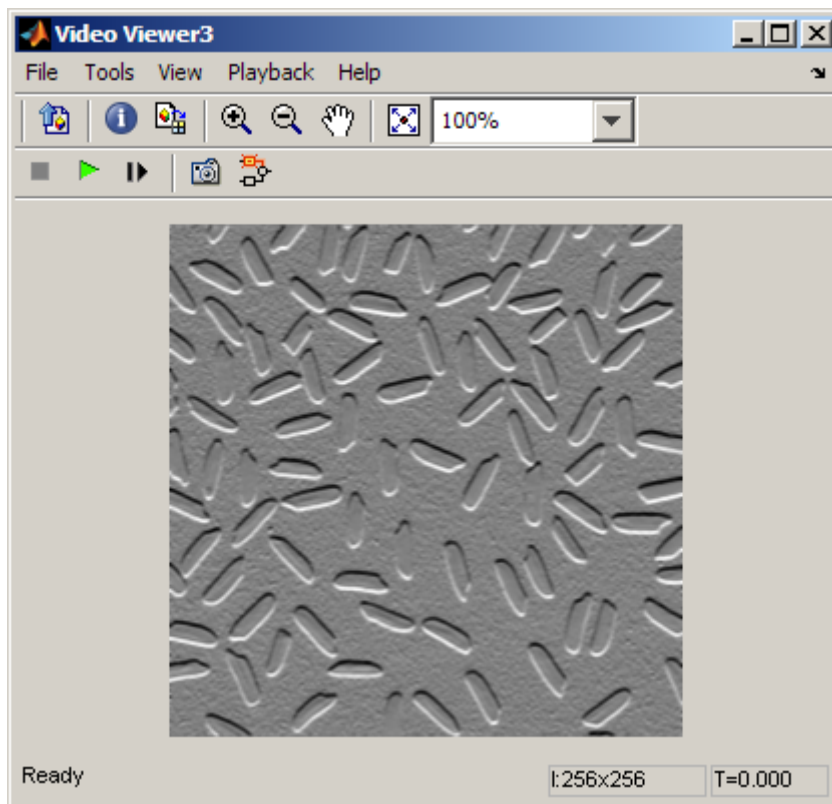
The Video Viewer window displays the original image. The Video Viewer1 window displays the edges of the rice grains in white and the background in black.



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.



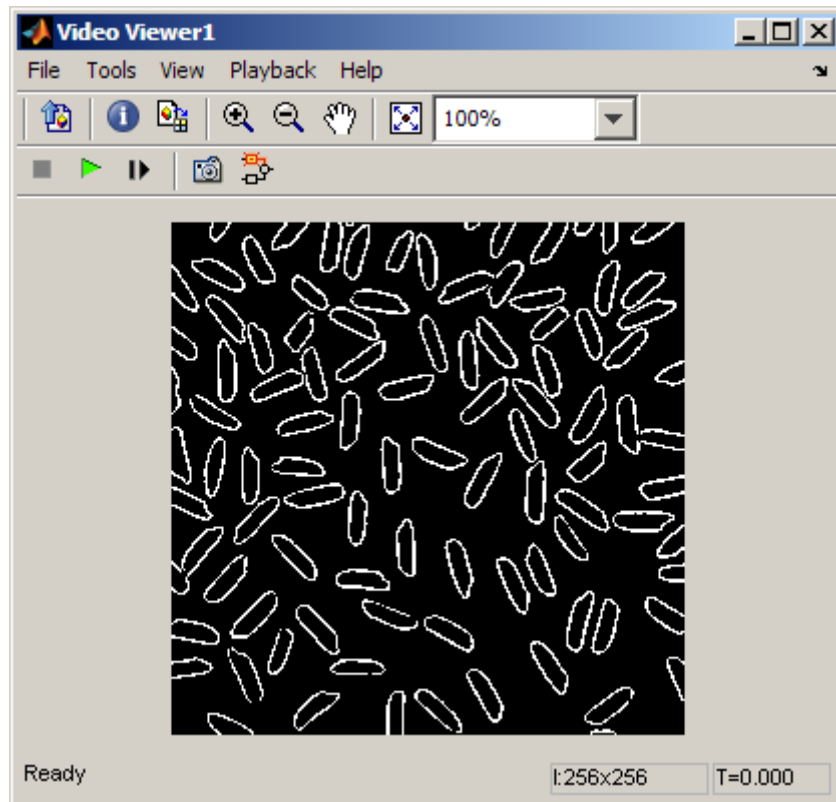
The Video Viewer3 window displays the intensity image of the horizontal gradient components of the image. In this image, the horizontal edges of the rice grains are more well defined.



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 *Video and Image Processing Blockset Reference*.

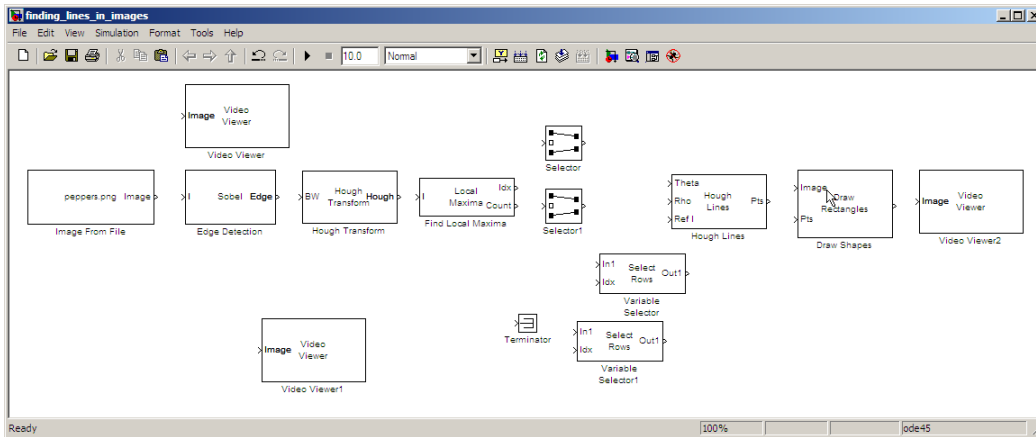
Finding Lines in Images

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

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

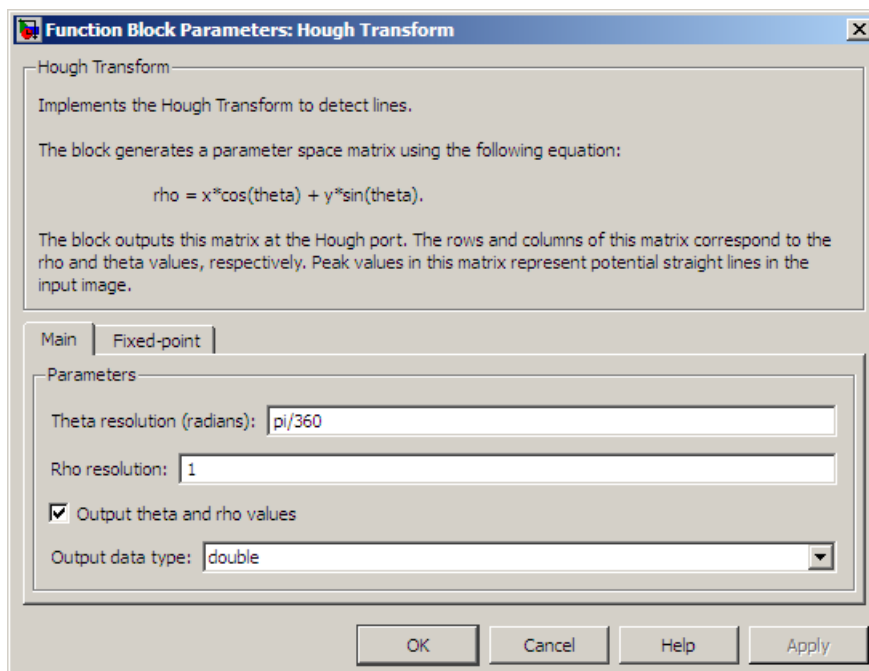
Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Edge Detection	Video and Image Processing Blockset > Analysis & Enhancement	1
Hough Transform	Video and Image Processing Blockset > Transforms	1
Find Local Maxima	Video and Image Processing Blockset > Statistics	1
Hough Lines	Video and Image Processing Blockset > Transforms	1
Draw Shapes	Video and Image Processing Blockset > Text & Graphics	1
Video Viewer	Video and Image Processing Blockset > Sinks	2
Variable Selector	Signal Processing Blockset > Signal Management > Indexing	2
Selector	Simulink > Signal Routing	2
Terminator	Simulink > Sinks	1

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

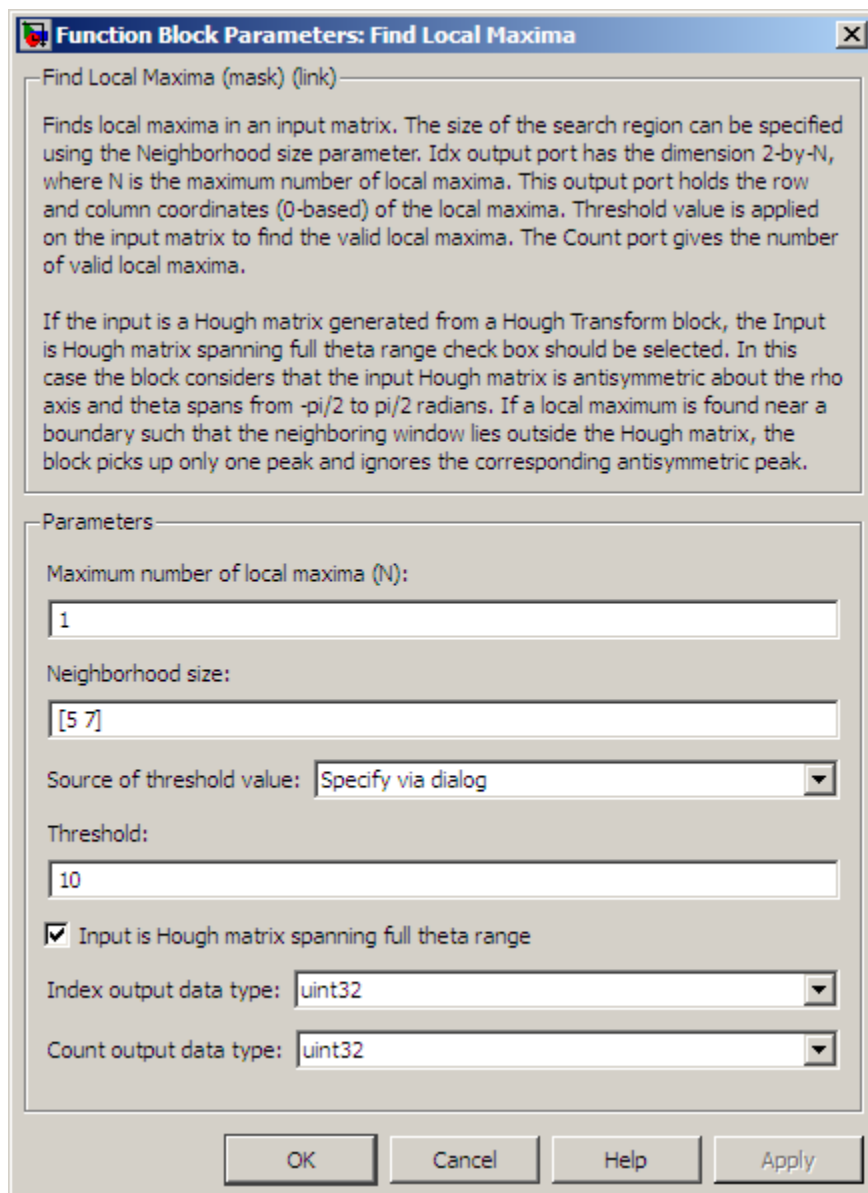


You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

- 3** Use the Image From File block to import your image. Set the **File name** parameter to `circuit.tif`.
- 4** Use the Edge Detection block to find the edges in the intensity image. This process improves the efficiency of the Hough Lines block as it reduces 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. Accept the default parameters.
- 5** Use the Video Viewer block to display the original image and the Video Viewer1 block to display the edges found by the Edge Detection block. Accept the default parameters.
- 6** Use the Hough Transform block to compute 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. Set the block parameters as follows:
 - **Theta resolution (radians) = $\pi/360$**
 - Select the **Output theta and rho values** check box.



- 7 Use the Find Local Maxima block to find the location of the maximum value in the Hough matrix. Set the block parameters as follows:
 - **Maximum number of local maxima (N) = 1**
 - Select the **Input is Hough matrix spanning full theta range** check box.



8 Use the Selector blocks to separate the indices of the rho and theta values, which are output at the Idx port, that are associated with the maximum value in the Hough matrix. Set the Selector block parameters as follows:

- **Number of input dimensions:** 1
- **Index mode** = Zero-based
 - **Index Option** = Index vector (dialog)
 - **Index** = 0
- **Input port size** = 2

Set the Selector1 block parameters as follows:

- **Number of input dimensions:** 1
- **Index mode** = Zero-based
 - **Index Option** = Index vector (dialog)
 - **Index** = 1
- **Input port size** = 2

9 Use the Variable Selector blocks to index into the rho and theta vectors and determine the rho and theta values that correspond to the longest line in the original image. Set the parameters of the Variable Selector blocks as follows:

- **Select** = Columns
- **Index mode** = Zero-based

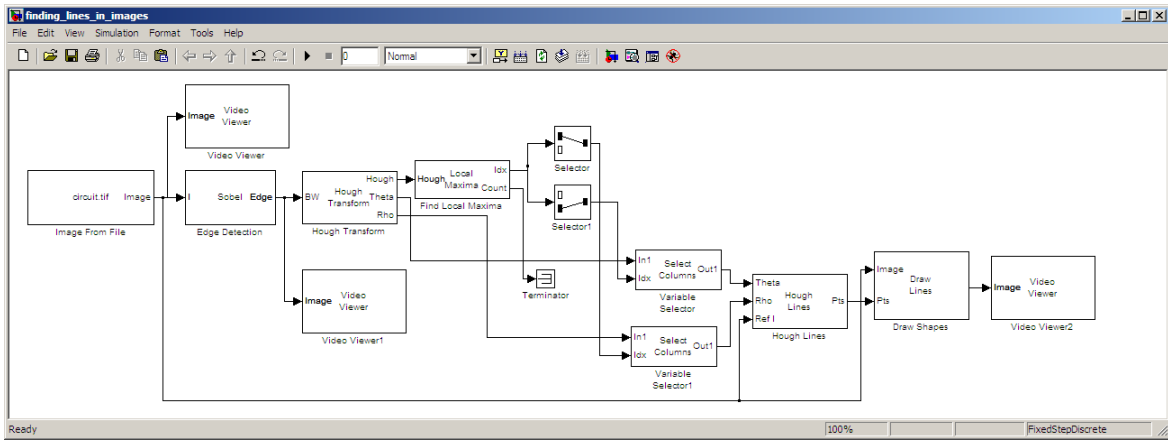
10 Use the Hough Lines block to determine where the longest line intersects the edges of the original image. You use these coordinates to superimpose a white line on the original image. Set the **Sine value computation method** to Trigonometric function.

11 Use the Draw Shapes block to draw a white line over the longest line on the original image. Set the block parameters as follows:

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

12 Use the Video Viewer2 block to display the original image with a white line superimposed over the longest line in the image. Accept the default parameters.

13 Connect the blocks as shown in the following figure.

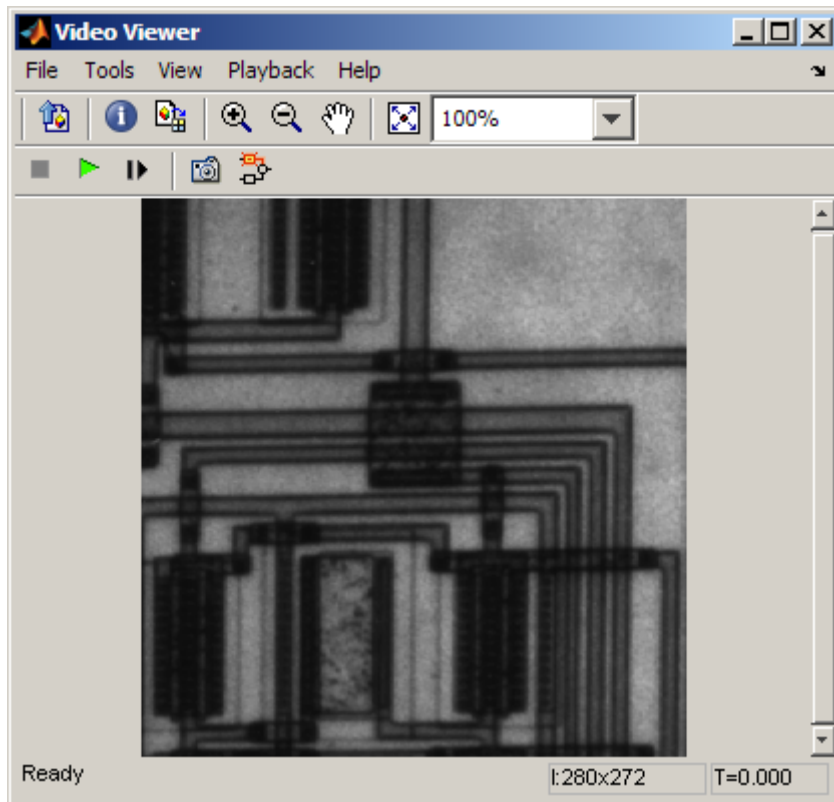


14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

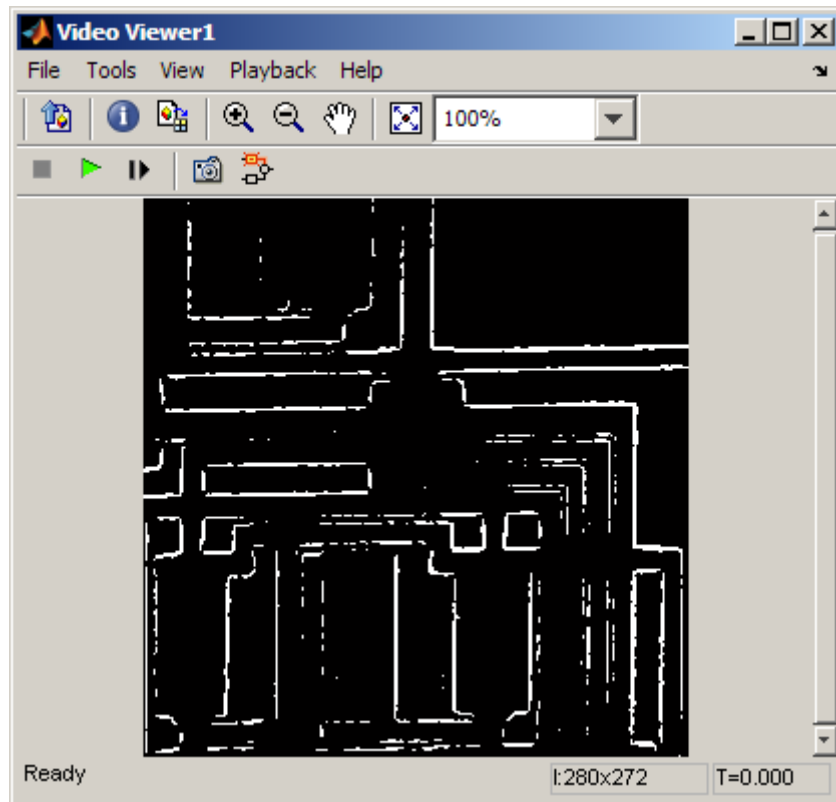
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

15 Run your model.

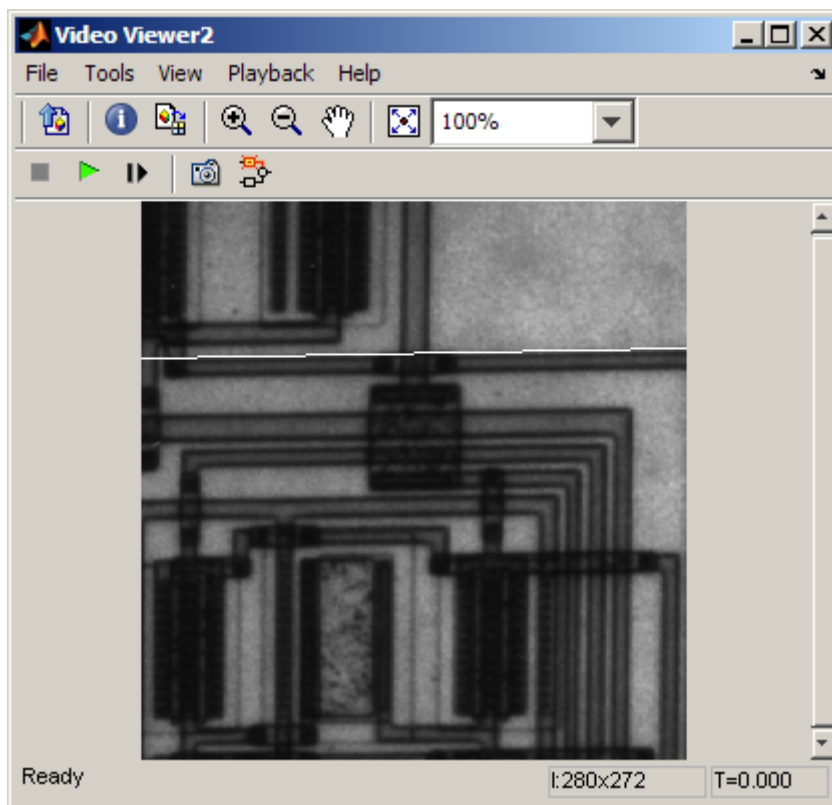
The Video Viewer window displays the original image.



Video Viewer1 displays the edges found in the original image in white and the background in black.



The Video Viewer2 window displays the original image with a white line drawn over the longest line in the image.



You have now used the Hough Transform, Find Local Maxima, and Hough Lines blocks to find the longest line in an image. For more information on these blocks, see the Hough Transform, Find Local Maxima, and Hough Lines block reference pages in the *Video and Image Processing Blockset Reference*. For additional examples of the techniques used in this section, see the Lane Departure Warning System and Rotation Correction demos. You can open these demos by typing `vipldws` and `viphough` at the MATLAB command prompt.

Measuring an Angle Between Lines

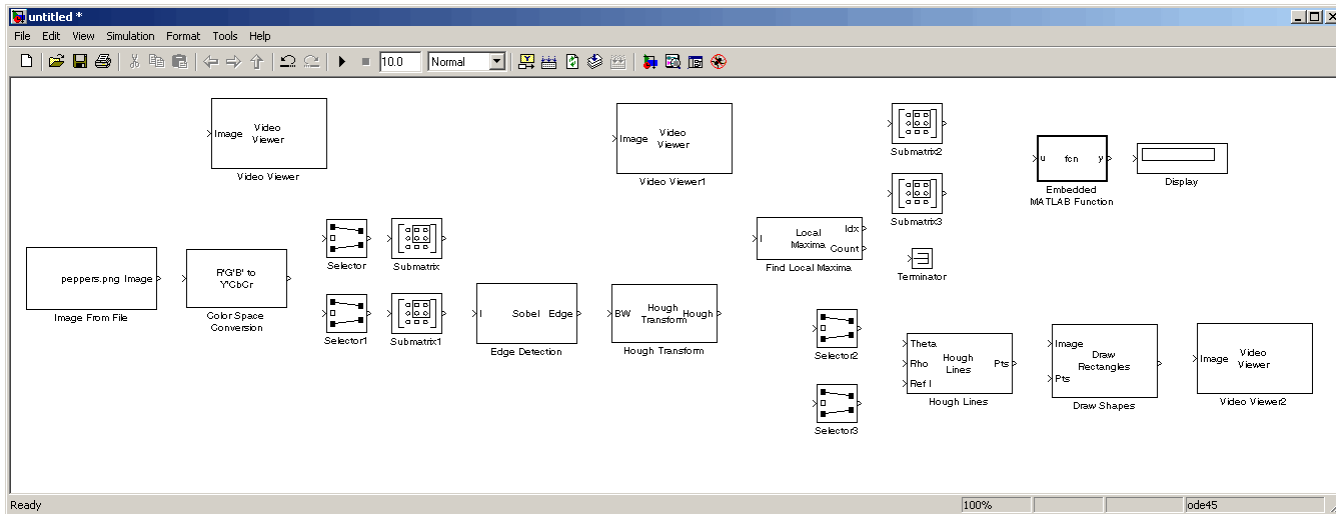
The Hough Transform, Find Local Maxima, and Hough Lines blocks enable you to find lines in images. With the Draw Shapes block, you can annotate

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	Video and Image Processing Blockset > Sources	1
Color Space Conversion	Video and Image Processing Blockset > Conversions	1
Edge Detection	Video and Image Processing Blockset > Analysis & Enhancement	1
Hough Transform	Video and Image Processing Blockset > Transforms	1
Find Local Maxima	Video and Image Processing Blockset > Statistics	1
Draw Shapes	Video and Image Processing Blockset > Text & Graphics	1
Video Viewer	Video and Image Processing Blockset > Sinks	3
Hough Lines	Video and Image Processing Blockset > Transforms	1
Submatrix	Signal Processing Blockset > Math Functions > Matrices and Linear Algebra > Matrix Operations	4
Terminator	Simulink > Sinks	1
Selector	Simulink > Signal Routing	4
Embedded 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 YCbCr 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 YCbCr 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** = Zero-based

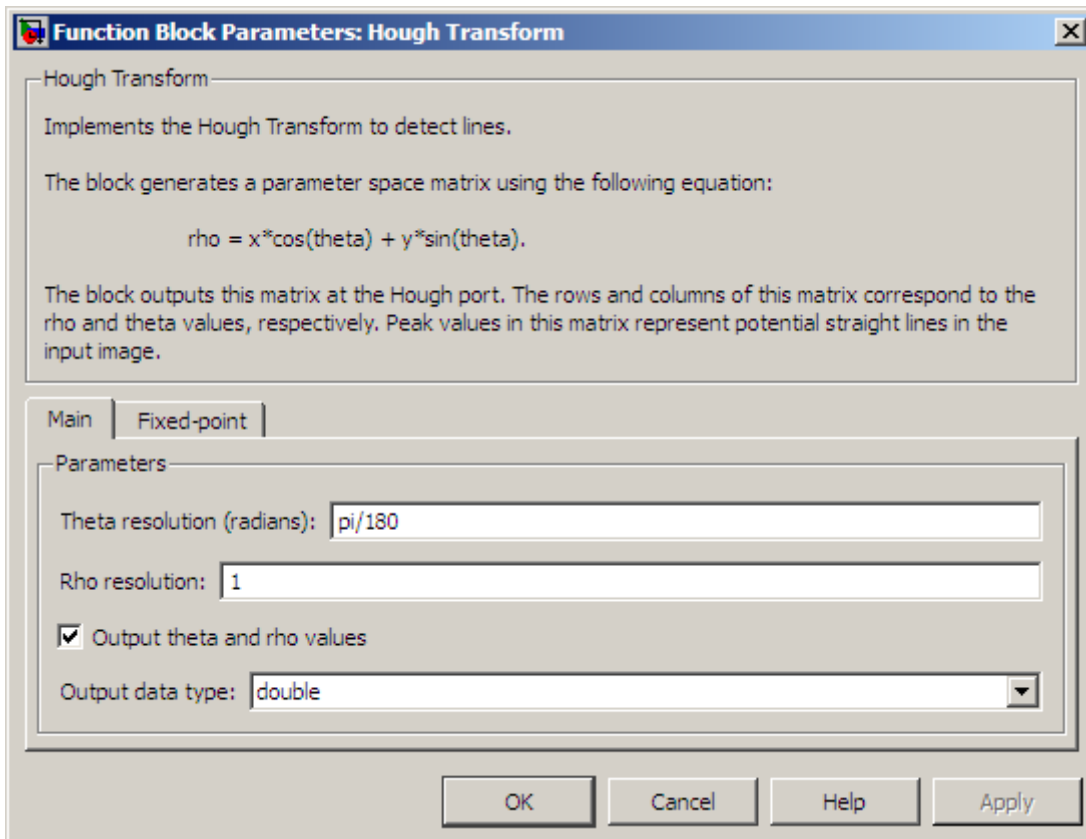
- 1
 - **Index Option** = Select all
- 2
 - **Index Option** = Select all
- 3
 - **Index Option** = Index vector (dialog)
 - **Index** = 1

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

- **Number of input dimensions** = 3
 - **Index mode** = Zero-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.



- 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 zero-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 zero-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:

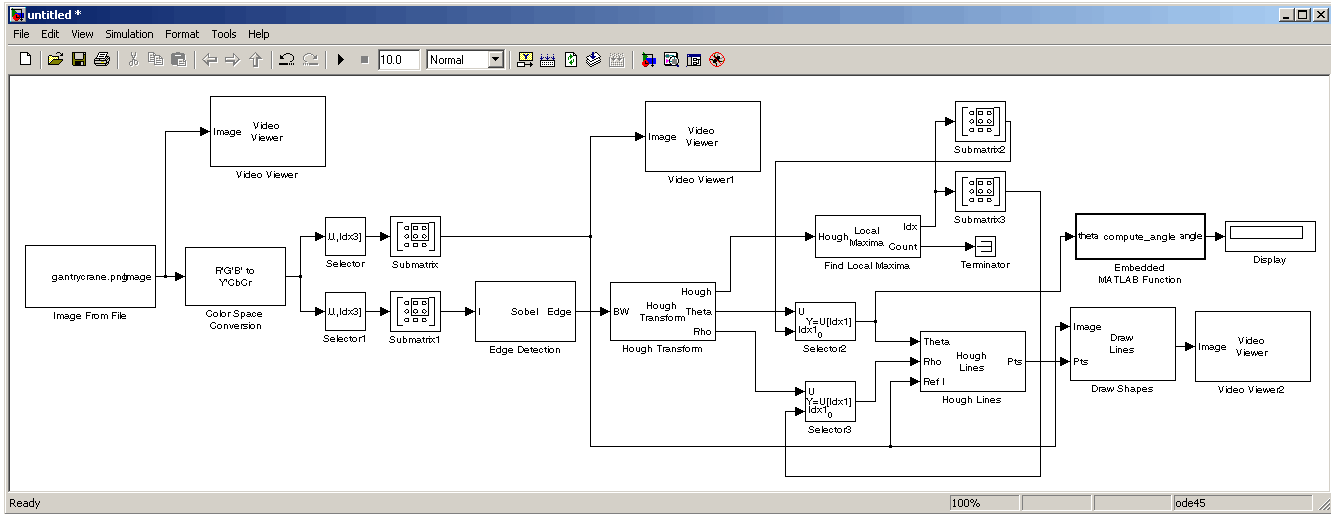
- **Index mode** = Zero-based
- **1**
 - **Index Option** = Index vector (port)
- **Input port size** = -1

You set the **Index mode** to Zero-based because the Find Local Maxima block outputs zero-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 Embedded 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.

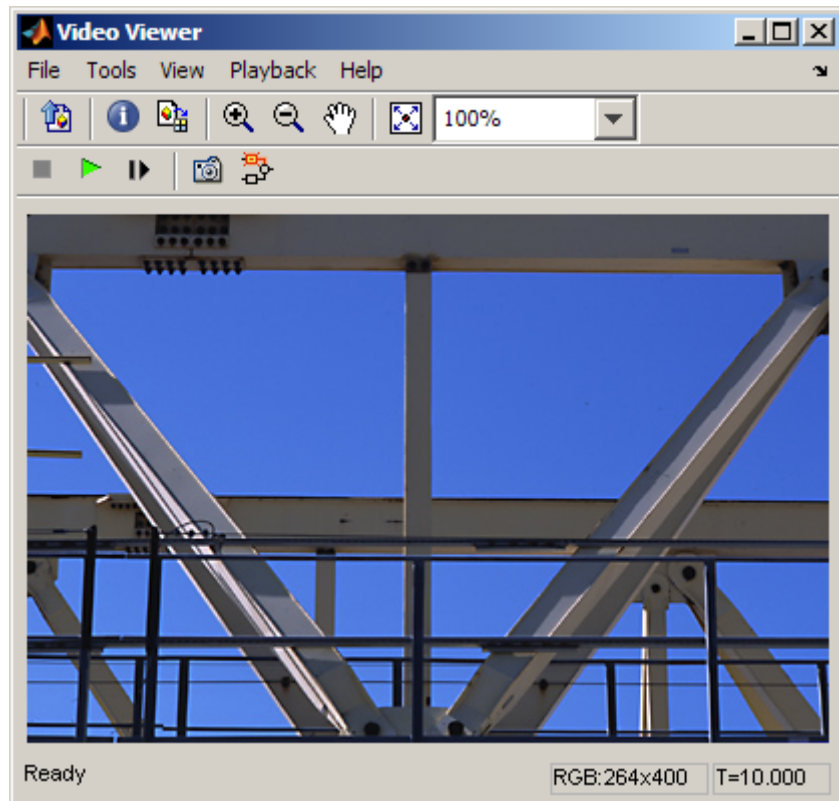


19 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

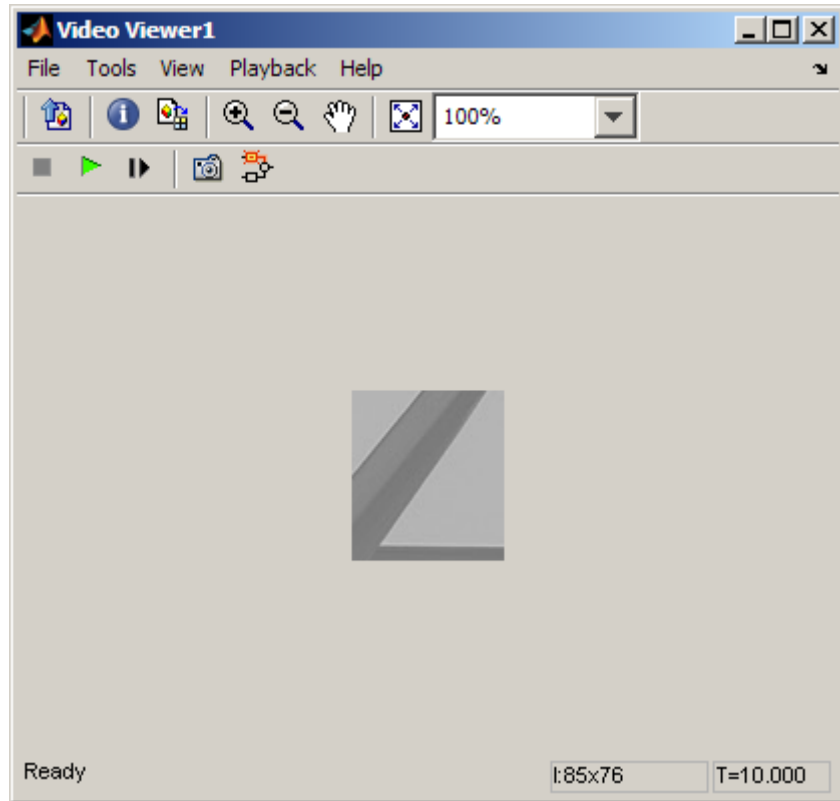
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

20 Run the model.

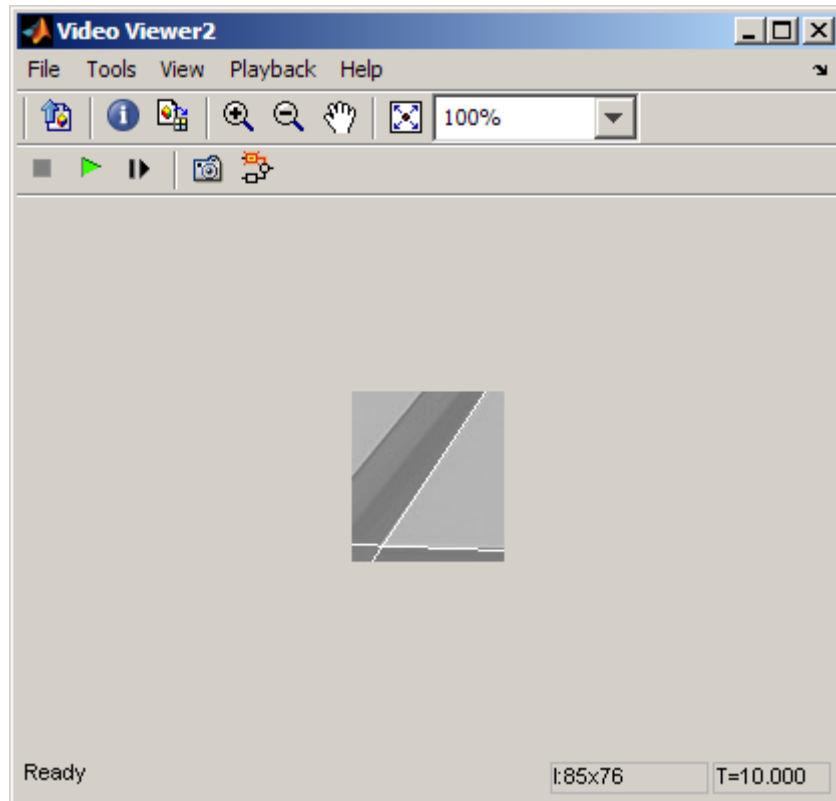
The Video Viewer window displays the original image.



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 in the *Video and Image Processing Blockset Reference*.

Image Enhancement

In this section...

“Sharpening and Blurring an Image” on page 4-30

“Removing Salt and Pepper Noise from Images” on page 4-39

“Removing Periodic Noise from Video” on page 4-45

“Adjusting the Contrast in Intensity Images” on page 4-54

“Adjusting the Contrast in Color Images” on page 4-61

Sharpening and Blurring 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 and blur an image. The prime notation indicates that the signals are gamma corrected.

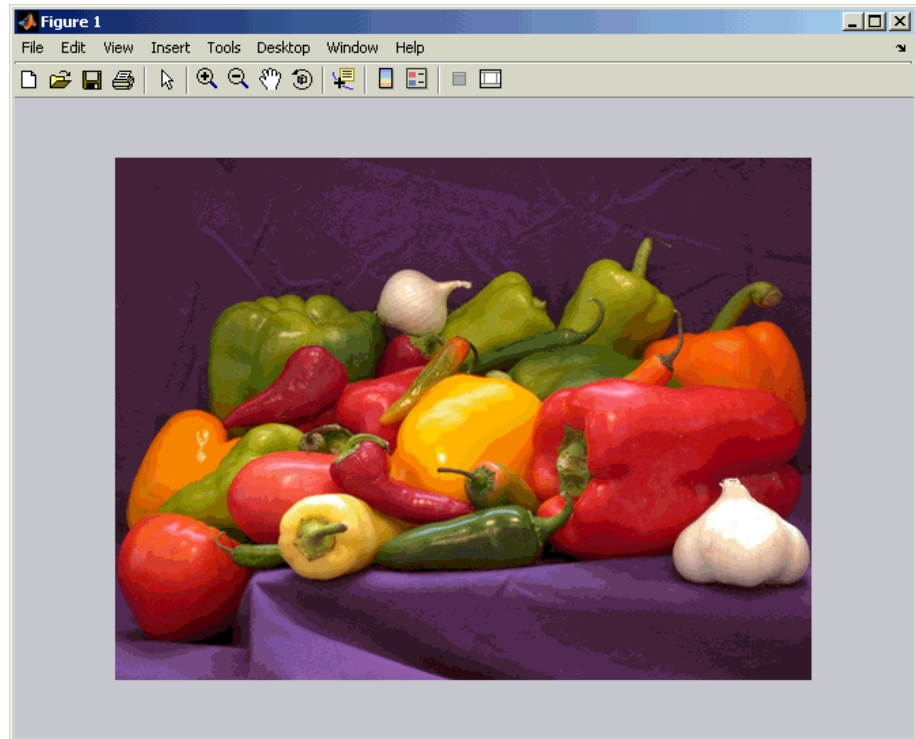
- 1 Define an R'G'B' image in the MATLAB workspace. To read in an R'G'B' image from a PNG file and cast it to the double-precision data type, at the MATLAB command prompt, type

```
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)
```

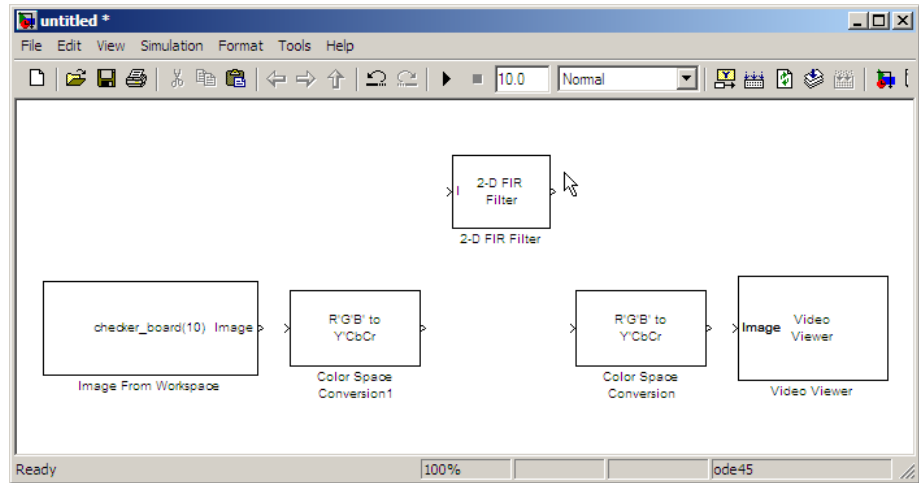


Now that you have defined your image, you can create your model.

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

Block	Library	Quantity
Image From Workspace	Video and Image Processing Blockset > Sources	1
Color Space Conversion	Video and Image Processing Blockset > Conversions	2
2-D FIR Filter	Video and Image Processing Blockset > Filtering	1
Video Viewer	Video and Image Processing Blockset > Sinks	1

4 Position the blocks as shown in the following figure.

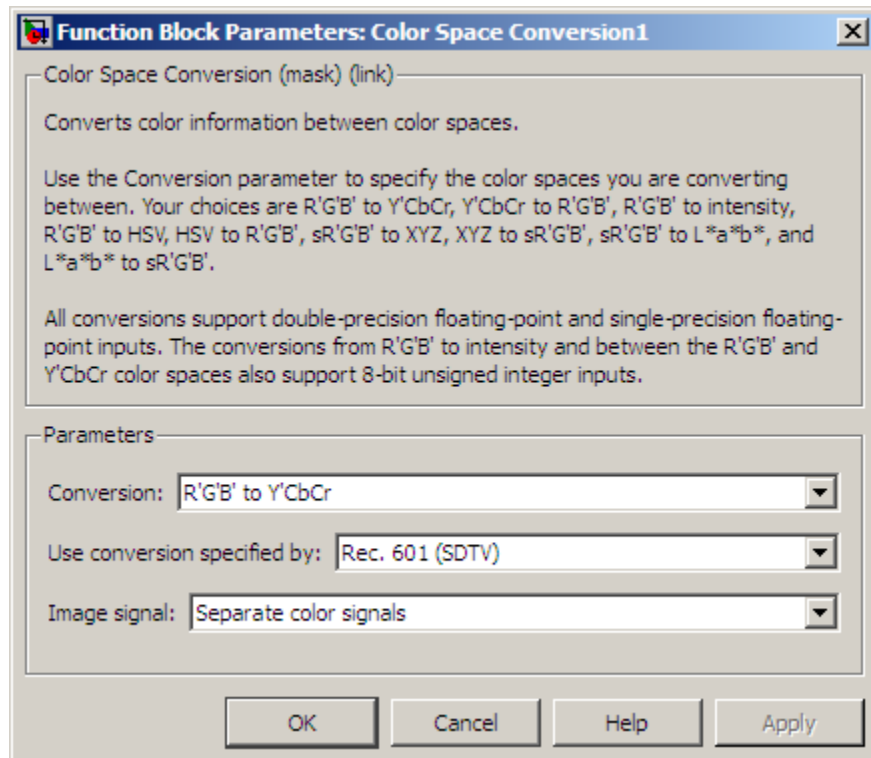


5 Use the Image From Workspace block to import the R'G'B' image from the MATLAB workspace. Set the parameters as follows:

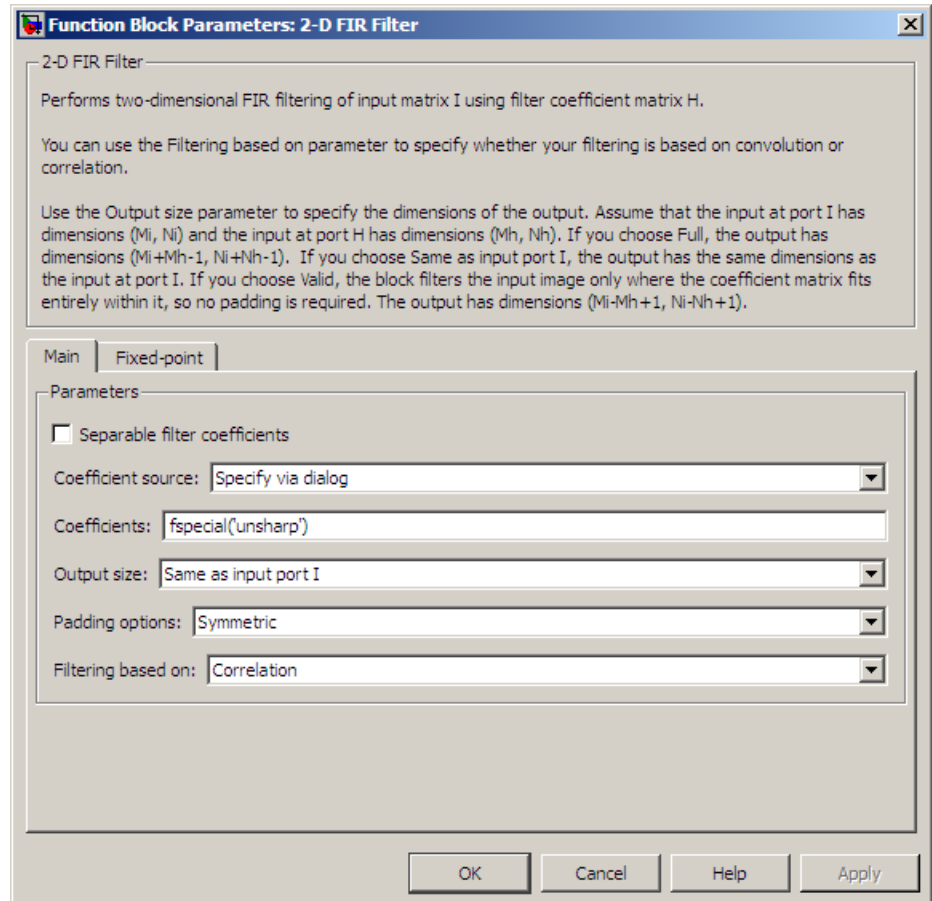
- **Main pane, Value** = I
- **Main pane, Image signal** = Separate color signals

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

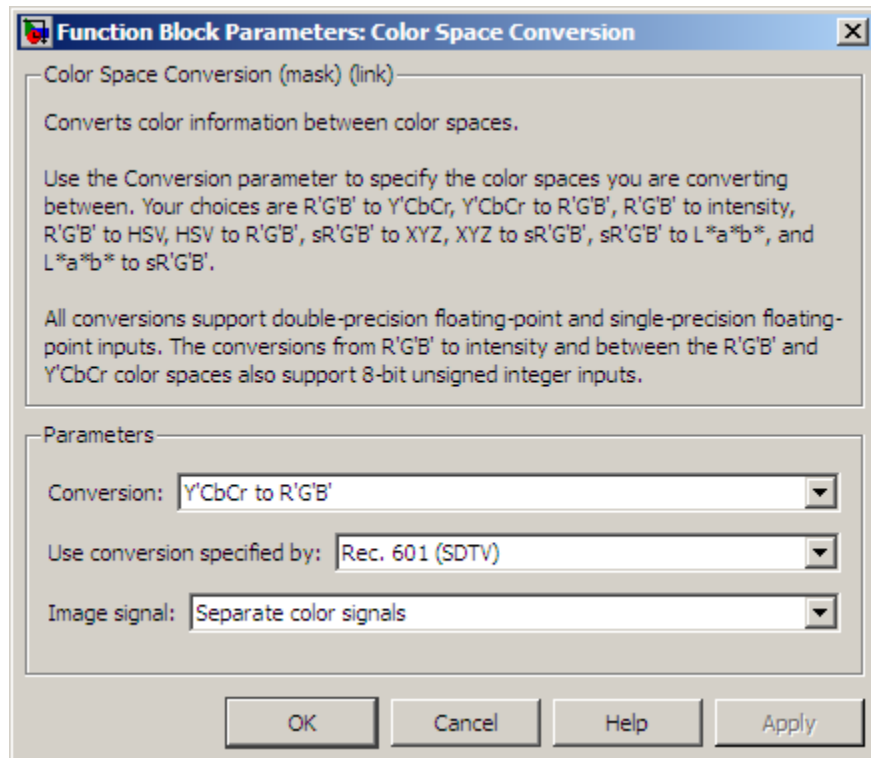


- 7 Use the 2-D FIR Filter block to filter the luma portion of the image. Set the block parameters as follows:
- **Coefficients** = `fspecial('unsharp')`
 - **Output size** = Same as input port I
 - **Padding options** = Symmetric
 - **Filtering based on** = Correlation

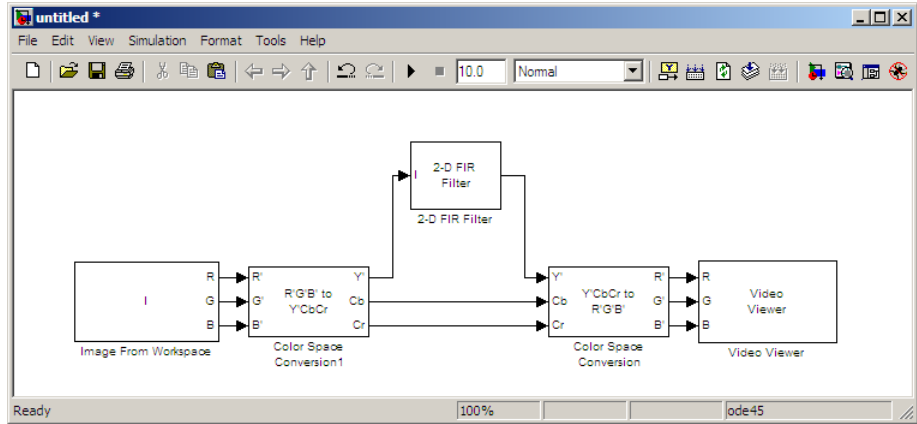


The `fspecial('unsharp')` command creates two-dimensional highpass filter coefficients suitable for correlation. This highpass filter sharpens the image by removing the low frequency noise in it.

- 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



- 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.

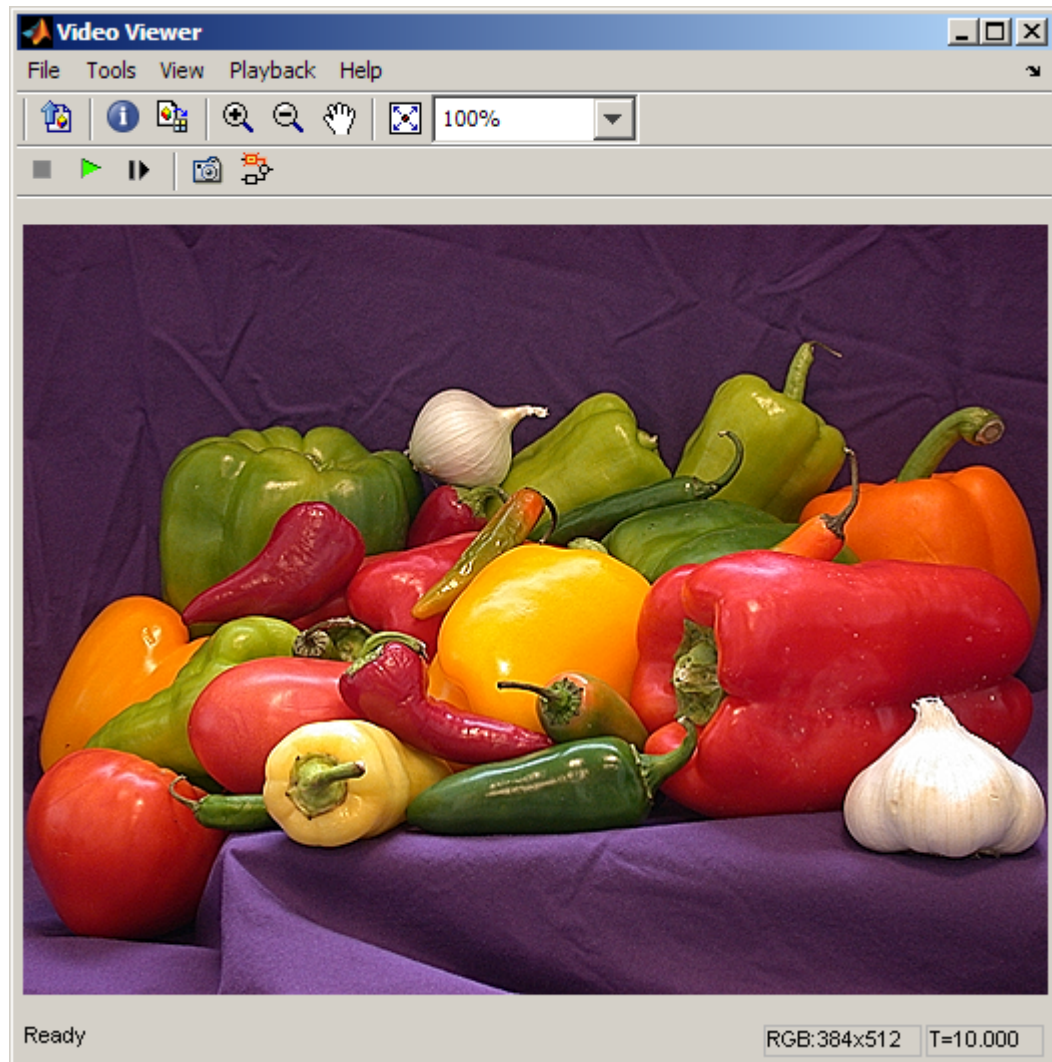


11 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **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.

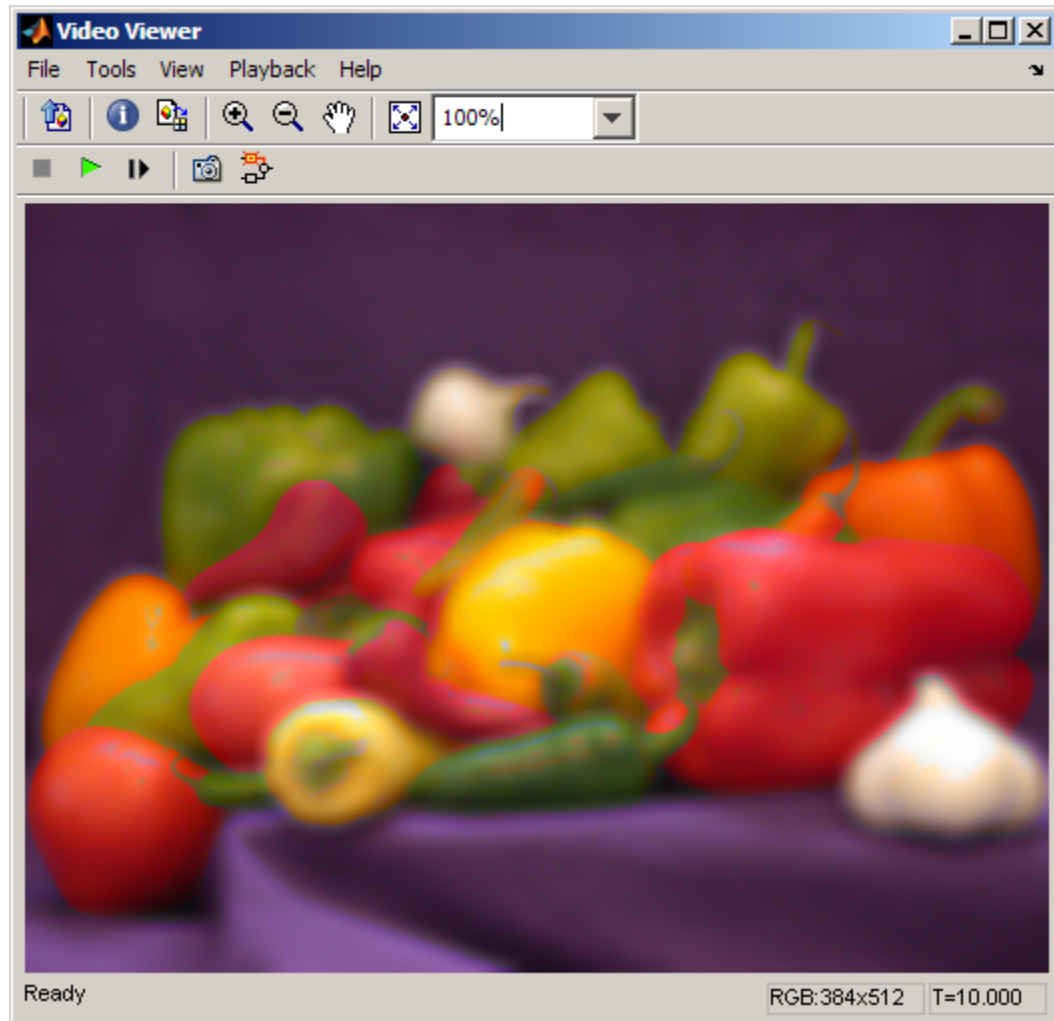


- 13** To blur the image, double-click the 2-D FIR Filter block. Set **Coefficients** parameter to `fspecial('gaussian',[15 15],7)` and then click **OK**.

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.

14 Run the model.

A blurred version of the original image appears in the Video Viewer window. The following image is shown at its true size.



In this example, you used the Color Space Conversion and 2-D FIR Filter blocks to sharpen and blur an image. For more information on these blocks,

see the Color Space Conversion and 2-D FIR Filter block reference pages in the *Video and Image Processing Blockset Reference*. For more information on the `fspecial` function, see the Image Processing Toolbox documentation.

Removing 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:

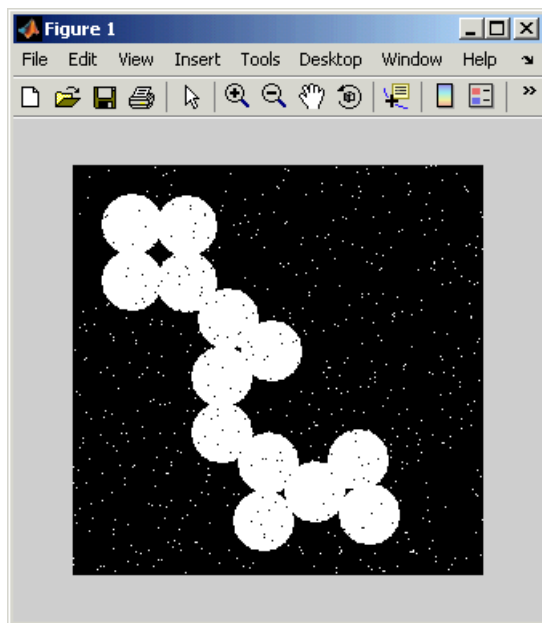
- 1 Define an intensity image in the MATLAB workspace and add noise to it by typing the following at the MATLAB command prompt:

```
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)
```

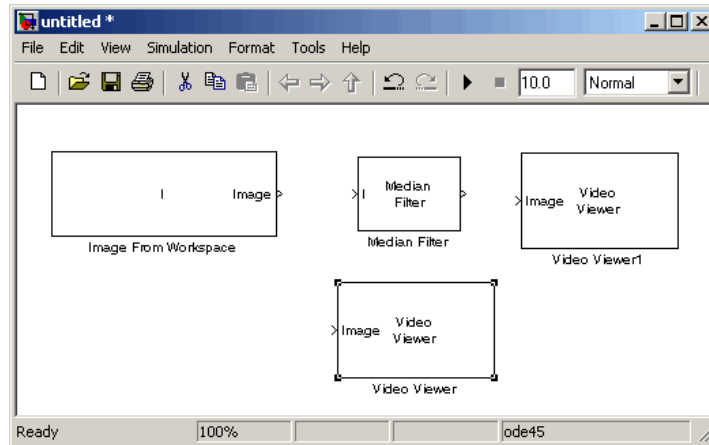


The intensity image contains noise that you want your model to eliminate.

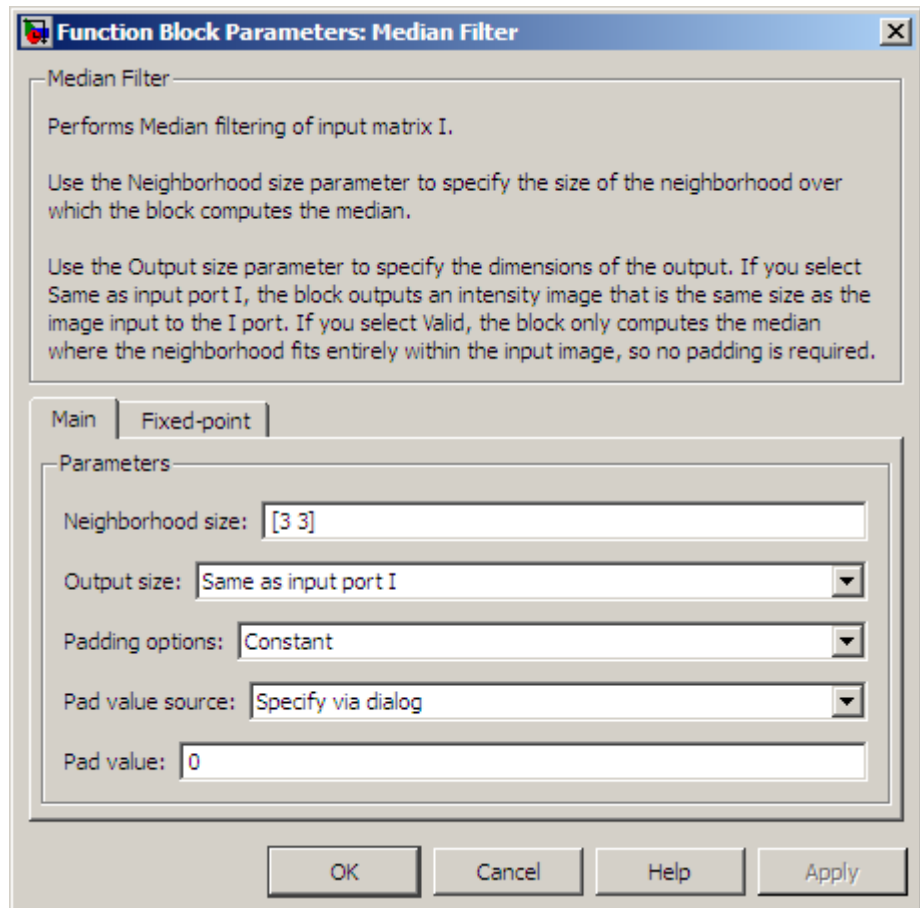
3 Create a Simulink model, and add the blocks shown in the following table.

Block	Library	Quantity
Image From Workspace	Video and Image Processing Blockset > Sources	1
Median Filter	Video and Image Processing Blockset > Filtering	1
Video Viewer	Video and Image Processing Blockset > Sinks	2

4 Position the blocks as shown in the following figure.

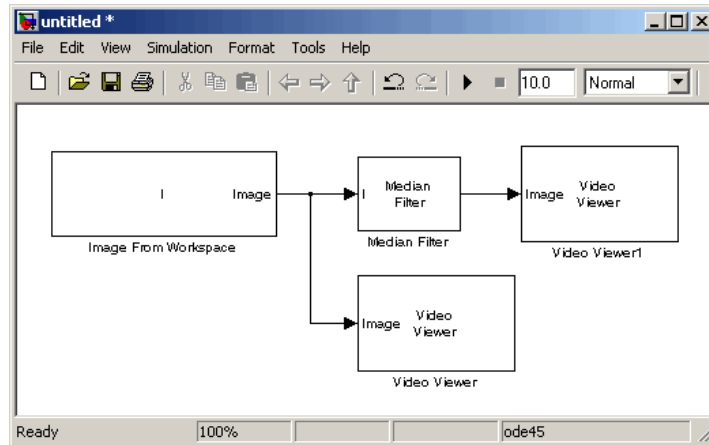


- 5 Use the Image From Workspace block to import the noisy image into your model. Set the **Value** parameter to I.
- 6 Use the Median Filter block to eliminate the black and white speckles in the image. Use the default parameters.



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.

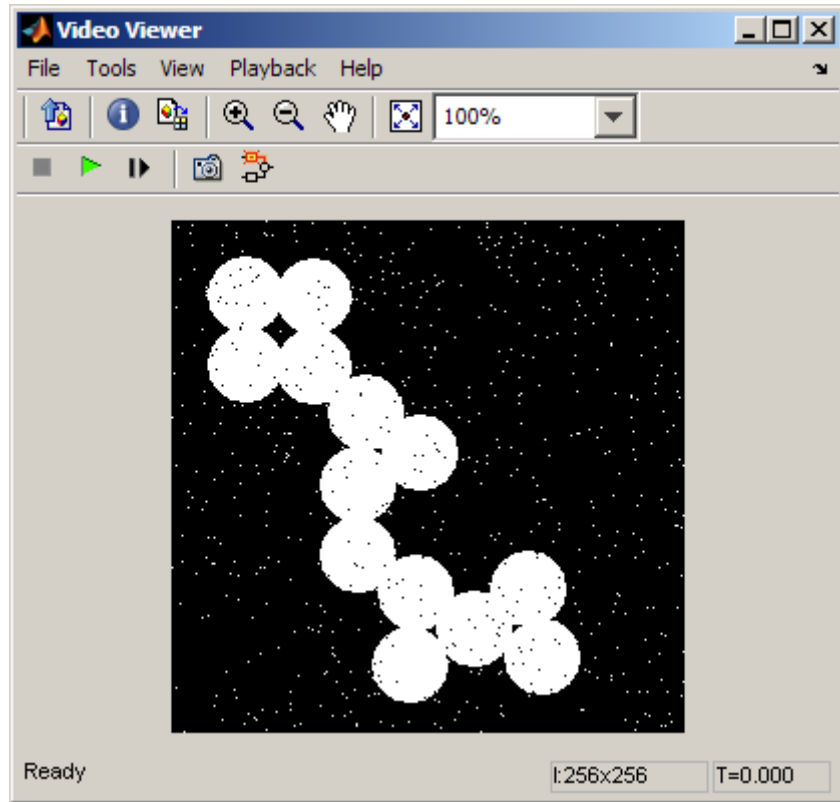


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

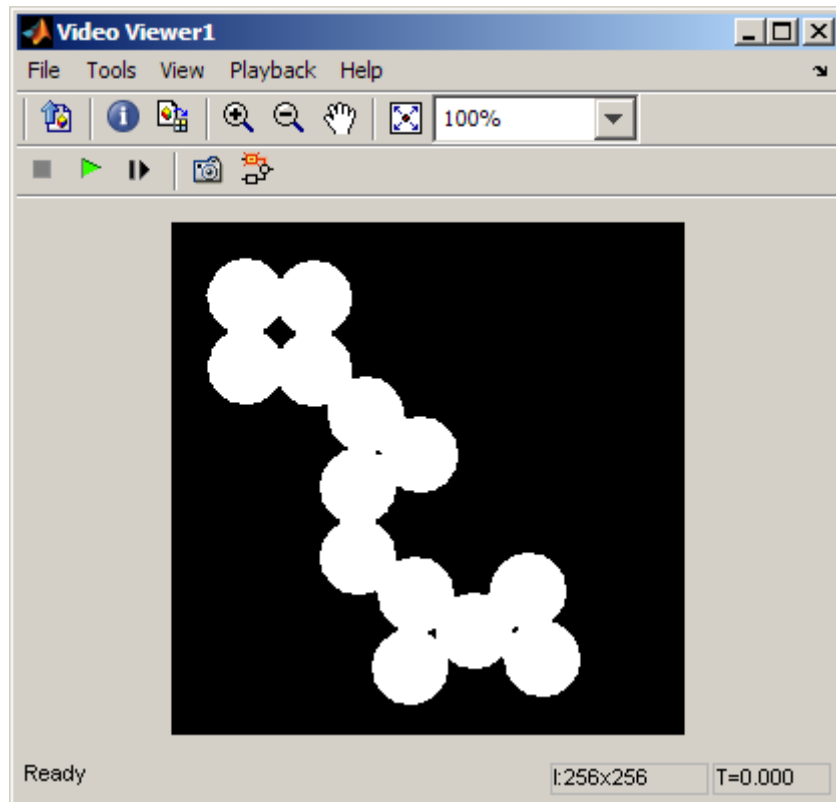
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

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 *Video and Image Processing Blockset Reference*.

Removing Periodic Noise from 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:

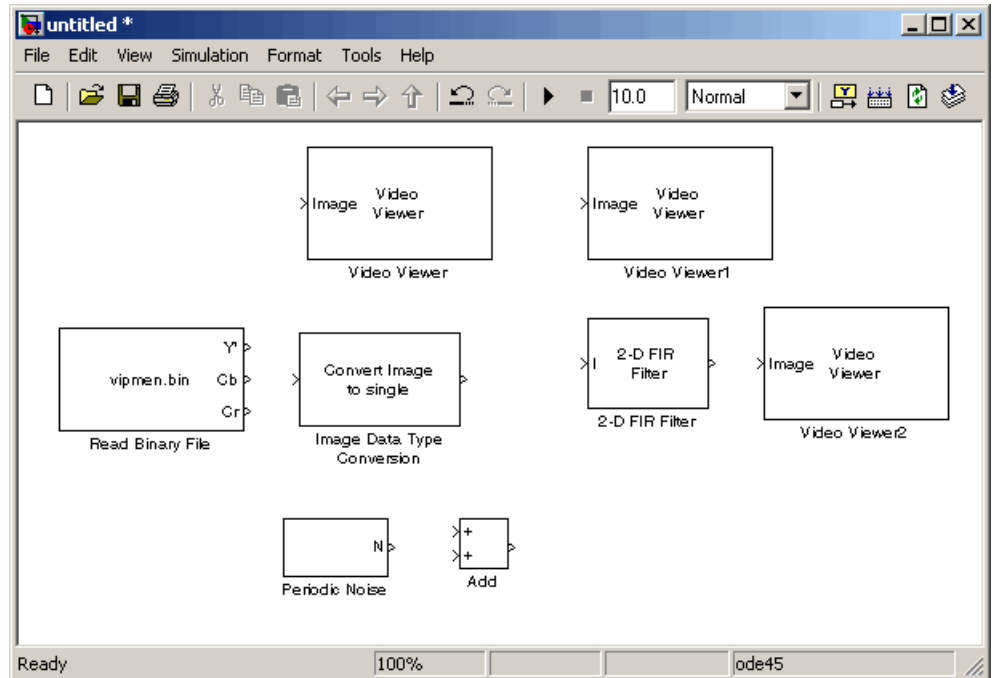
- 1 Create a Simulink model, and add the blocks shown in the following table.

Block	Library	Quantity
Read Binary File	Video and Image Processing Blockset > Sources	1
Image Data Type Conversion	Video and Image Processing Blockset > Conversions	1
2-D FIR Filter	Video and Image Processing Blockset > Filtering	1
Video Viewer	Video and Image Processing Blockset > 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.

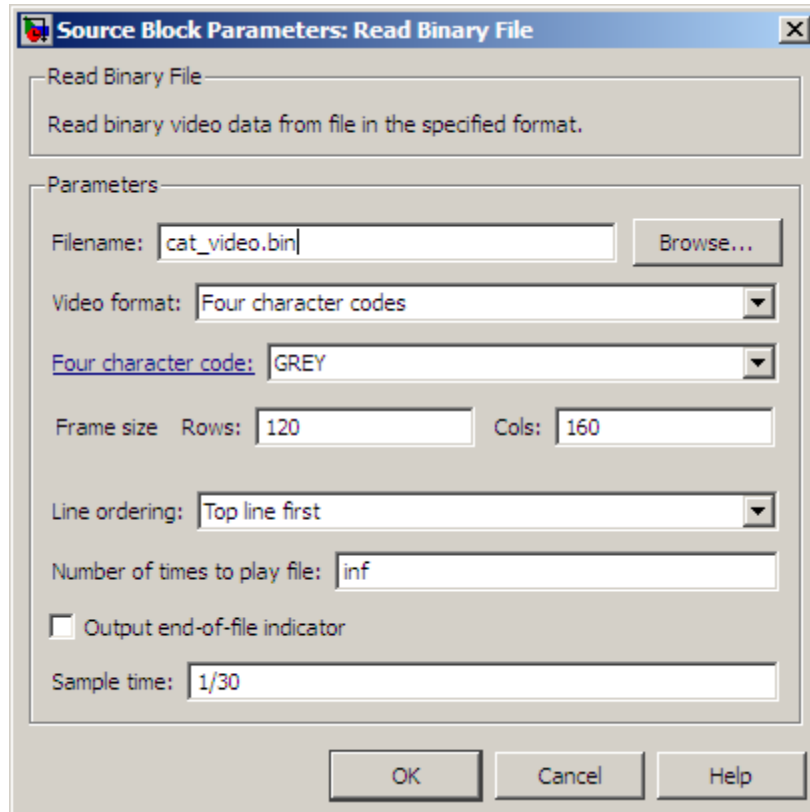
The block outputs a sinusoid with a normalized frequency that ranges between 0.61π and 0.69π radians per sample and a phase that varies between zero and three radians. You are using this sinusoid to represent periodic noise.

- 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



- 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:

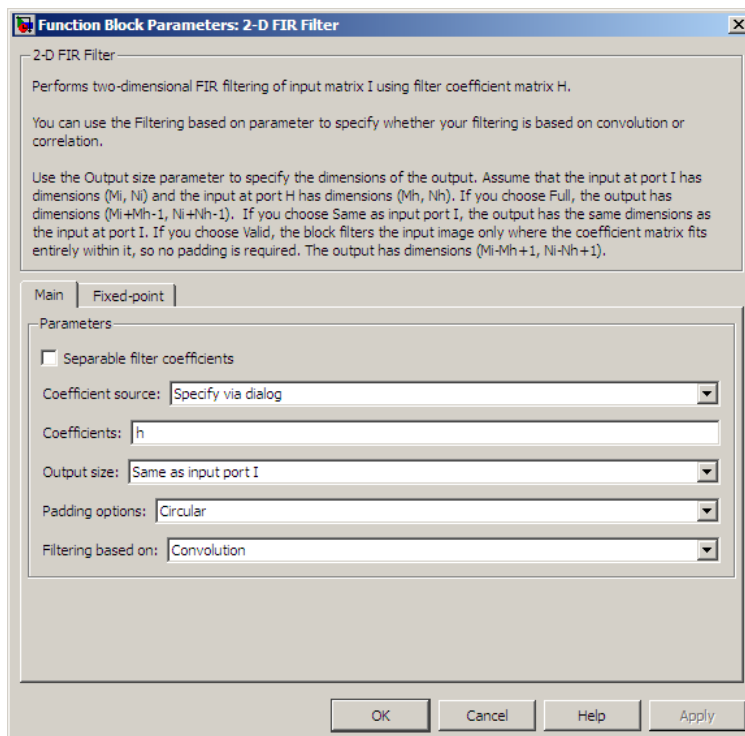
`vipdh_stripes`

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.

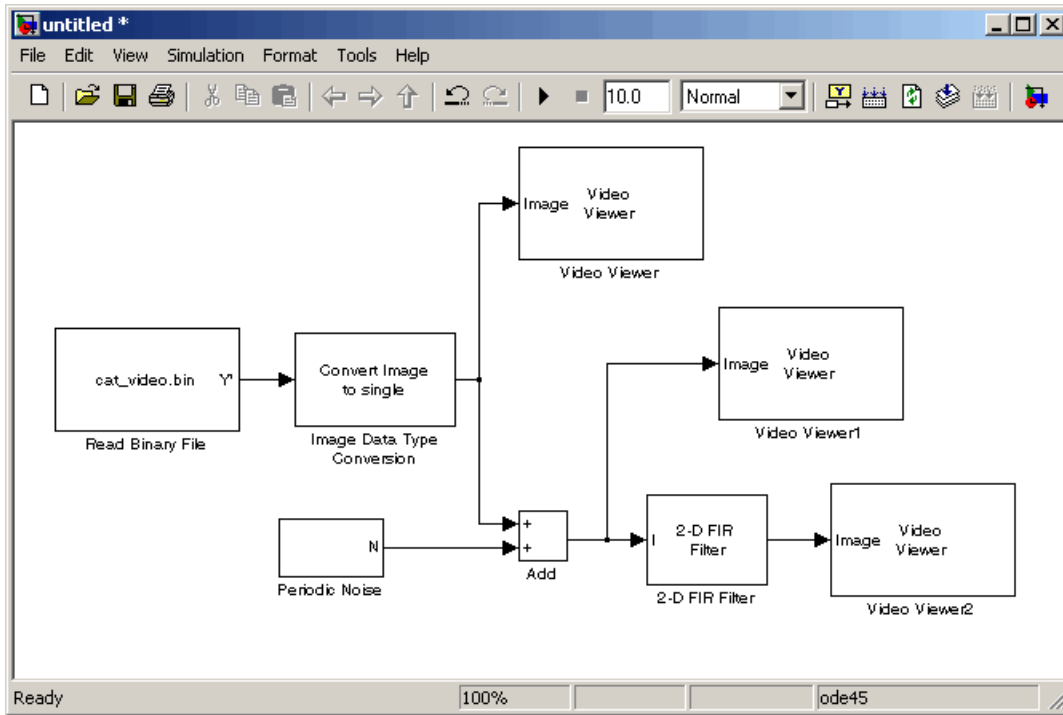
- 11** Use the 2-D FIR Filter block to model a band-reject filter capable of removing the periodic noise from the video. Set the block parameters as follows:

- **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.



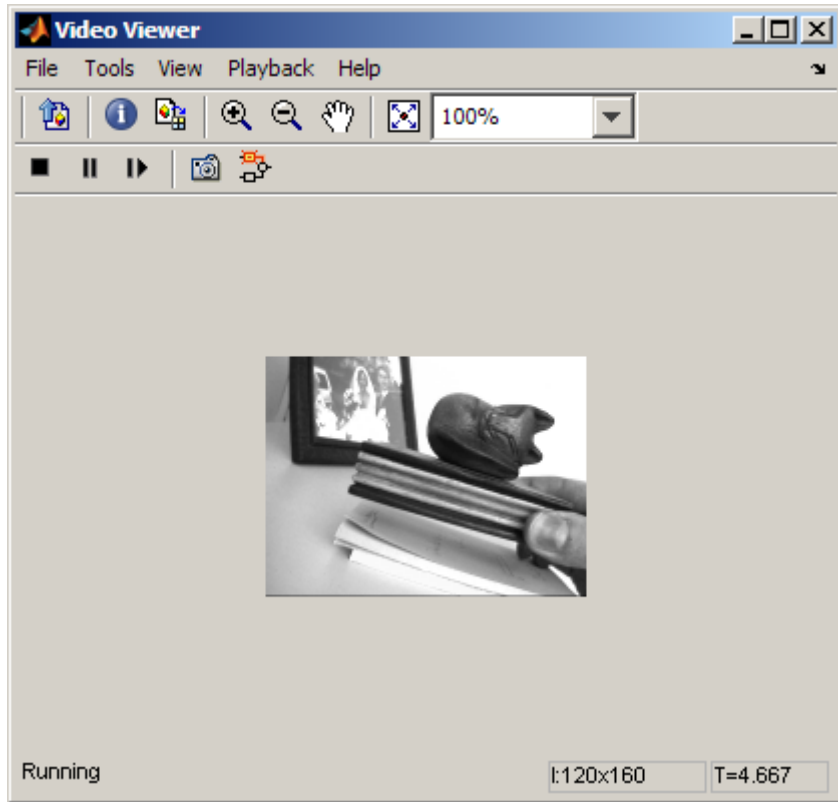
- 12** Use the Video Viewer2 block to view the approximation of the original video. Accept the default parameters.
- 13** Connect the block as shown in the following figure.



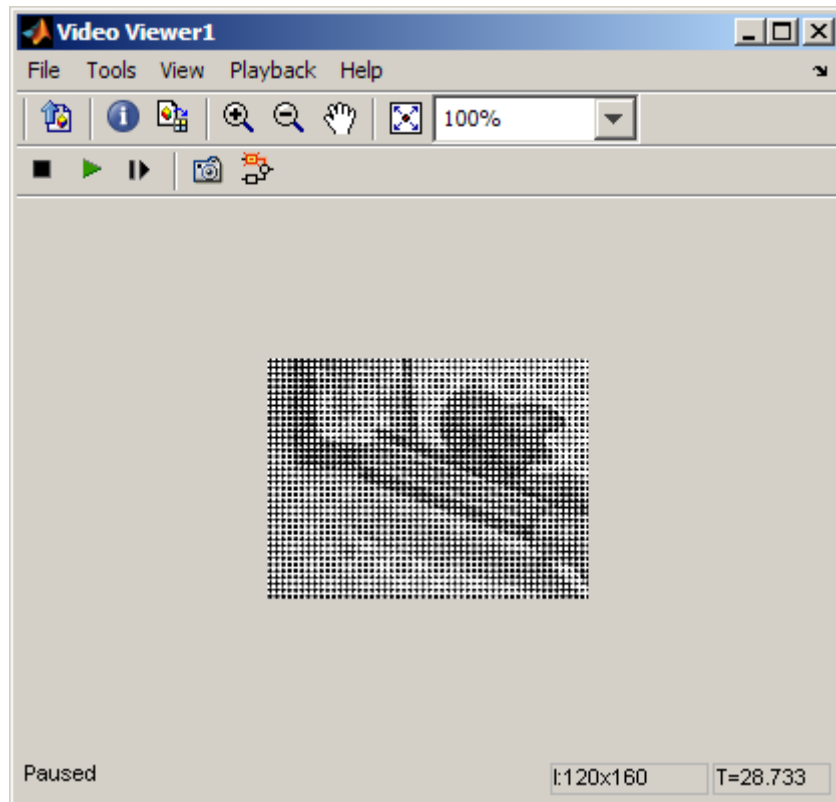
14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

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

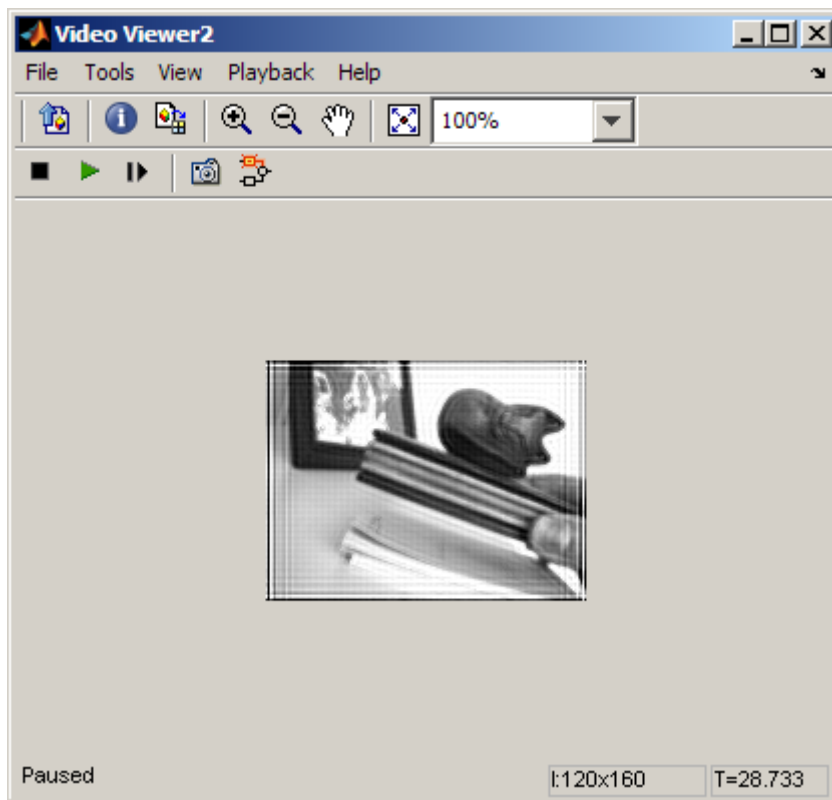
15 Run the model.



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 in the *Video and Image Processing Blockset Reference*. 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.

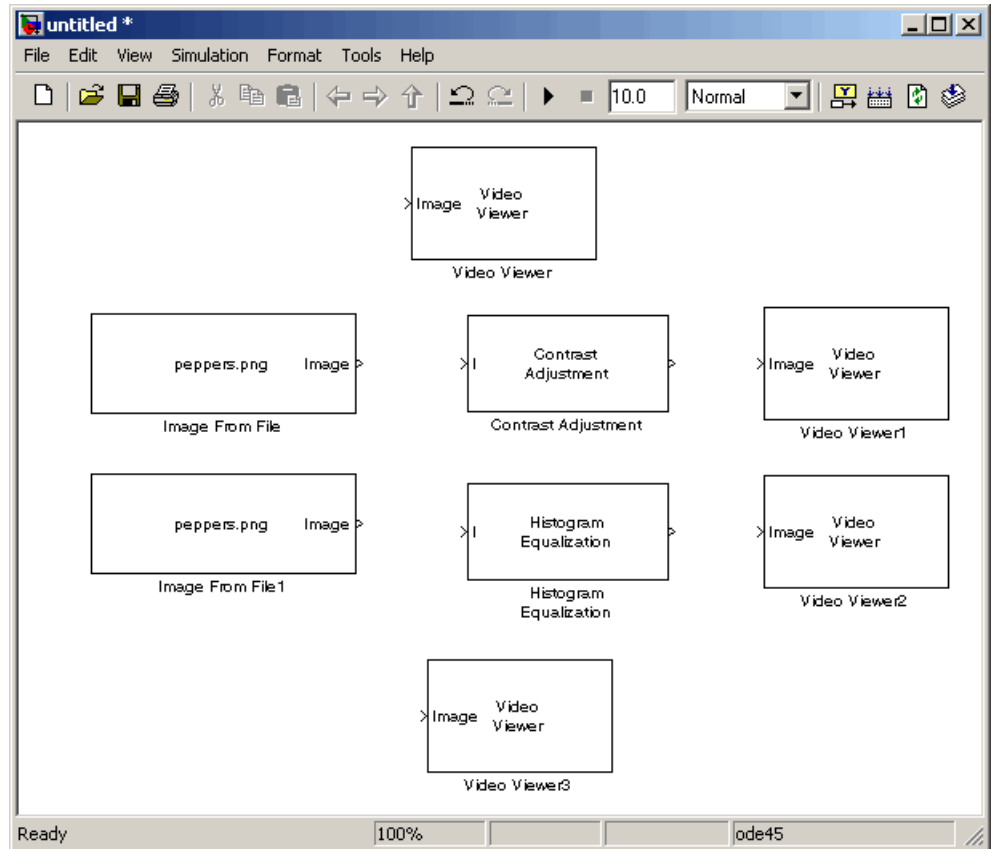
Adjusting the Contrast in Intensity Images

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

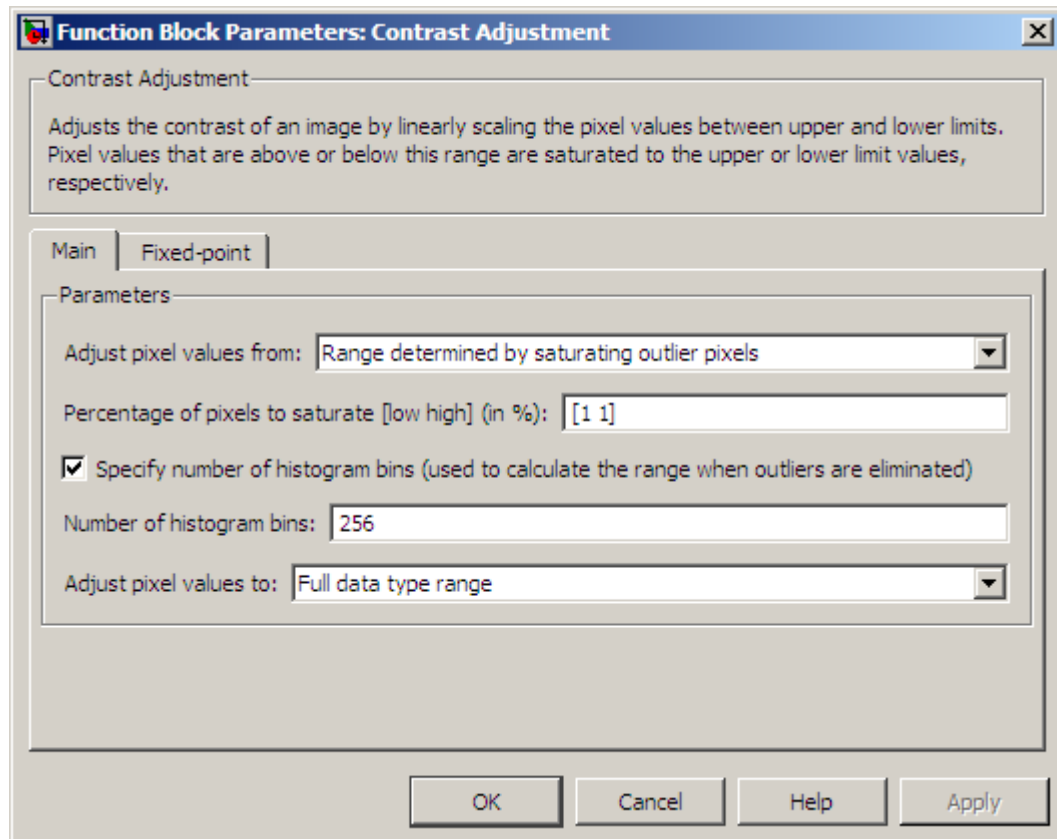
- 1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	2
Contrast Adjustment	Video and Image Processing Blockset > Analysis & Enhancement	1
Histogram Equalization	Video and Image Processing Blockset > Analysis & Enhancement	1
Video Viewer	Video and Image Processing Blockset > Sinks	4

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

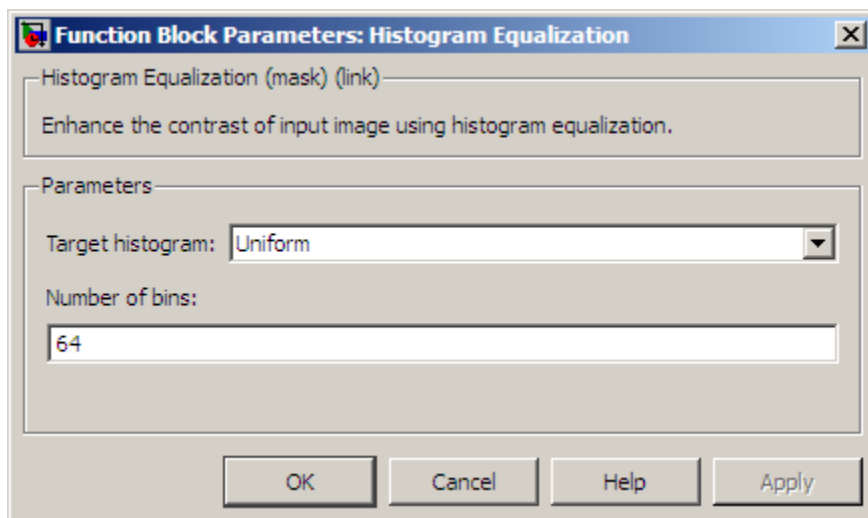


- 3 Use the Image From File block to import the first image into the Simulink model. Set the **File name** parameter to `pout.tif`.
- 4 Use the Image From File1 block to import the second image into the Simulink model. Set the **File name** parameter to `tire.tif`.
- 5 Use the Contrast Adjustment block 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.



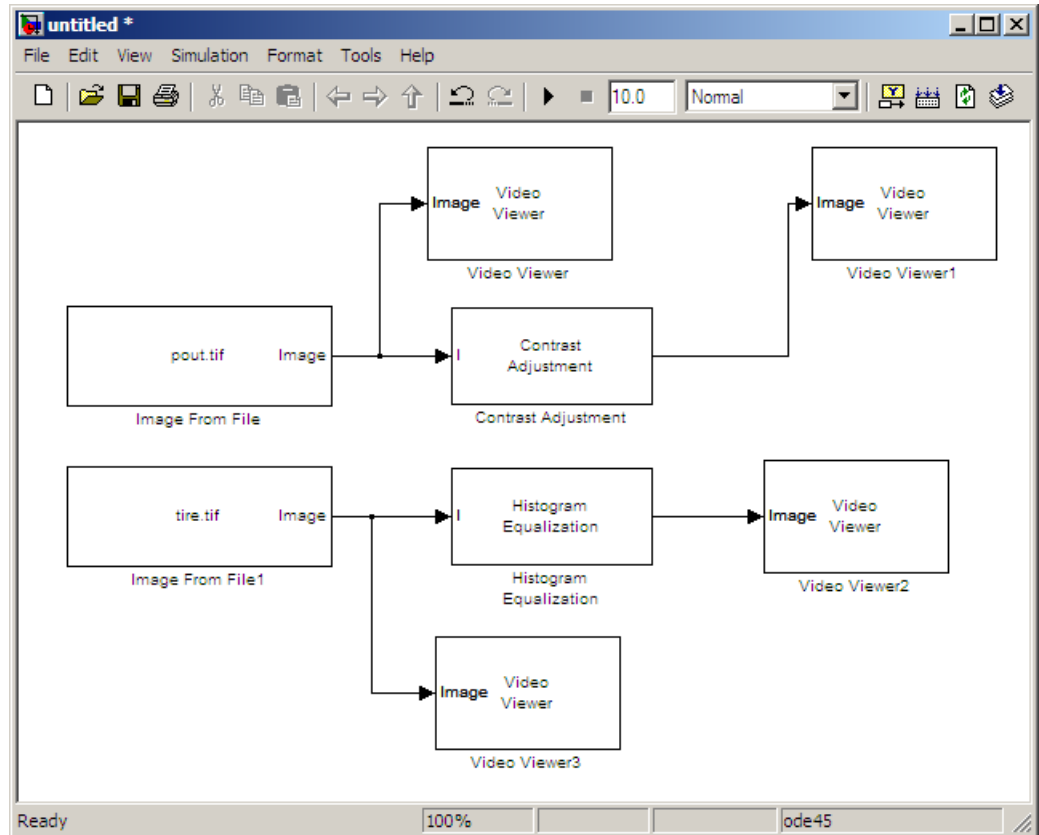
This block adjusts the contrast of the image by linearly scaling the pixel values between user-specified upper and lower limits.

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



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.

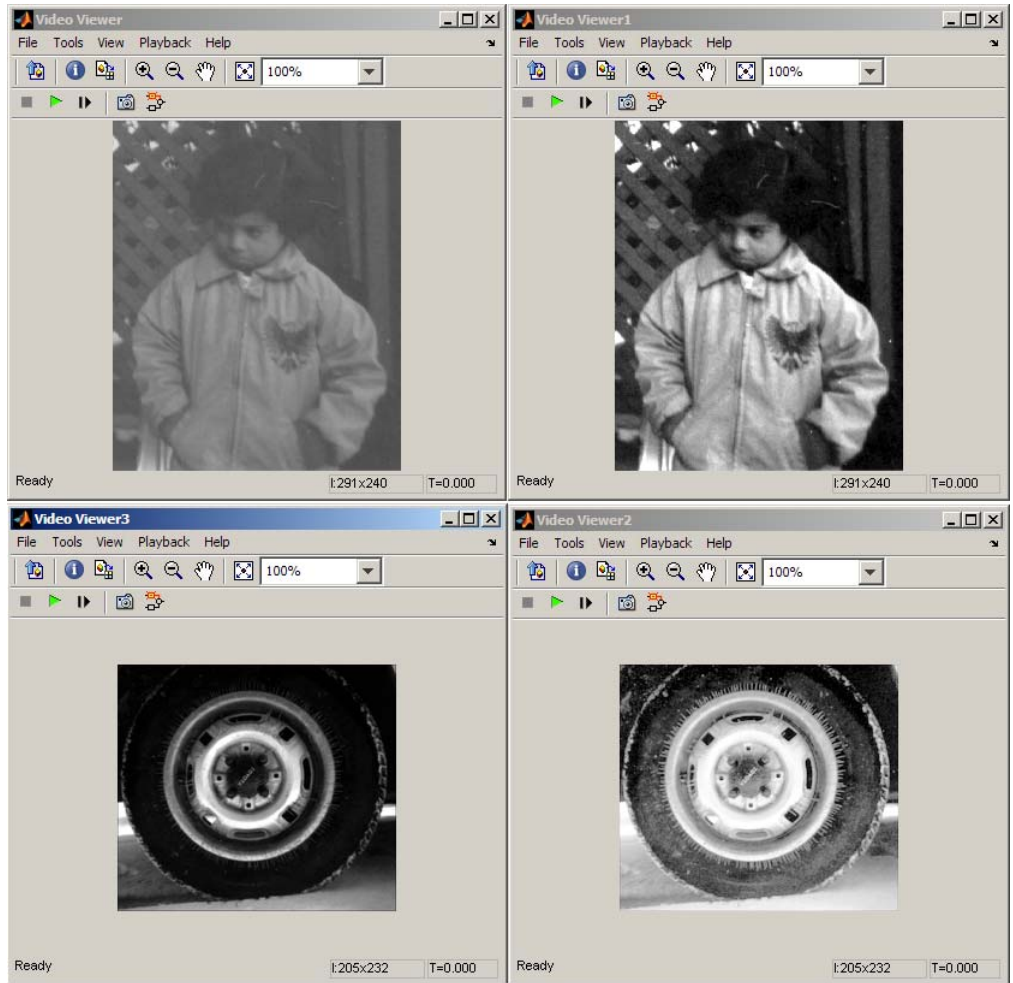


9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- **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 `put.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 block reference pages in the *Video and Image Processing Blockset Reference*.

Adjusting the Contrast in Color Images

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

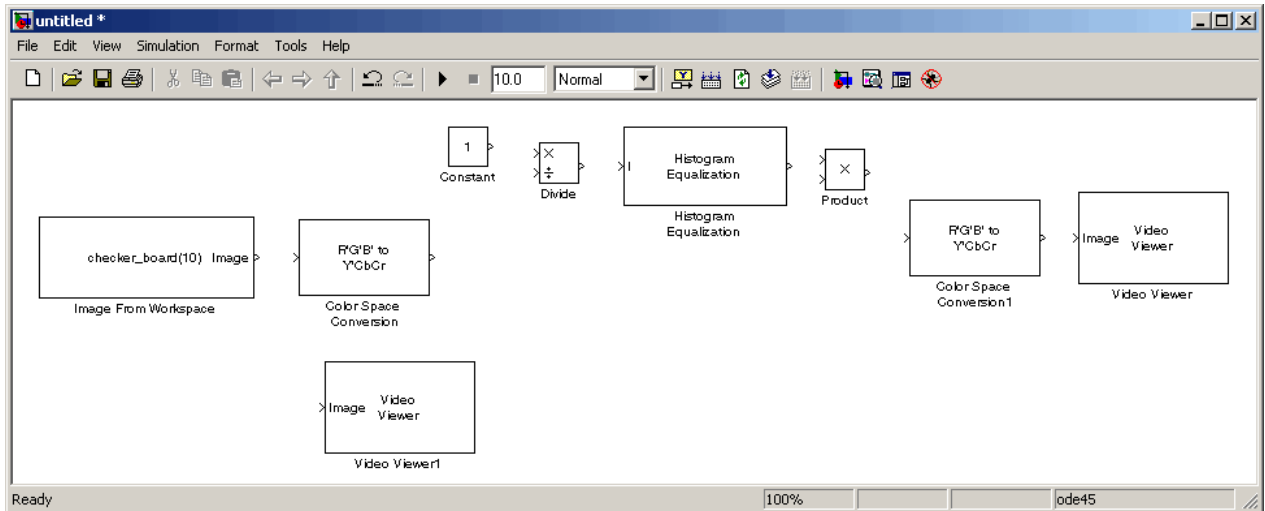
- 1 Use the following code to read in an indexed RGB image, `shadow.tif`, and convert it to an RGB image.

```
[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	Video and Image Processing Blockset > Sources	1
Color Space Conversion	Video and Image Processing Blockset > Conversions	2
Histogram Equalization	Video and Image Processing Blockset > Analysis & Enhancement	1
Video Viewer	Video and Image Processing Blockset > 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:

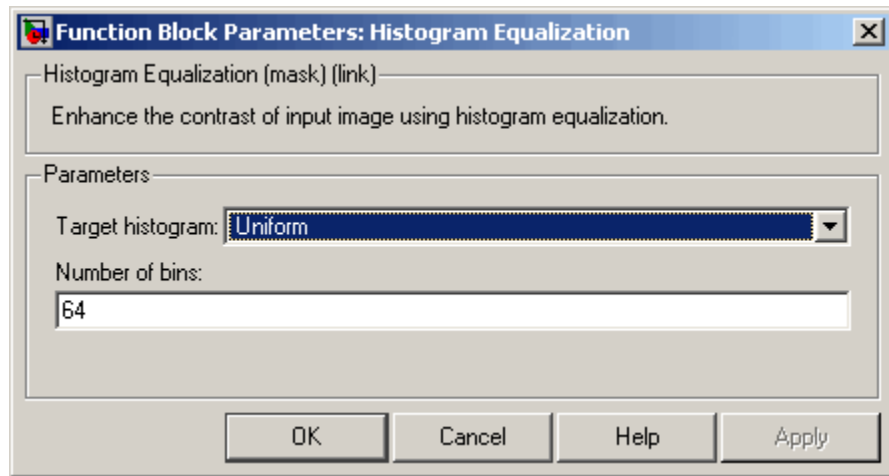
- **Conversion** = sR'G'B' to L*a*b*
- **Image signal** = Separate color signals

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.

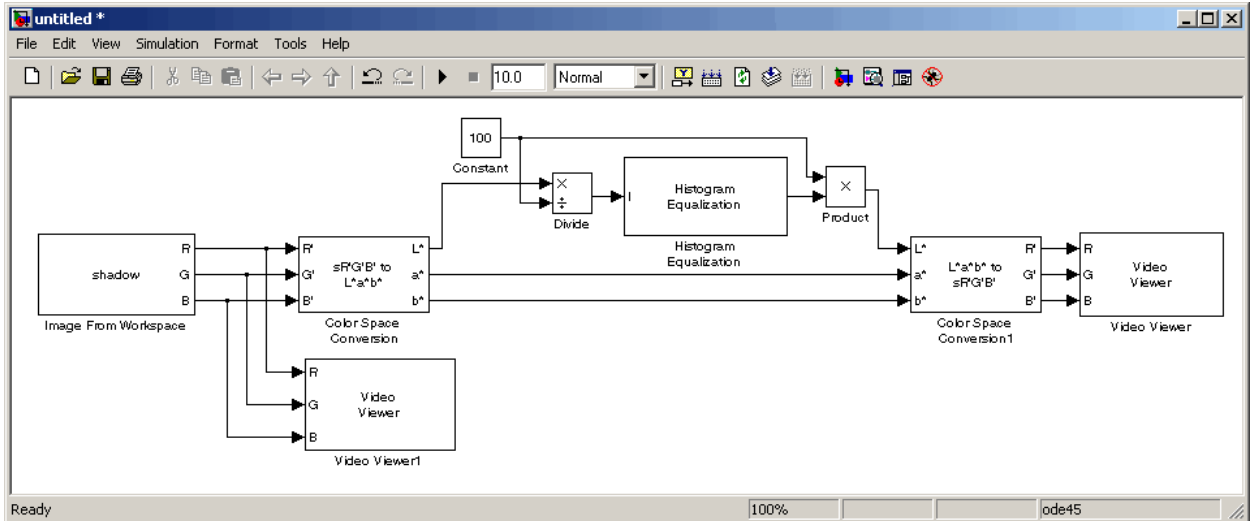
7 Use the Divide block to normalize the L^* values to be between 0 and 1. Accept the default parameters.

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



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.
- 12 Connect the blocks as shown in the following figure.

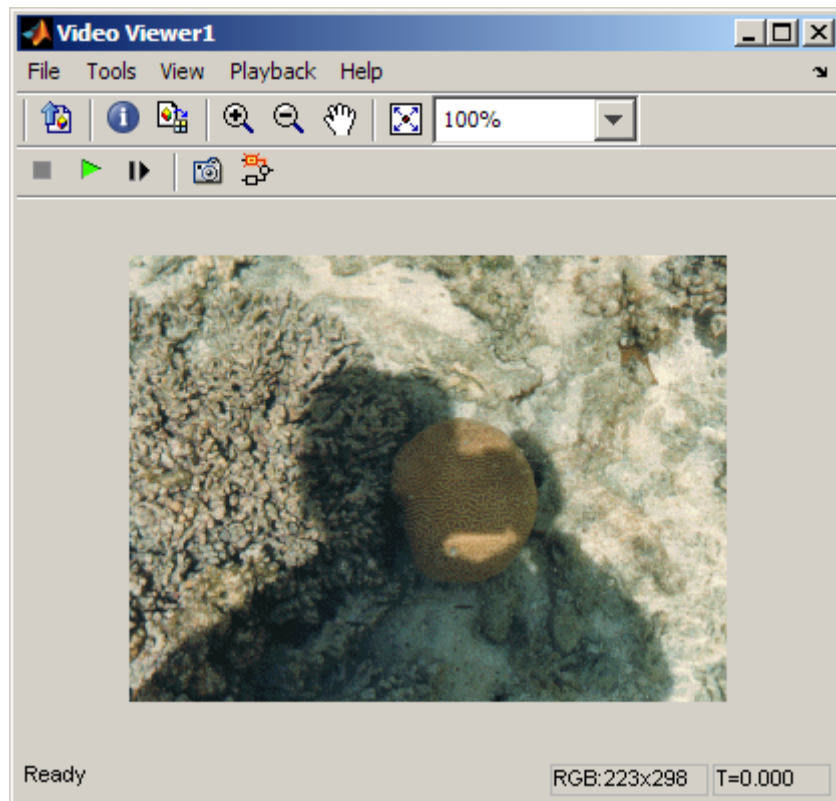


13 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

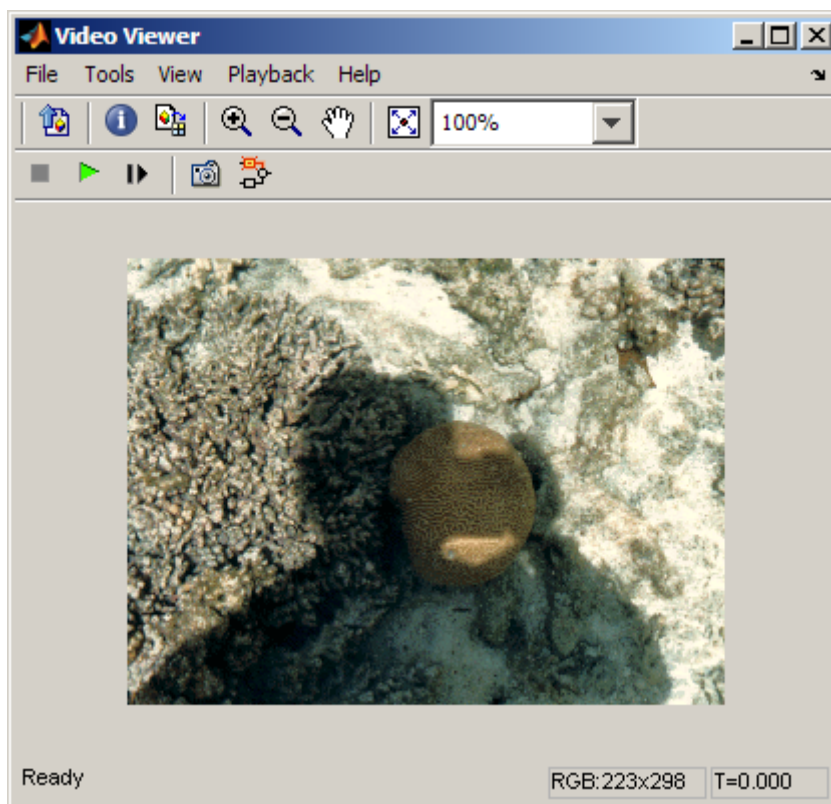
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

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 block reference page in the *Video and Image Processing Blockset Reference*.

Template Matching

In this section...

“Using the Template Matching Block” on page 4-67

“Video Stabilization” on page 4-71

“Panorama Creation” on page 4-72

Using the Template Matching Block

Template matching is a technique in image processing for finding subregions of an image which match a template image. Template matching applications include manufacturing, robotics, edge, or shape detection and facial recognition. The matching process moves the template image to all possible positions in a larger source image. It computes a numerical metric that indicates how well the template matches the image in that position.

You can use any of the follow metrics for template matching with the Video and Image Processing Blockset:

- Sum of Absolute Differences (SAD)
- Sum of Squared Differences (SSD)
- Maximum Absolute Difference (MaxAD)

Choosing an Output Option

The block outputs either a matrix of match metric values or the zero-based location of the best template match. The block outputs the matrix to the **Metric** port or the location to the **Loc** port. Optionally, the block can output an $N \times N$ matrix of neighborhood match metric values to the **NMetric** port.

Input and Output Signal Sizes

Since the Template Matching block does not pad the input data, it can only compute values for the match metrics between the input image and the template for where the template is positioned such that it entirely falls on the input image. A set of all such positions of the template is termed as the

valid region of the input image. The size of the *valid* region is the difference between the sizes of the input and template images plus one.

$$size_{valid} = size_{input} - size_{template} + 1$$

The output at the **Metric** port for the Match metric mode is of the *valid* image size. The output at the **Loc** port for the Best match index mode is a two-element vector of indices relative to the top-left corner of the input image.

The neighborhood metric output at the **NMetric** port is of the size $N \times N$ where N must be an odd number defined in the block mask.

Defining the Region of Interest (ROI)

To perform template matching on a subregion of the input image, select the **Enable ROI processing** check box. This check box adds an input port **ROI** to the Template Matching block. The ROI processing option is only available for the Best match index mode.

The ROI port requires a four-element vector that defines a rectangular area. The first two elements represent the zero-based row and column coordinates for the upper-left corner. The second two elements represent the height and width of the ROI. The block outputs best match location index relative to the top left corner of the input image.

Choosing a Match Metric

The block computes the match metric at each step of the iteration. Choose the match metric which best suits your application. The block evaluates the best metric value based on the metric choice. The best metric is the global optimum over the support of the “valid” subregion of the input image intersected by the ROI, if provided.

Returning the Matrix of Match Metric Values

The matrix of the match metric values always implements single-step exhaustive window iteration. Therefore, the block computes the metric values at every pixel.

Returning the Best Match Location

When in the ROI processing mode, the block treats the image around the ROI as an extension of the ROI subregion. Therefore, it computes the best match locations true to the actual boundaries of the ROI. The best match location indices returned are relative to the top-left corner of the input image.

Returning the Neighborhood Match Metric around the Best Match

If you choose to return the matrix of metric values in a neighborhood around the best match, an exhaustive loop computes all the metric values for the $N \times N$ neighborhood. This output is particularly useful for performing template matching with subpixel accuracy.

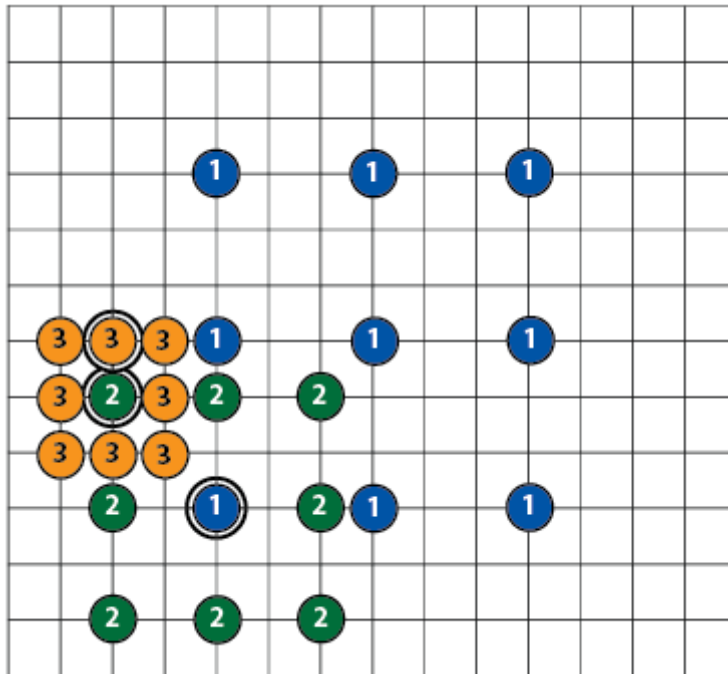
Choosing a Search Method

When you select `Best match location` as the output option, you can choose to use either `Exhaustive` or `Three-step` search methods.

The `Exhaustive` search method is computation intensive because it searches at every pixel location of the image.

The `Three-step` search method is a fast search method employing the following steps:

- 1** The search starts with a step size equal to or slightly greater than half of the maximum search range.
- 2** The block compares nine search points in each step. They comprise the central point of the search square, and eight search points located on the search area boundaries.
- 3** The block decrements the step size by one, after each step, ending the search with a step size of 1 pixel.
- 4** At each new step, the block moves the search center to the best matching point resulting from the previous step.



Three-Step Search

Using the ROIValid and NValid flags for Diagnostics

The **ROIValid** and **NValid** ports represent boolean flags, which track the valid Region of Interest (ROI) and neighborhood. You can use these flags to communicate with the downstream blocks and operations.

Valid Region of Interest

If you select the **Output flag indicating if ROI is valid** check box, the block adds the **ROIValid** port. If the ROI lies partially outside the valid image, the block only processes the intersection of the ROI and the valid image. The block sets the ROI flag output to this port as follows:

- True, set to 1 indicating the ROI lies completely inside the valid part of the input image.

- False, set to 0 indicating the ROI lies completely or partially outside of the valid part of the input image.

Valid Neighborhood

The neighborhood matrix of metric values is valid inside of the Region of Interest (ROI). You can use the Boolean flag at the **NValid** port to track the valid neighborhood region. The block sets the neighborhood **NValid** boolean flag output as follows:

- True, set to 1 indicating that the neighborhood containing the best match is completely inside the region of interest.
- False, set to 0 indicating that the neighborhood containing the best match is completely or partially outside of the region of interest.

Video Stabilization

The *Estimate Motion Subsystem for Video Stabilization* demo implements the Template Matching block. The purpose of the model is to track 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 Video and Image Processing Blockset by typing `vipdemos` on the MATLAB command line. You can launch the Video Stabilization model directly by typing `vipstabilize` on the MATLAB command line.

Panorama Creation

The *Motion Estimation Subsystem of the Panorama Creation* demo implements the Template Matching block. This model uses the block to estimate the motion between consecutive video frames. Then it 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 Video and Image Processing Blockset by typing `vipdemos` on the MATLAB command line. You can launch the Panorama model directly by typing `vippanorama` on the MATLAB command line.

Pixel Statistics

In this section...

“Blocks That Compute Pixel Statistics” on page 4-73

“Finding the Histogram of an Image” on page 4-73

Blocks That Compute Pixel Statistics

The Video and Image Processing Blockset software together with the Signal Processing Blockset software contains blocks that can provide information about the data values that make up an image. Blocks from the Statistics library, such as the Maximum and 2-D Autocorrelation blocks, can help you determine this information.

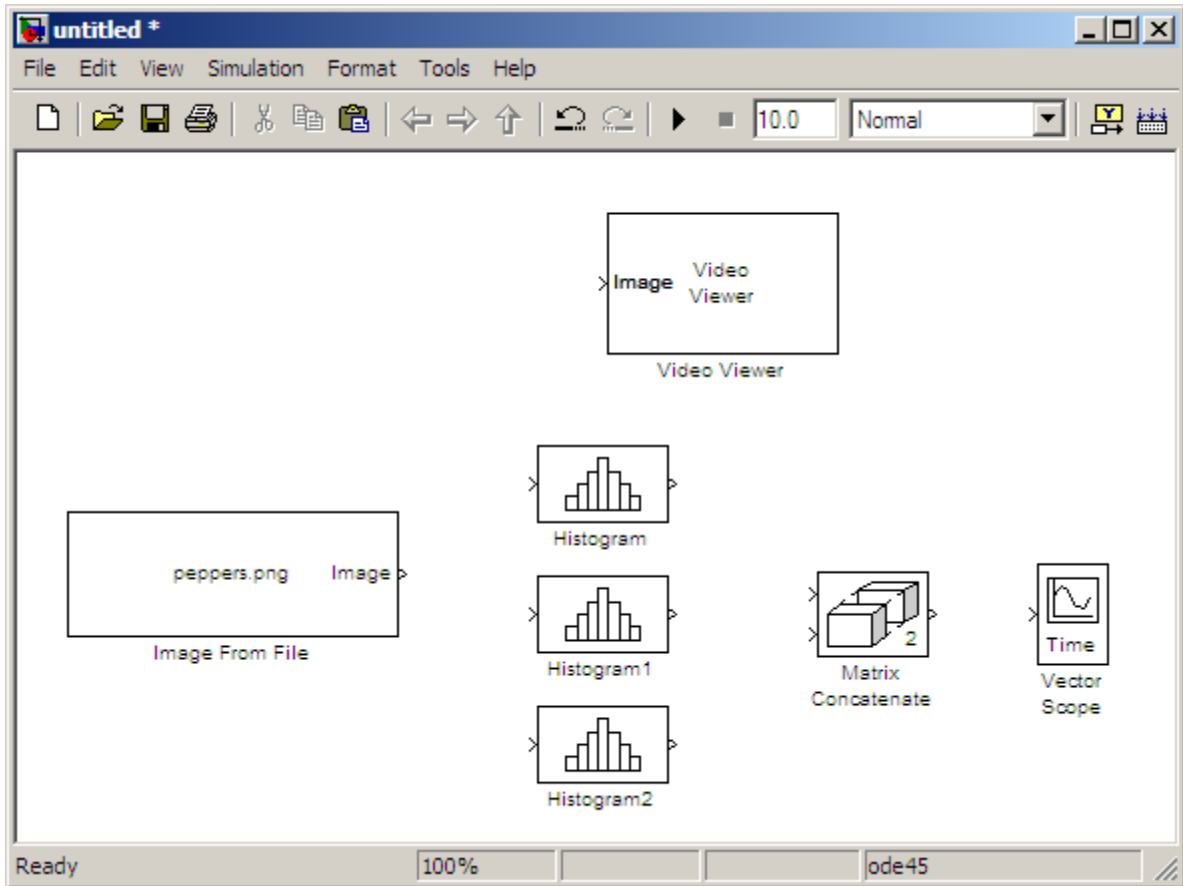
Finding the Histogram of an Image

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	Video and Image Processing Blockset > Sources	1
Video Viewer	Video and Image Processing Blockset > Sinks	1
Matrix Concatenate	Simulink > Math Operations	1
Vector Scope	Signal Processing Blockset > Signal Processing Sinks	1
Histogram	Signal Processing Blockset > Statistics	3

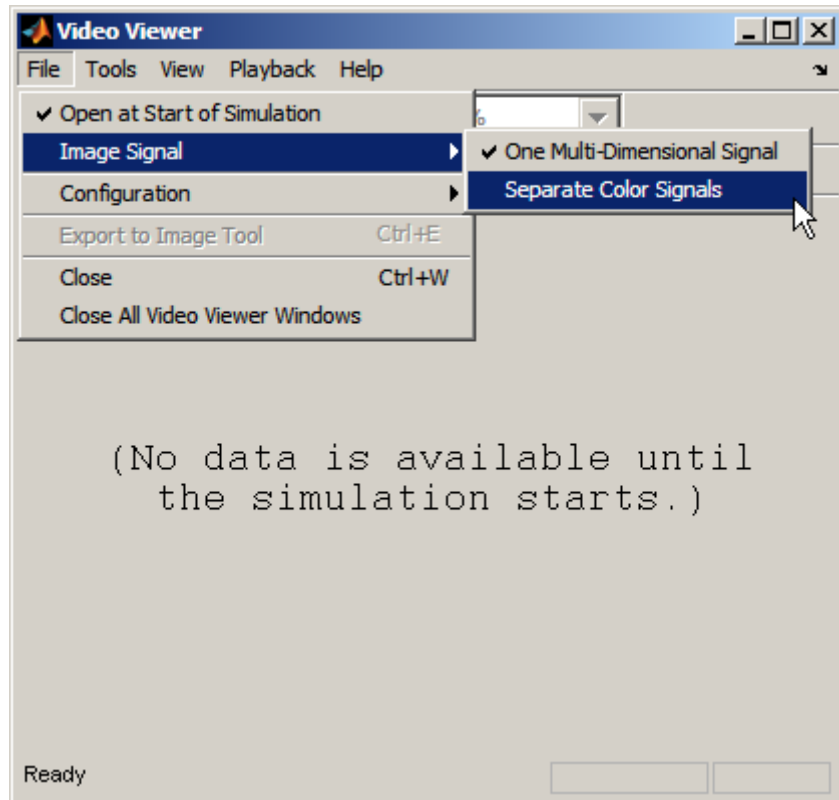
2 Place the blocks so that your model resembles the following figure.



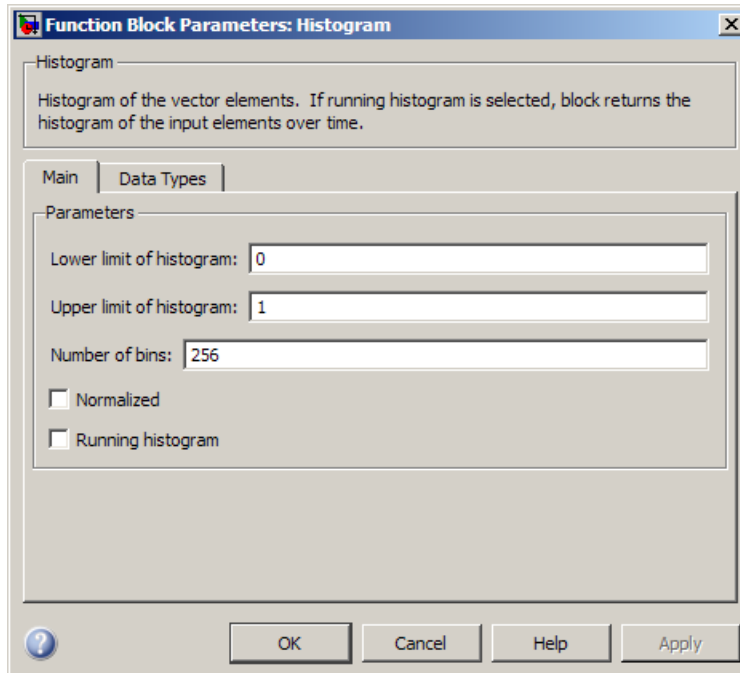
3 Use the Image From File block to import an RGB image. Set the block parameters as follows:

- **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**.



- 5 Use the Histogram block to calculate the histogram of the R, G, and B values in the image. Set the Main tab block parameters as follows:
 - **Lower limit of histogram:** 0
 - **Upper limit of histogram:** 1
 - **Number of bins:** = 256

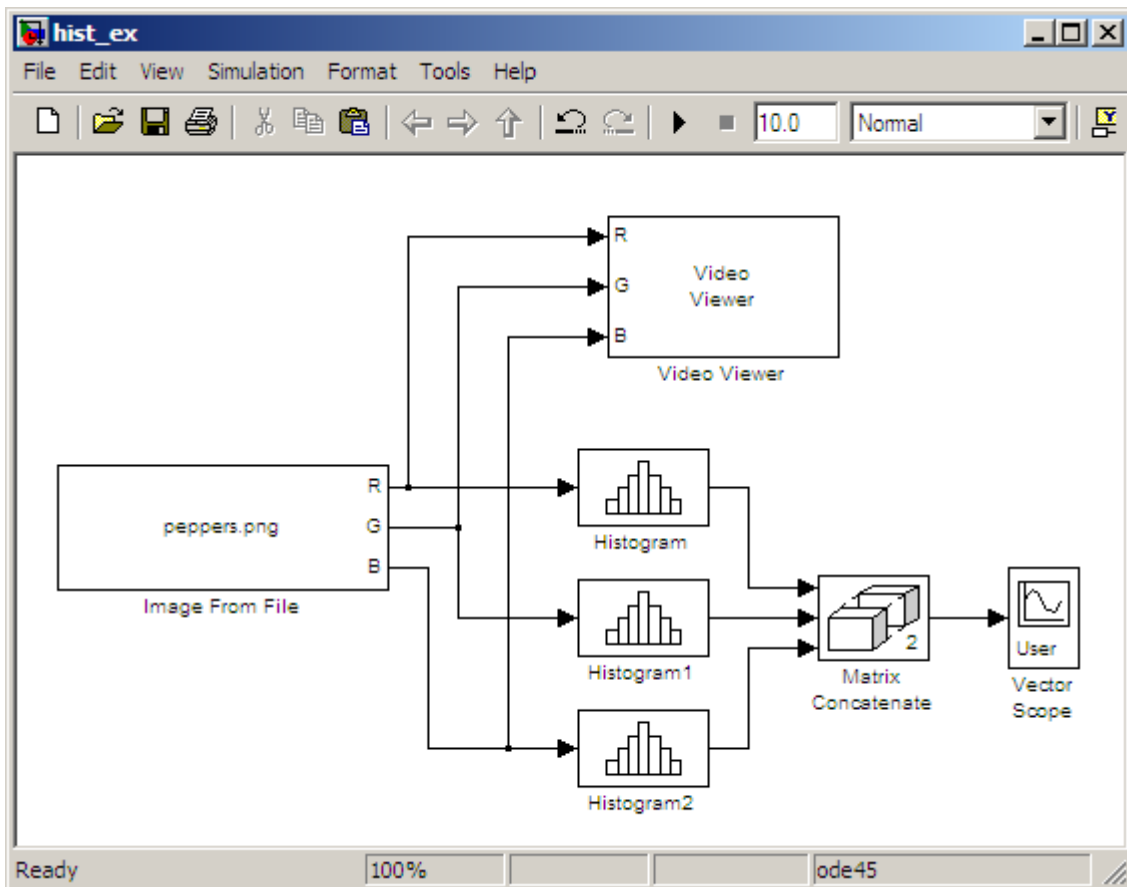


The **R**, **G**, and **B** input values to the Histogram block are double-precision floating point and range between 0 and 1. The block creates 256 bins between the maximum and minimum input values and counts the number of **R**, **G**, and **B** values in each bin.

- 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

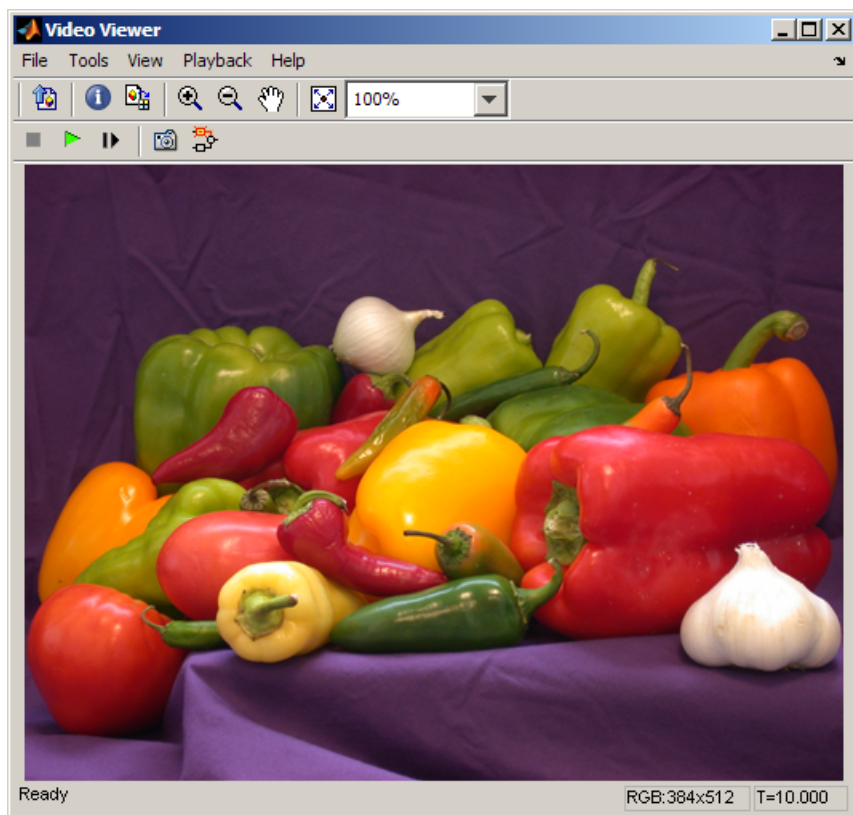
- **Axis Properties** pane, clear the **Inherit sample increment from input** check box.
- **Axis Properties** pane, **Minimum Y-limit** = 0
- **Axis Properties** pane, **Maximum Y-limit** = 1
- **Axis Properties** pane, **Y-axis label** = Count
- **Line Properties** pane, **Line markers** = . |s|d
-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

8 Connect the blocks as shown in the following figure.



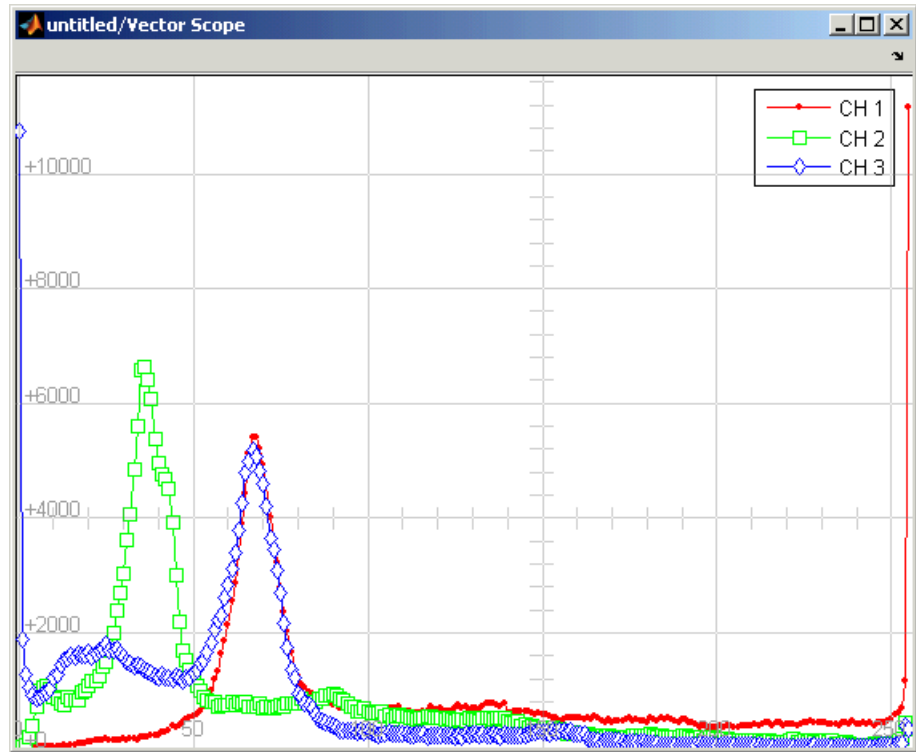
- 9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
 - **Solver** pane, **Stop time** = 0
 - **Solver** pane, **Type** = Fixed-step
 - **Solver** pane, **Solver** = Discrete (no continuous states)
- 10 Run the model.

The original 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 block reference page in the *Video and Image Processing Blockset Reference*. 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.

Conversions

- “Intensity to Binary Conversion” on page 5-2
- “Color Space Conversion” on page 5-14
- “Chroma Resampling” on page 5-19

Intensity to Binary Conversion

In this section...

“Overview of Intensity and Binary Images” on page 5-2

“Thresholding Intensity Images Using Relational Operators” on page 5-2

“Thresholding Intensity Images Using the Autothreshold Block” on page 5-7

Overview of Intensity and Binary Images

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. Intensity images can contain only 0s and 1s, but they are not binary images unless their data type is Boolean.

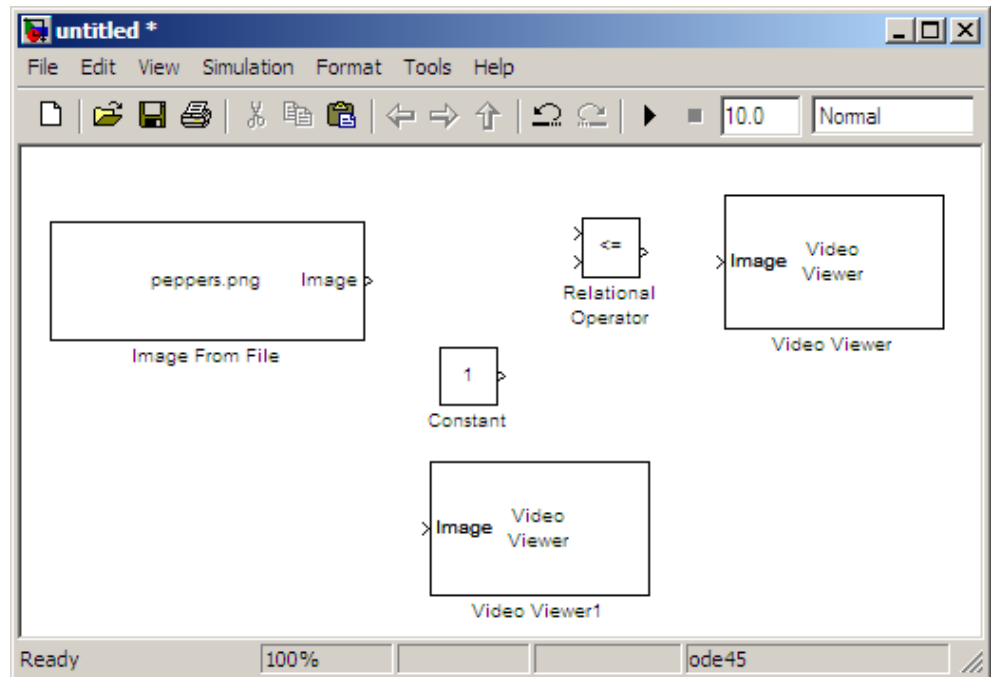
Thresholding Intensity Images Using Relational Operators

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:

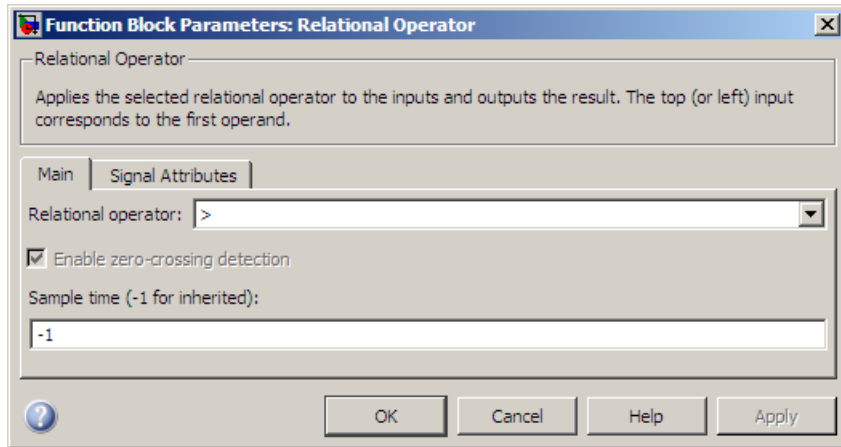
- 1 Create a new Simulink model, and add to it the blocks shown in the following table.

Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Video Viewer	Video and Image Processing Blockset > Sinks	2
Relational Operator	Simulink > Logic and Bit Operations	1
Constant	Simulink > Sources	1

2 Position the blocks as shown in the following figure.

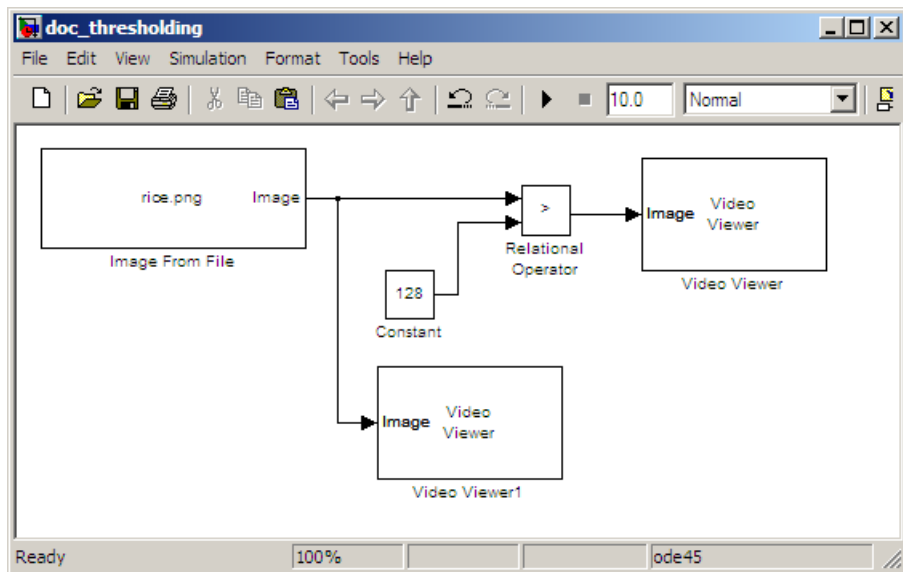


- 3 Use the Image from File block to import your image. In this example the image file is a 256-by-256 matrix of 8-bit unsigned integer values that range from 0 to 255. Set the **File name** parameter to `rice.png`
- 4 Use the Video Viewer1 block to view the original intensity image. Accept the default 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.



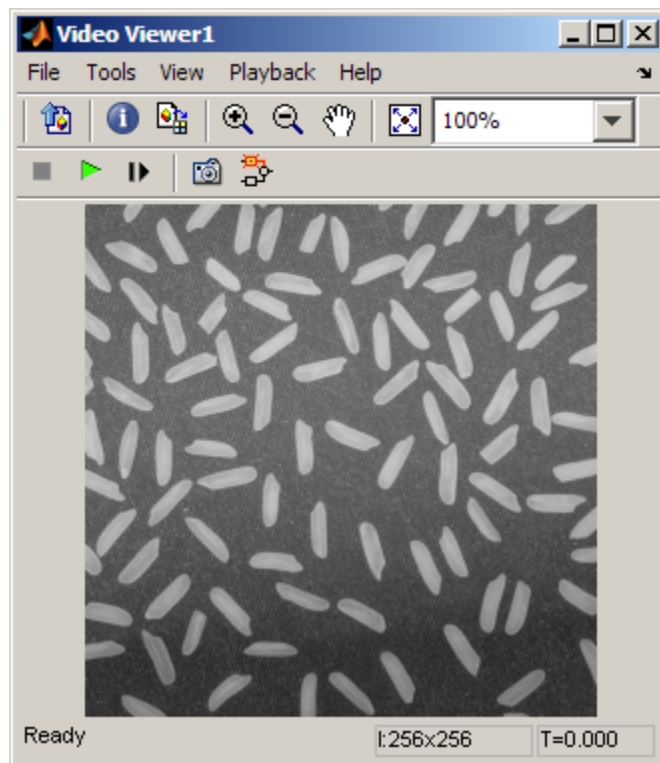
7 Use the Video Viewer block to view the binary image. Accept the default parameters.

8 Connect the blocks as shown in the following figure.

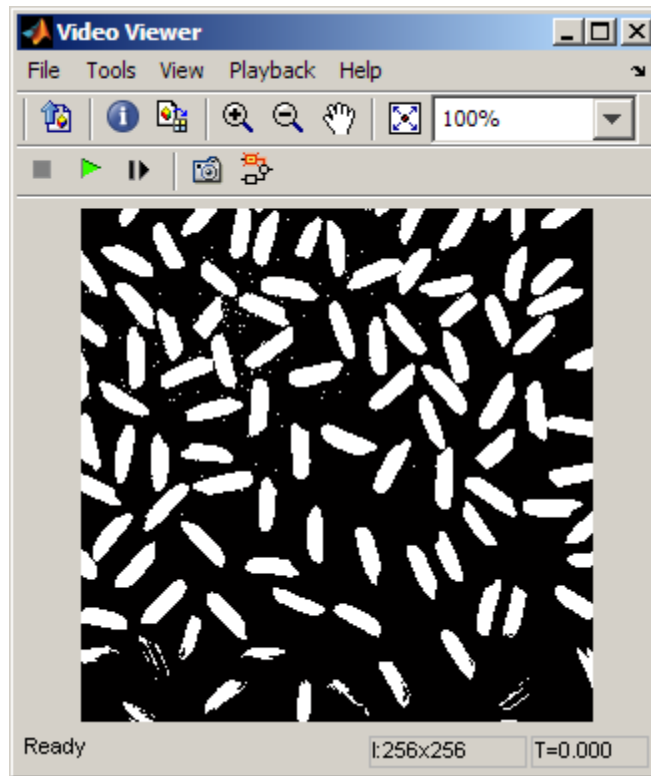


- 9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
 - **Solver** pane, **Stop time** = 0
 - **Solver** pane, **Type** = Fixed-step
 - **Solver** pane, **Solver** = Discrete (no 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 “Correcting for Nonuniform Illumination” on page 7-10.

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 another example that uses this technique, see “Counting Objects in an Image” on page 7-3. For additional information, see “Converting Between Image Types” in the Image Processing Toolbox documentation.

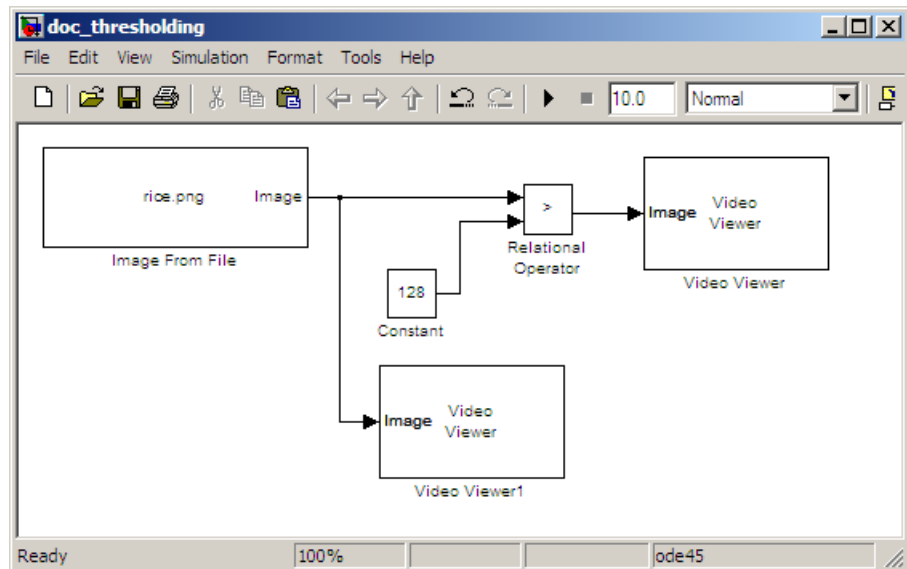
Thresholding Intensity Images Using the Autothreshold Block

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 5-2 is not open on your desktop, you can open an equivalent model by typing

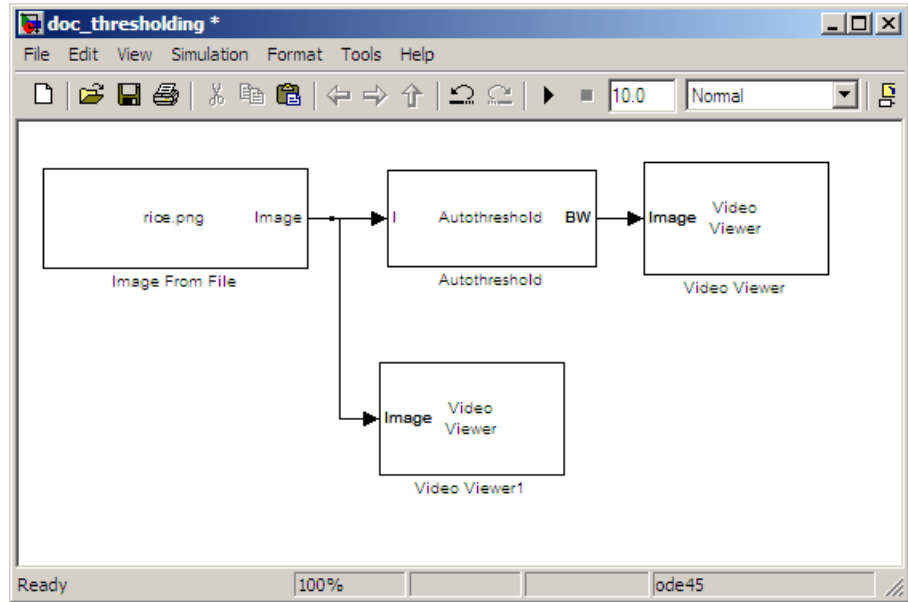
```
doc_thresholding
```

at the MATLAB command prompt.

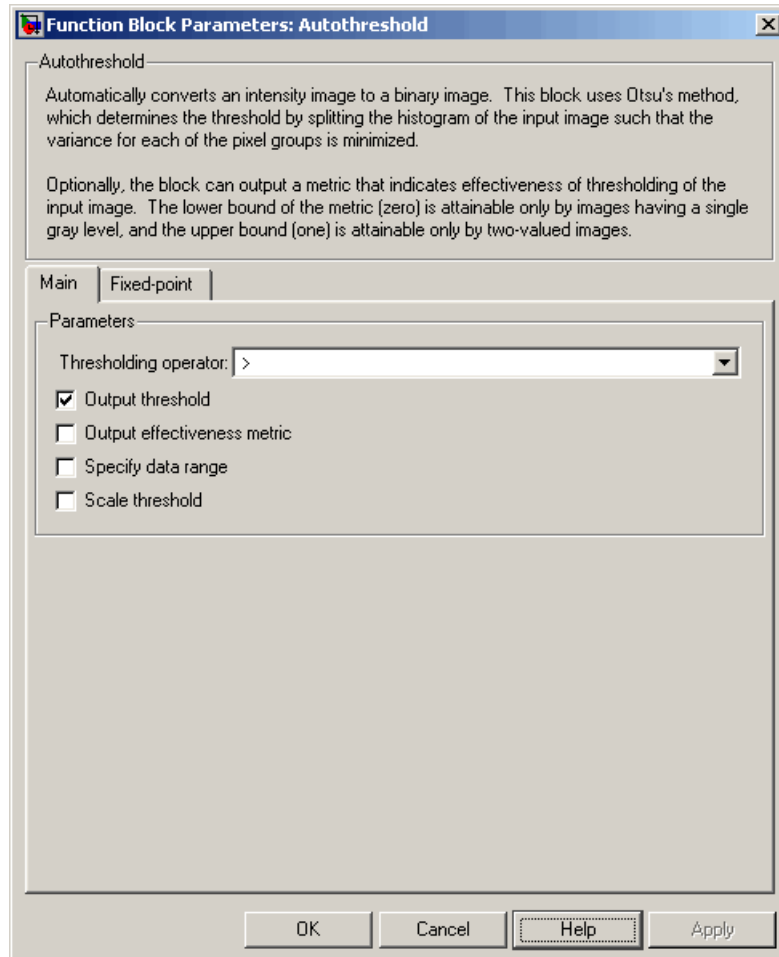


- 2 Use the Image from File block to import your image. In this example the image file is a 256-by-256 matrix of 8-bit unsigned integer values that range from 0 to 255. Set the **File name** parameter to `rice.png`
- 3 Delete the Constant and the Relational Operator blocks in this model.

- 4 Add an Autothreshold block from the Conversions library of the Video and Image Processing Blockset into your model.
- 5 Connect the blocks as shown in the following figure.



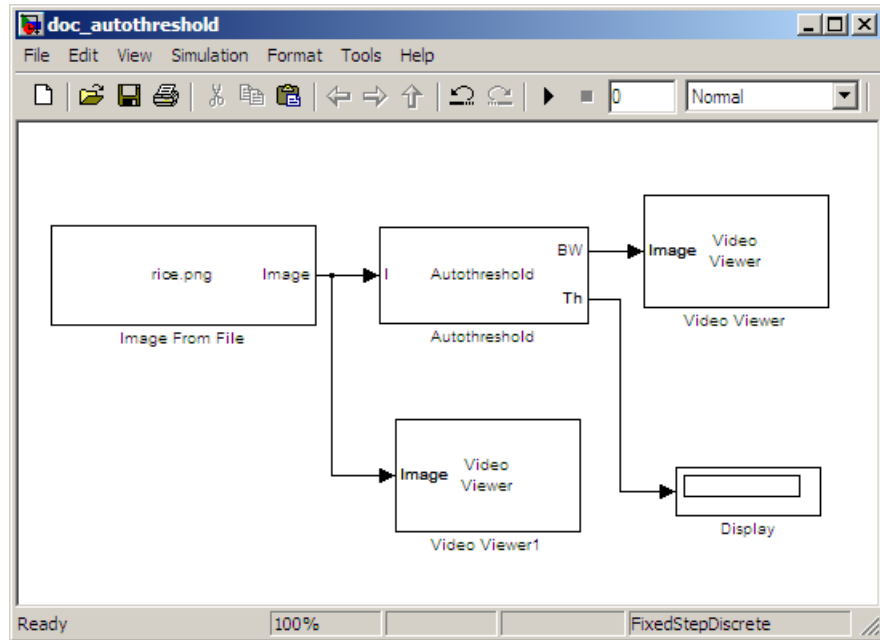
- 6 Use the Autothreshold block to perform a thresholding operation that converts your intensity image to a binary image. Select the **Output threshold** check box.



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

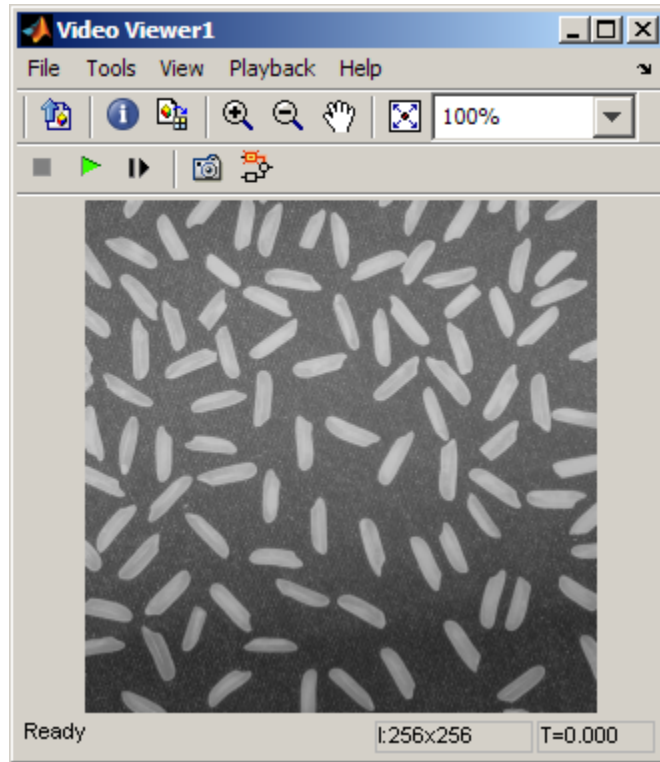
- 7 Add a Display block from the Sinks library of the Signal Processing Blockset library. Connect the Display block to the **Th** output port of the Autothreshold block.

Your model should look similar to the following figure:

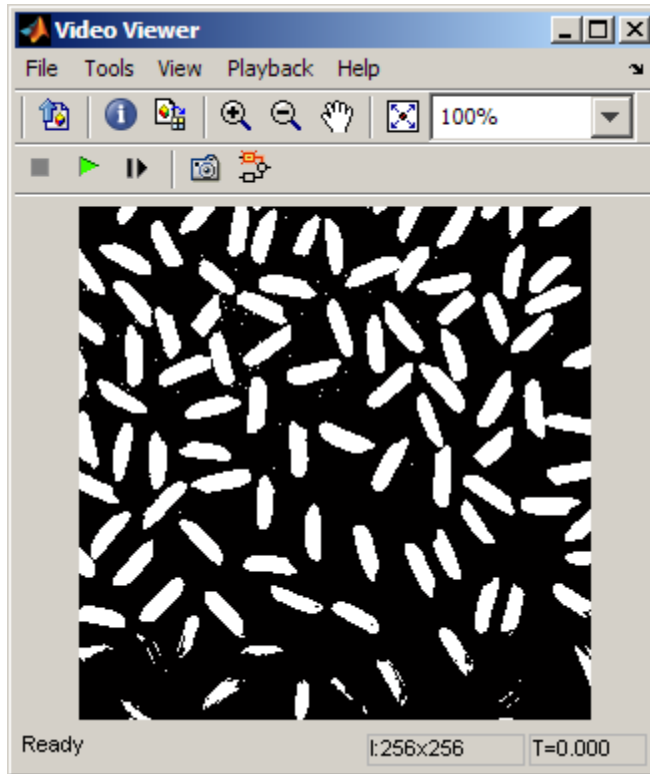


- 8 Double-click the Image From File block. On the **Data Types** pane, set the **Output data type** parameter to double.
- 9 If you have not already done so, set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set 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.

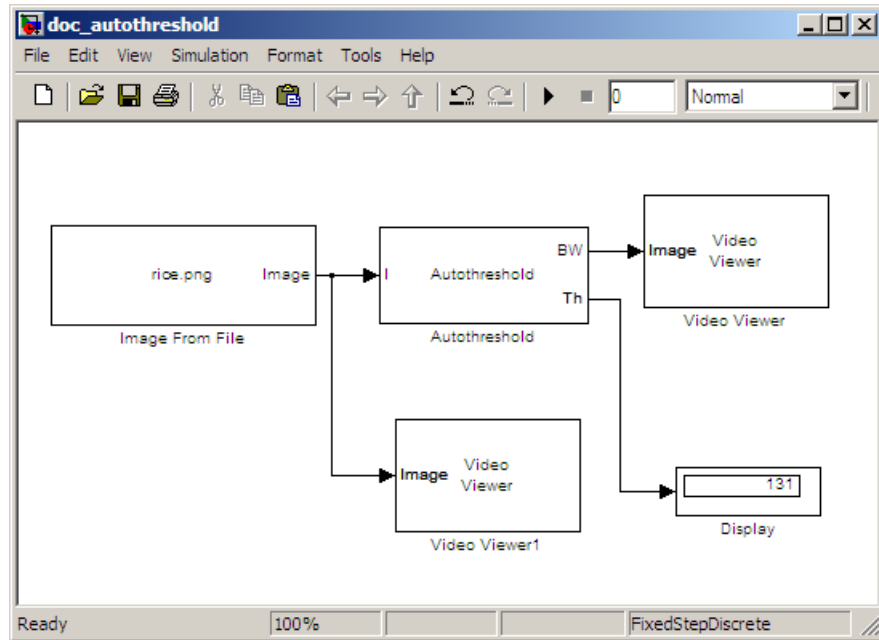
The original intensity image appears in the Video Viewer1 window.



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 *Video and Image Processing Blockset Reference*. To open a demo model that uses this block, type `vipstaples` at the MATLAB command prompt.

Color Space Conversion

In this section...

“Overview of Color Space Conversion Block” on page 5-14

“Converting Color Information from R’G’B’ to Intensity” on page 5-14

Overview of Color Space Conversion Block

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.

Converting Color Information from R’G’B’ to Intensity

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:

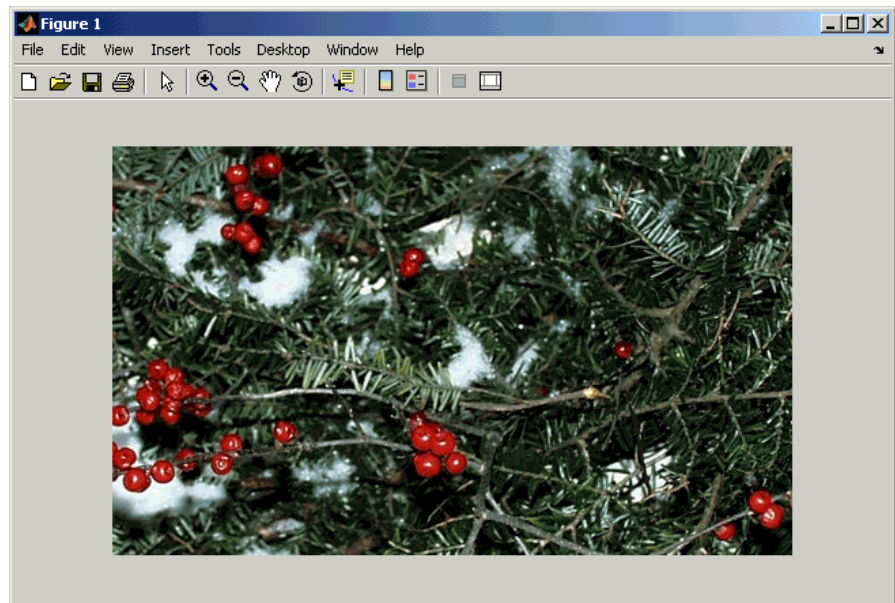
- 1 Define an R’G’B’ image in the MATLAB workspace. To read in an R’G’B’ image from a JPG file, at the MATLAB command prompt, type

```
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

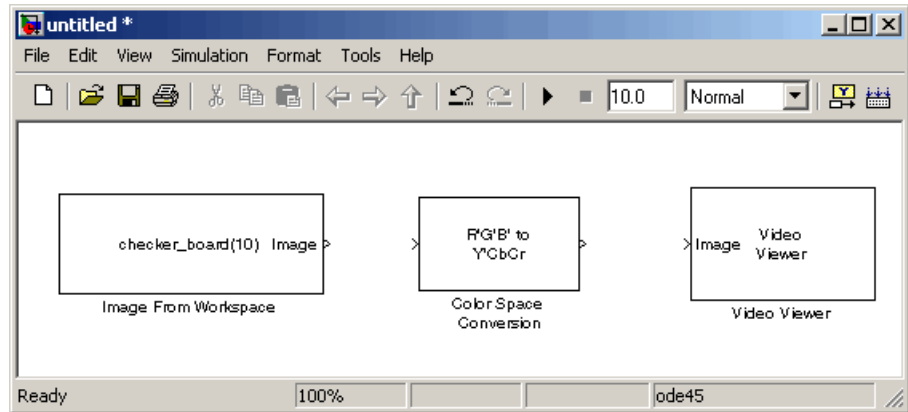
```
imshow(I)
```



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

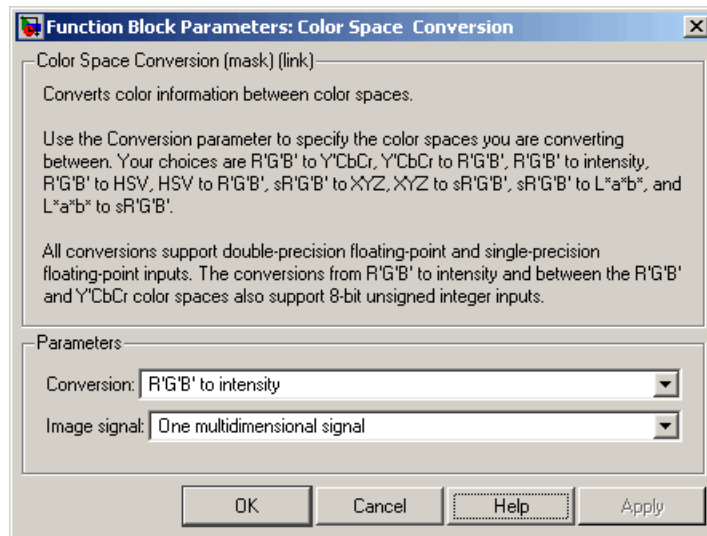
Block	Library	Quantity
Image From Workspace	Video and Image Processing Blockset > Sources	1
Color Space Conversion	Video and Image Processing Blockset > Conversions	1
Video Viewer	Video and Image Processing Blockset > Sinks	1

- 4** Position the blocks as shown in the following figure.

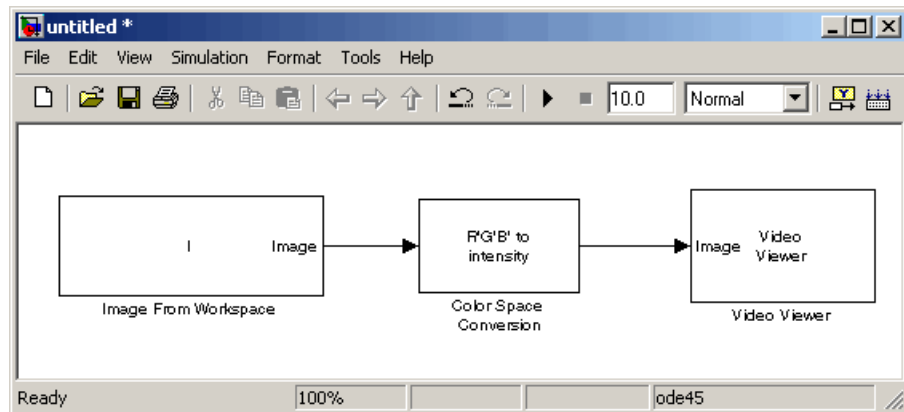


Once you have assembled the blocks needed to convert a R'G'B' image to an intensity image, you are ready to set your block parameters. To do this, double-click the blocks, modify the block parameter values, and click **OK**.

- 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.



- 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.

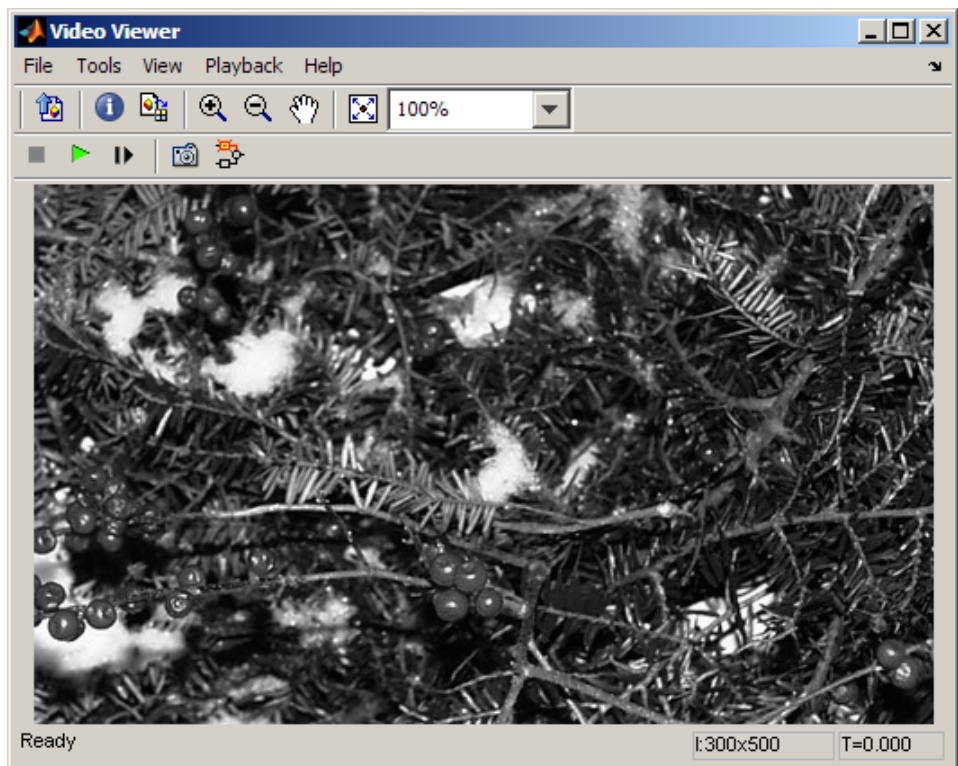


- 9 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

- 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 in the *Video and Image Processing Blockset Reference*.

Chroma Resampling

The YCbCr 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. Because the human eye is more sensitive to changes in luma than to changes in chroma, 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. In the following example, you use the Chroma Resampling block to downsample the Cb and Cr components of an image:

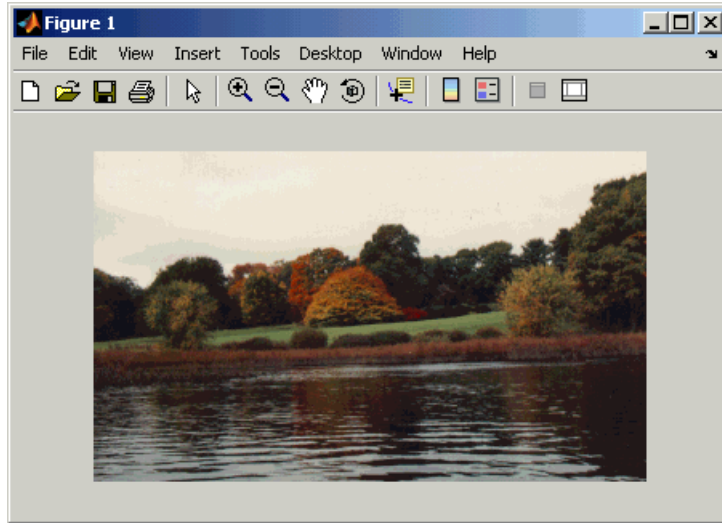
- 1 Define an RGB image in the MATLAB workspace. To read in an RGB image from a TIF file, at the MATLAB command prompt, type

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

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

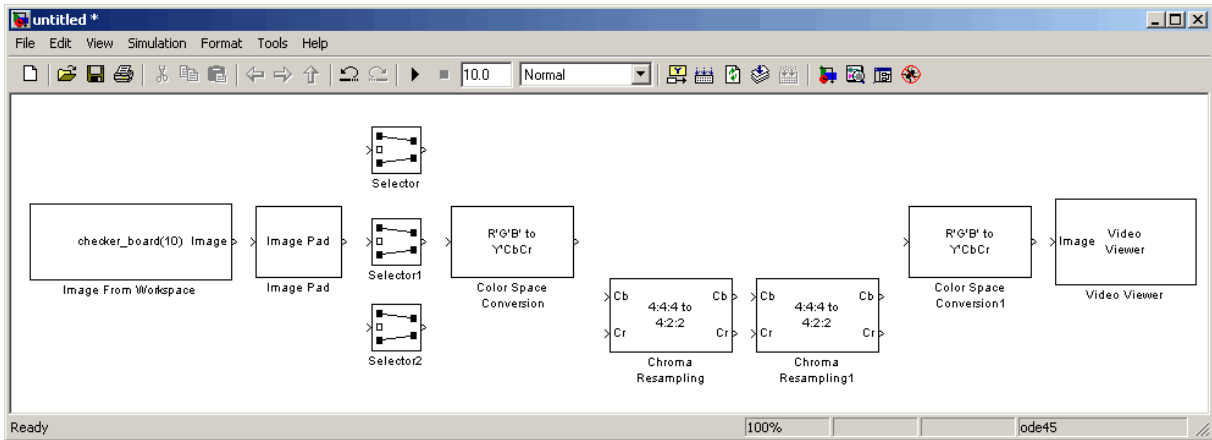
```
imshow(I)
```



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

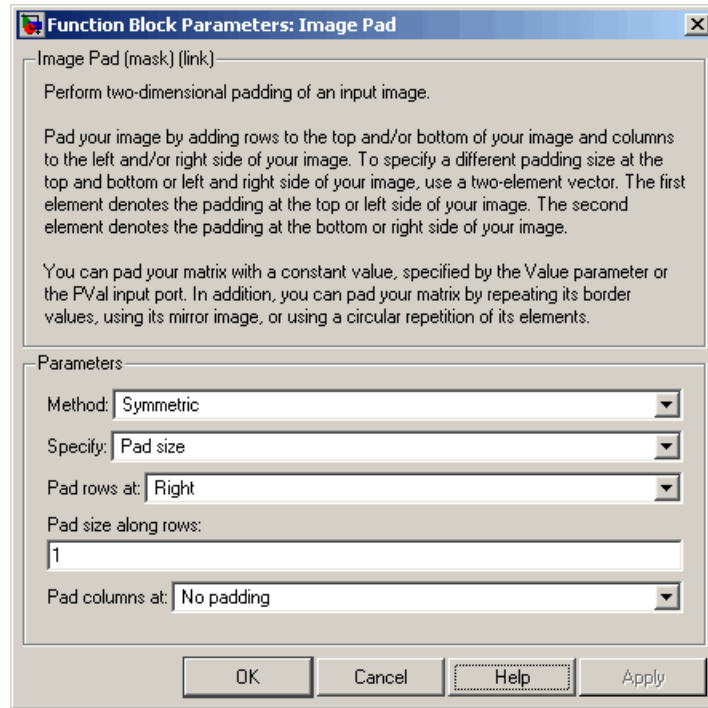
Block	Library	Quantity
Image From Workspace	Video and Image Processing Blockset > Sources	1
Image Pad	Video and Image Processing Blockset > Utilities	1
Color Space Conversion	Video and Image Processing Blockset > Conversions	2
Chroma Resampling	Video and Image Processing Blockset > Conversions	2
Video Viewer	Video and Image Processing Blockset > Sinks	1
Selector	Simulink > Signal Routing	3

4 Position the blocks as shown in the following figure.



The blocks to the left of and including the Chroma Resampling block represent the transmission portion of the model. The remaining blocks represent the receiving portion of the model. Once you have assembled these blocks, you are ready to set your block parameter values. 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 Image Pad block to change the dimensions of the 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
 - **Pad rows at** = Right
 - **Pad size along rows** = 1
 - **Pad columns at** = 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 processing video streams, it is computationally expensive to pad every video frame. You should change the dimensions of the video stream before you process it with Video and Image Processing Blockset blocks.

7 Use the Selector blocks to separate the individual color planes from the main signal. This simplifies the color space conversion section of the model. Set the Selector block parameters as follows:

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

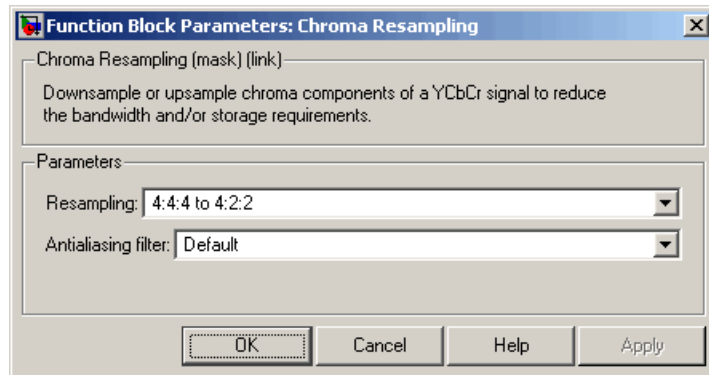
Set the Selector1 block parameters as follows:

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

Set the Selector2 block parameters as follows:

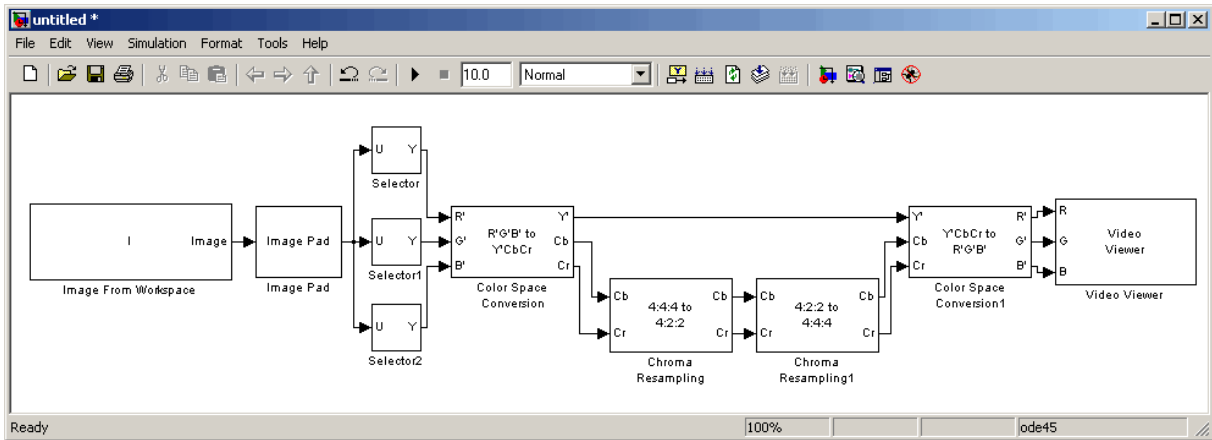
- **Number of input dimensions = 3**
- 1
 - **Index Option = Select all**
- 2
 - **Index Option = Select all**
- 3
 - **Index Option = Index vector (dialog)**
 - **Index = 3**

- 8 Use the Color Space Conversion block to 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**.
- 9 Use the Chroma Resampling block to 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.

- 10 Use the Chroma Resampling1 block to 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**.
- 11 Use the Color Space Conversion1 block to 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
- 12 Use the Video Viewer block to display the recovered image. Set the **Image signal** parameter to **Separate color signals**.
- 13 Connect the blocks as shown in the following figure.



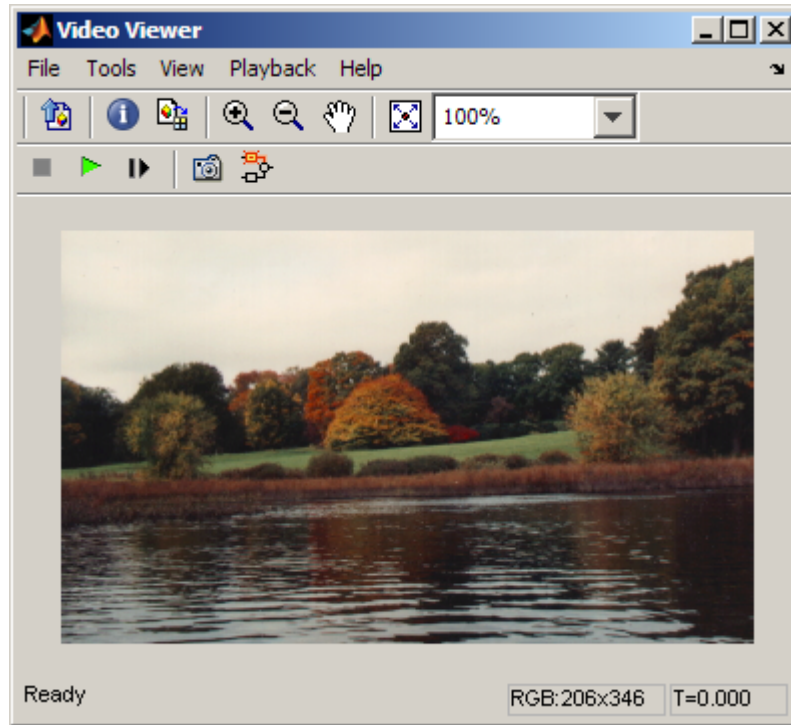
14 Configure Simulink to display signal dimensions next to each signal line. Click **Format > Port/Signal Displays > Signal Dimensions**.

15 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)

16 Run the model.

The recovered image appears in the Video Viewer window.



- 17** 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.

You have used the Chroma Resampling block to downsample the Cb and Cr components of an image. For more information about this block, see the Chroma Resampling block reference page in the *Video and Image Processing Blockset Reference*.

Geometric Transformation

- “Geometric Transformation Interpolation Methods” on page 6-2
- “Rotating an Image” on page 6-6
- “Resizing an Image” on page 6-14
- “Cropping an Image” on page 6-20

Geometric Transformation Interpolation Methods

In this section...
“Overview of Interpolation Methods” on page 6-2
“Nearest Neighbor Interpolation” on page 6-2
“Bilinear Interpolation” on page 6-3
“Bicubic Interpolation” on page 6-4

Overview of Interpolation Methods

The Geometric Transformations library of Video and Image Processing Blockset software contains blocks that perform geometric transformations. These blocks use interpolation to calculate the appropriate pixel values so that images appear rotated, translated, resized, or sheared.

Note The examples in the following sections are illustrations of interpolation methods. The block algorithms are implemented in a slightly different way so that they are optimized for speed and memory.

Nearest Neighbor Interpolation

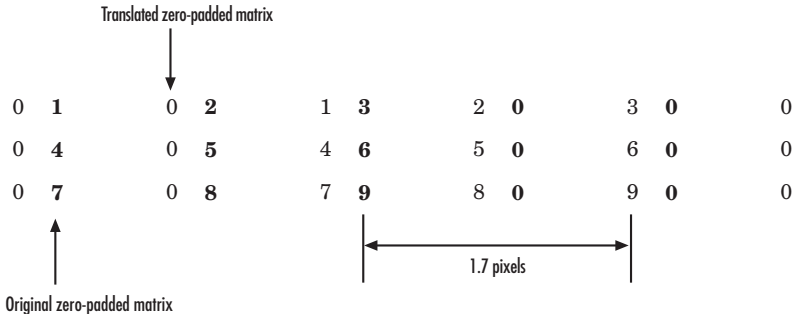
For nearest neighbor interpolation, the block uses the value of nearby translated pixel values for the output pixel values.

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

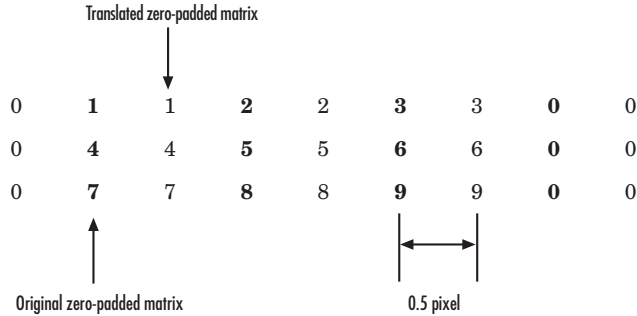
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

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

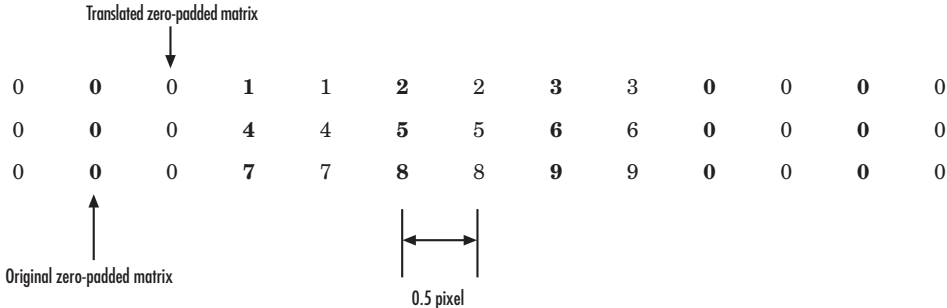
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

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

Rotating 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

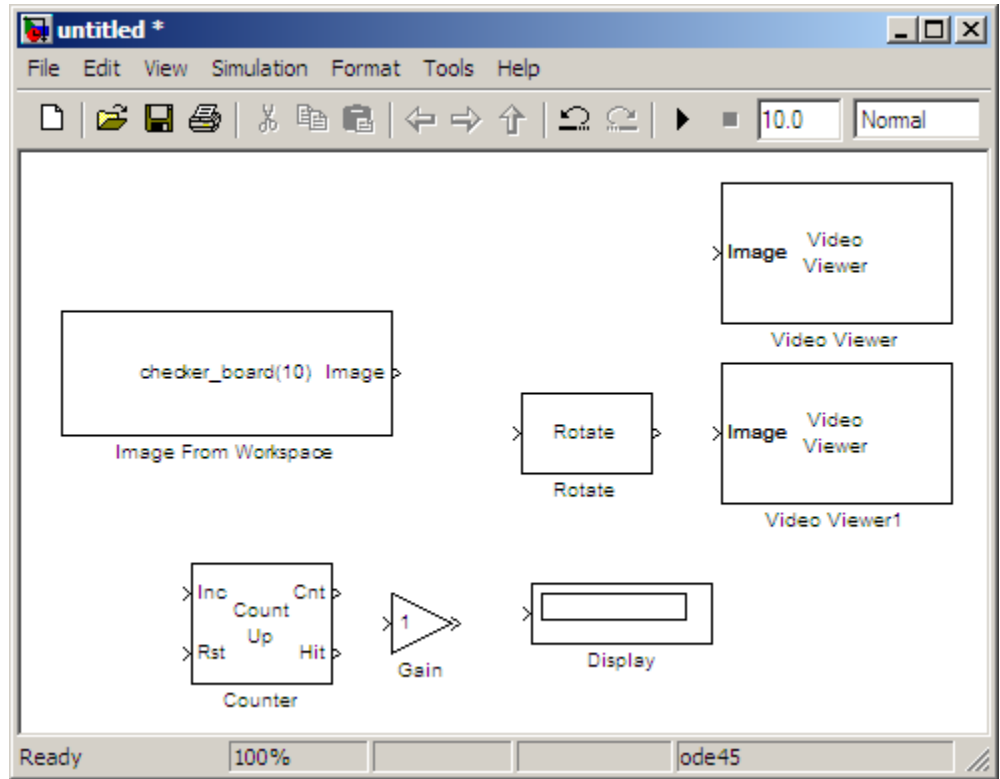
```
imshow(I)
```



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

Block	Library	Quantity
Image From Workspace	Video and Image Processing Blockset > Sources	1
Rotate	Video and Image Processing Blockset > Geometric Transformations	1
Video Viewer	Video and Image Processing Blockset > Sinks	2
Gain	Simulink > Math Operations	1
Display	Signal Processing Blockset > Signal Processing Sinks	1
Counter	Signal Processing Blockset > Signal Management > Switches and Counters	1

4 Position the blocks as shown in the following figure.

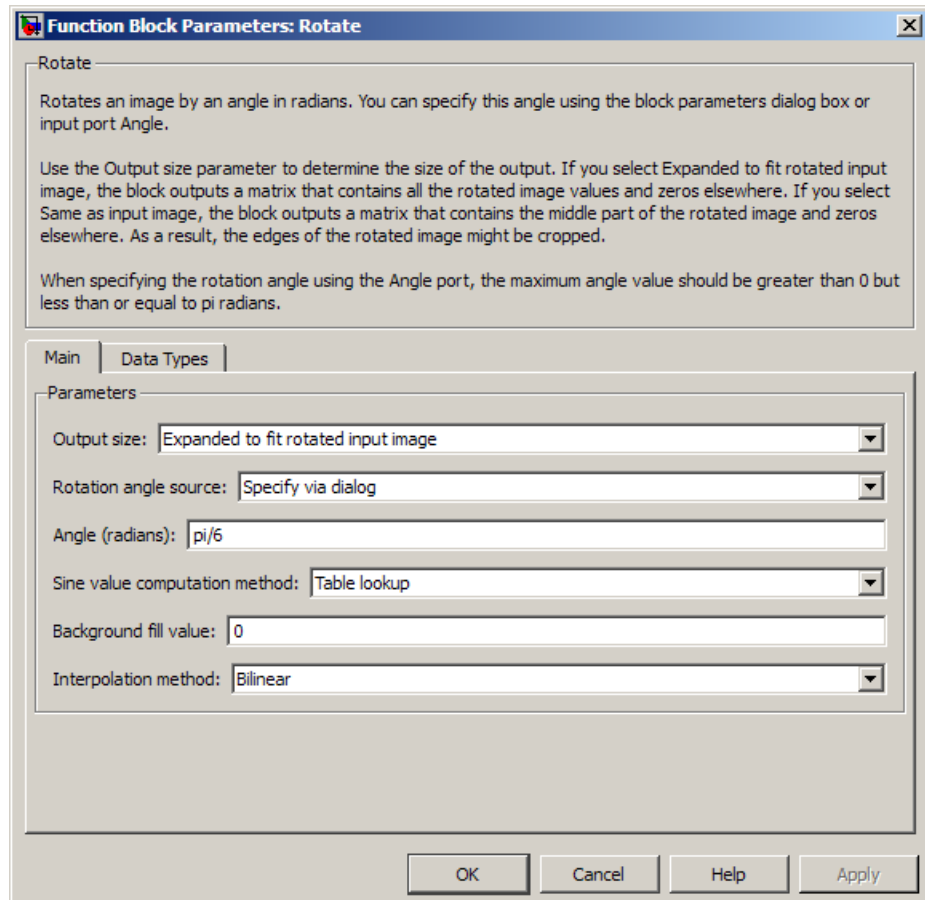


You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

- 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

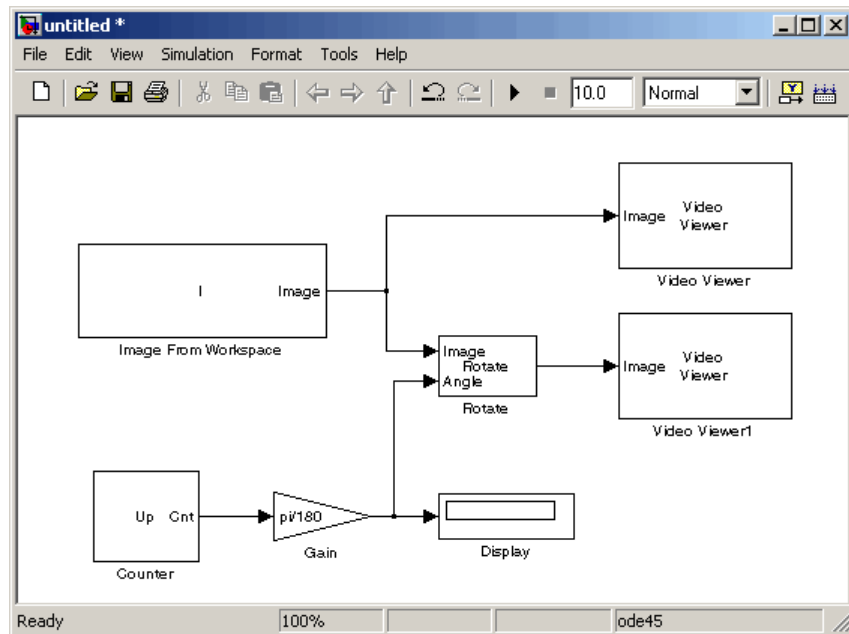


The Angle port appears on the block. You use this port to input a steadily increasing angle. Setting the **Output size** parameter to Expanded to fit rotated input image ensures that the block does not crop the output.

- 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. You are using the Counter block from Signal Processing Blockset software because its **Count data type** parameter enables you to specify the data type of its output.

- 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.

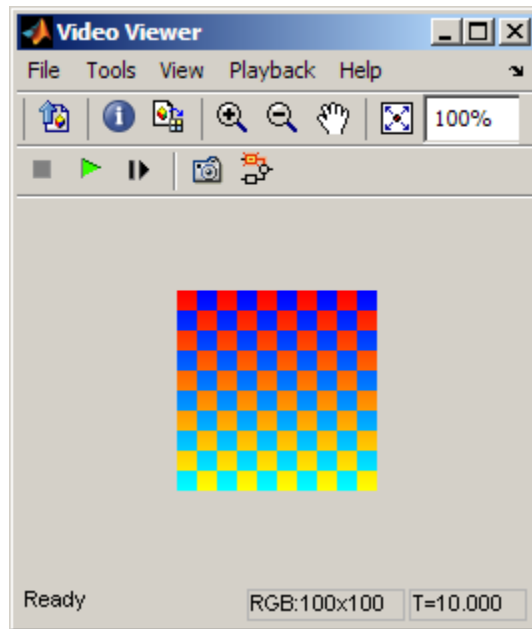


12 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

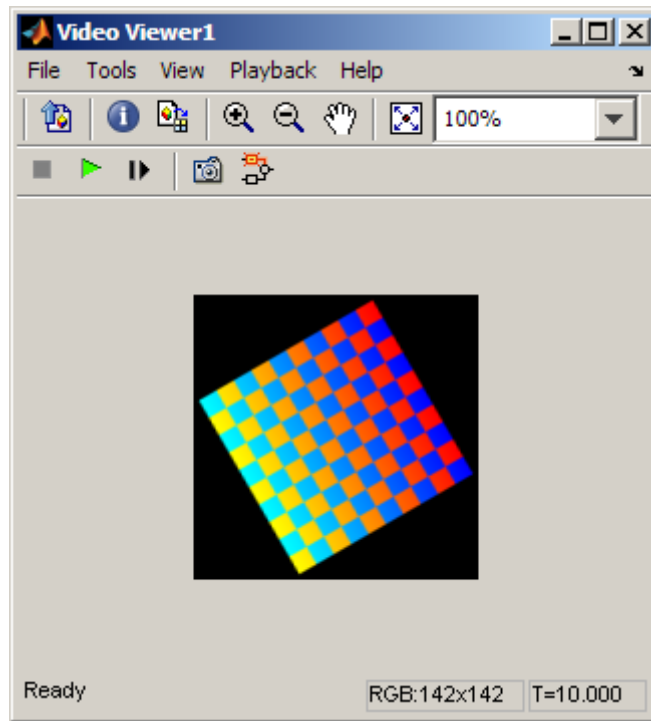
- **Solver** pane, **Stop time** = inf
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

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 *Video and Image Processing Blockset 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.

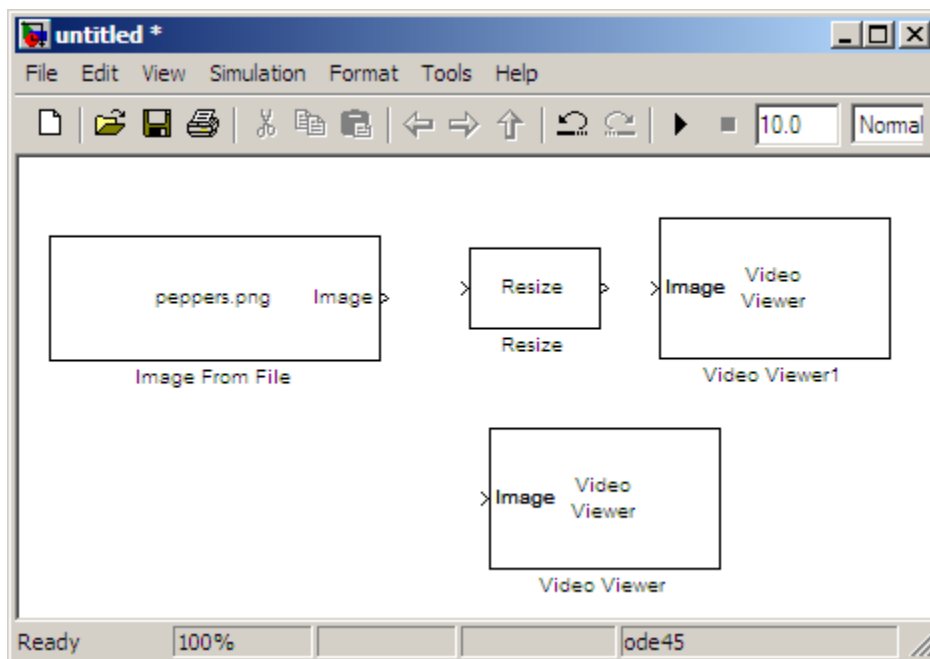
Resizing an Image

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.

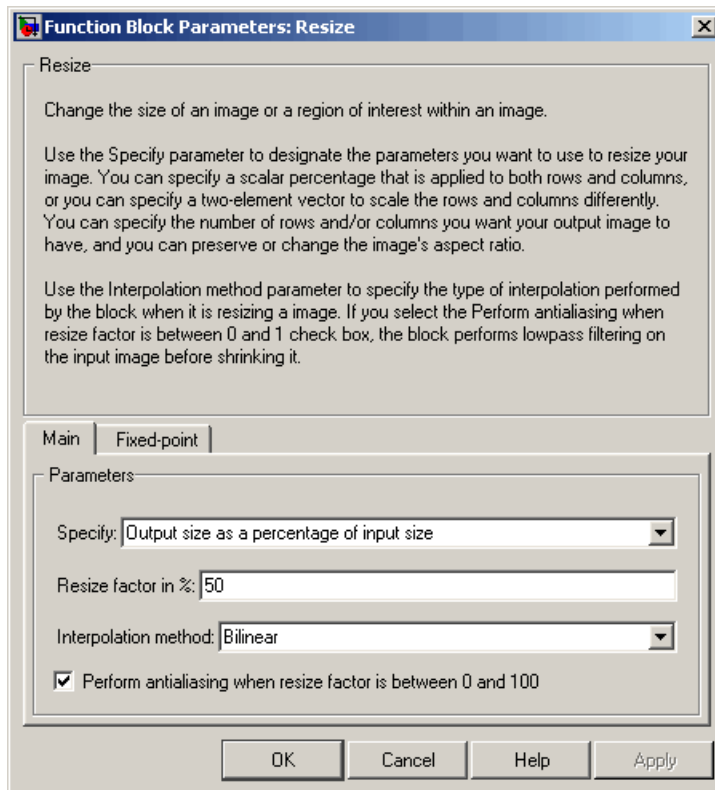
Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Resize	Video and Image Processing Blockset > Geometric Transformations	1
Video Viewer	Video and Image Processing Blockset > Sinks	2

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



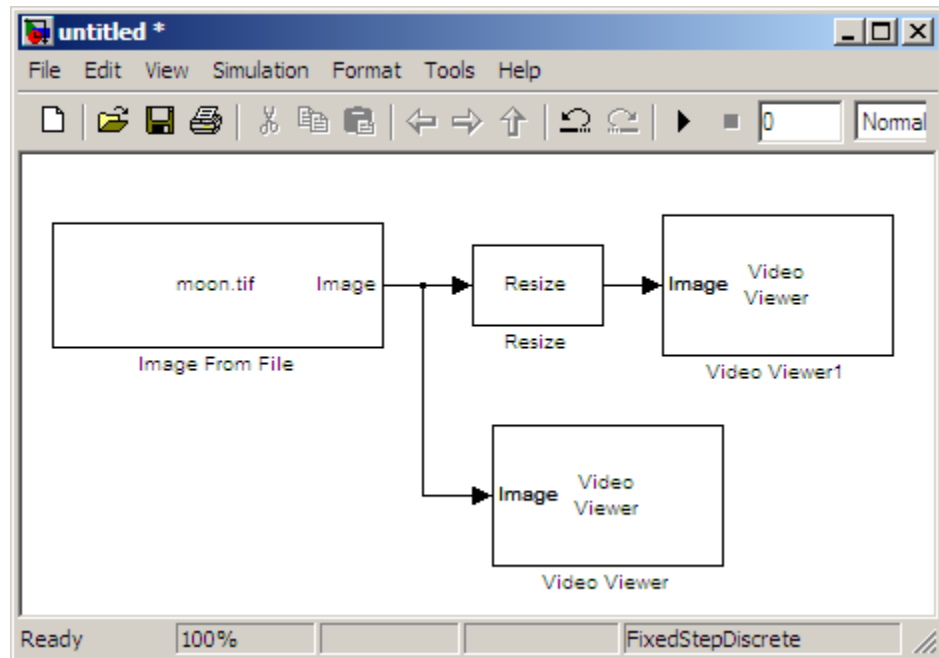
- 3 Use the Image From File block to import the intensity image. Set the **File name** parameter to `moon.tif`. The tif file is a 537-by-358 matrix of 8-bit unsigned integer values.
- 4 Use the Video Viewer block to display the original image. Accept the default parameters.

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.



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.

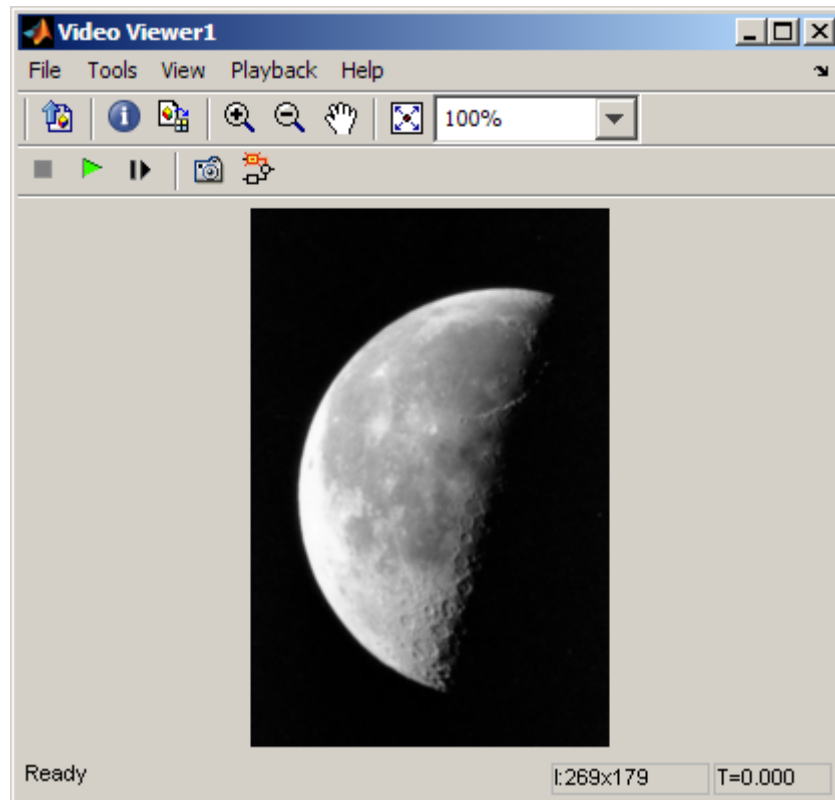


- 8 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
 - **Solver** pane, **Stop time** = 0
 - **Solver** pane, **Type** = Fixed-step
 - **Solver** pane, **Solver** = Discrete (no 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 in the *Video and Image Processing Blockset Reference*. For more information about other geometric transformation blocks, see the Rotate, Shear, and Translate block reference pages.

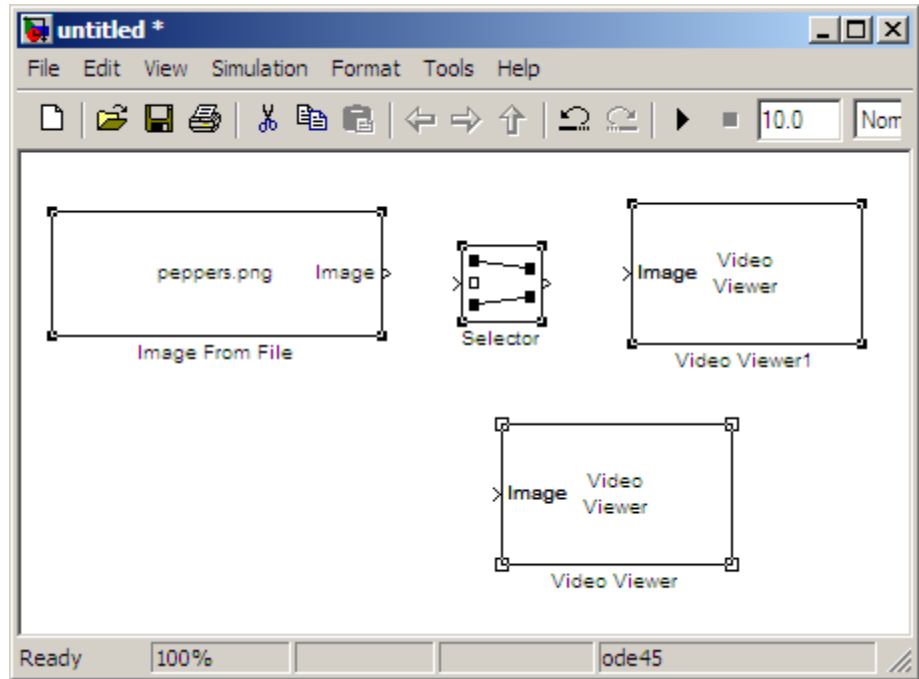
Cropping an Image

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	Video and Image Processing Blockset > Sources	1
Video Viewer	Video and Image Processing Blockset > Sinks	2
Selector	Simulink > Signal Routing	1

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

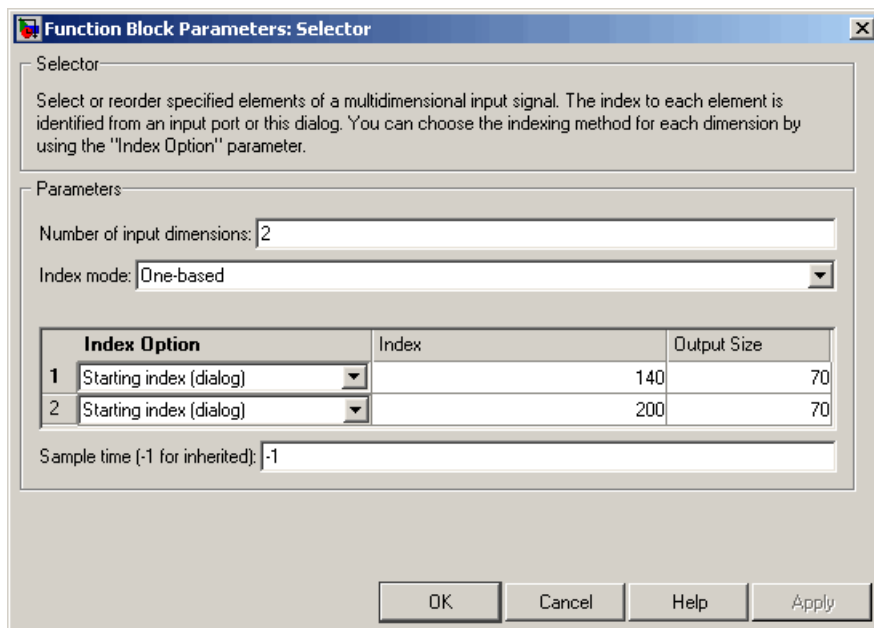


- 3 Use the Image From File block to import the intensity image. Set the **File name** parameter to `coins.png`. The image is a 246-by-300 matrix of 8-bit unsigned integer values.
- 4 Use the Video Viewer block to display the original image. Accept the default parameters.

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

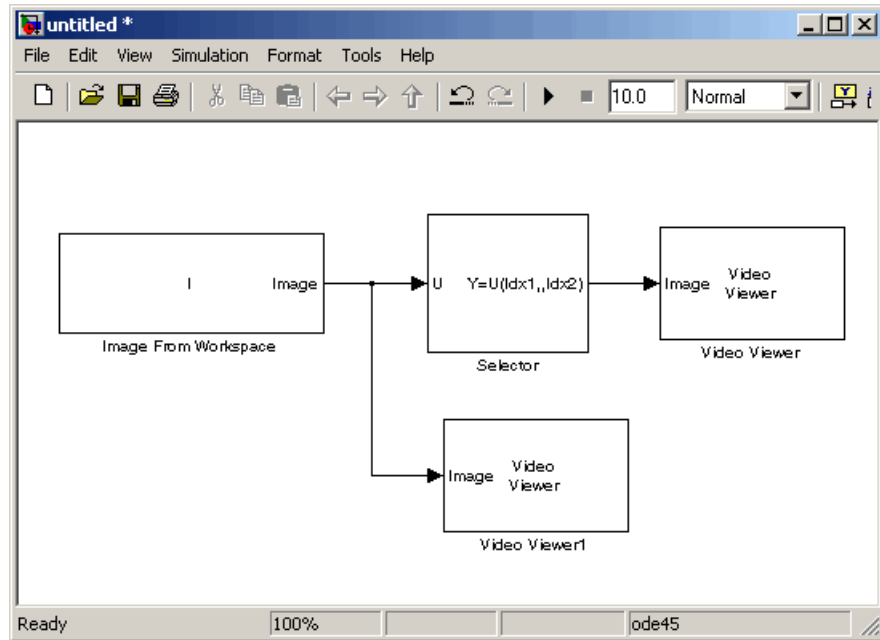


The Selector block starts at row 140 and column 200 of the image and outputs the next 70 rows and columns of the image.

- 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:

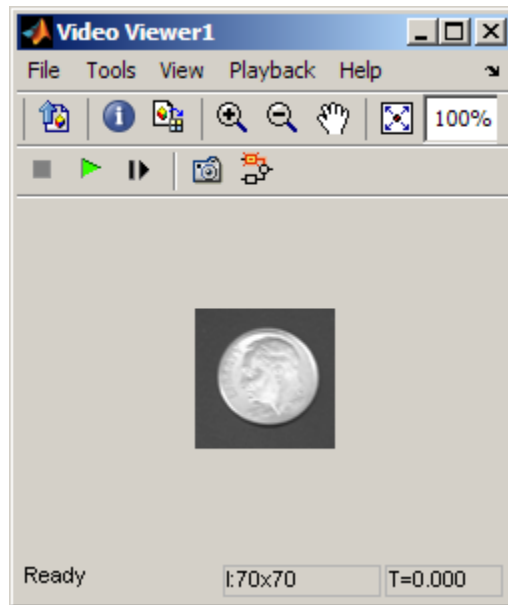
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

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.

Morphological Operations

- “Overview of Morphology” on page 7-2
- “Counting Objects in an Image” on page 7-3
- “Correcting for Nonuniform Illumination” on page 7-10

Overview of Morphology

Morphology is the study of the shape and form of objects. Morphological image analysis can be used to perform

- Object extraction
- Image filtering operations, such as removal of small objects or noise from an image
- Image segmentation operations, such as separating connected objects
- Measurement operations, such as texture analysis and shape description

The Video and Image Processing Blockset software contains blocks that perform morphological operations such as erosion, dilation, opening, and closing. Often, you need to use a combination of these blocks to perform your morphological image analysis. Morphological image analysis can be used to perform image filtering, image segmentation, and measurement operations.

The examples in this chapter show you how to use blocks from the Morphological Operations library to count the number of objects in an image and how to correct for uneven illumination.

For more information, see “Morphological Operations” in the Image Processing Toolbox documentation.

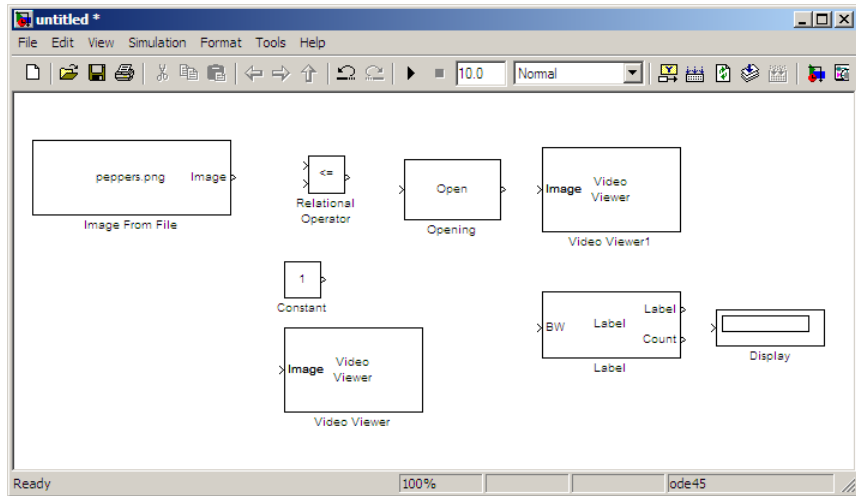
Counting Objects in an Image

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	Video and Image Processing Blockset > Sources	1
Opening	Video and Image Processing Blockset > Morphological Operations	1
Label	Video and Image Processing Blockset > Morphological Operations	1
Video Viewer	Video and Image Processing Blockset > Sinks	2
Constant	Simulink > Sources	1
Relational Operator	Simulink > Logic and Bit Operations	1
Display	Signal Processing Blockset > Signal Processing Sinks	1

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

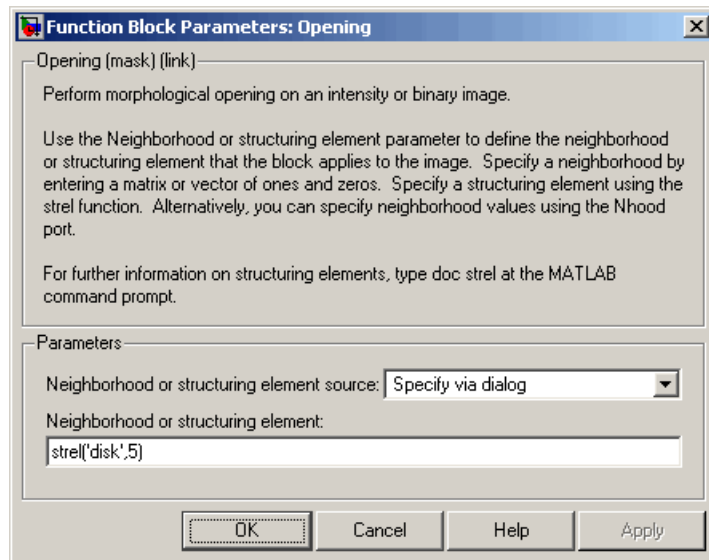


You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

- 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 `<`.

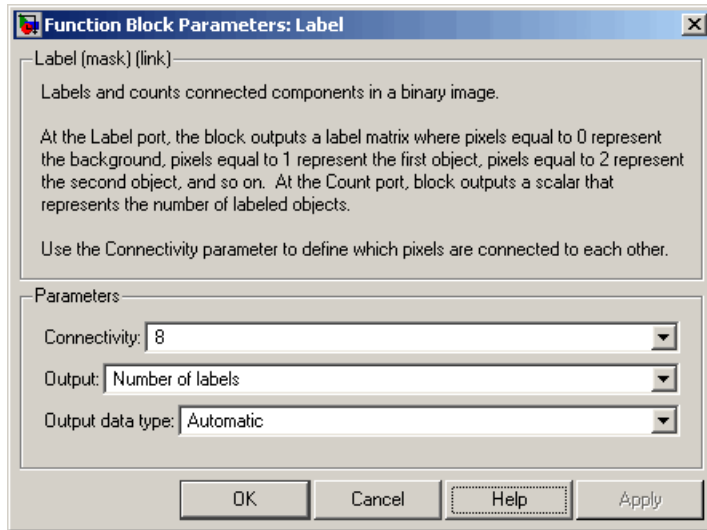
If the input to the Relational Operator block is less than 200, its output is 1; otherwise, its output is 0. You must threshold your intensity image because the Label block expects binary input. Also, the objects it counts must be white.

- 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.



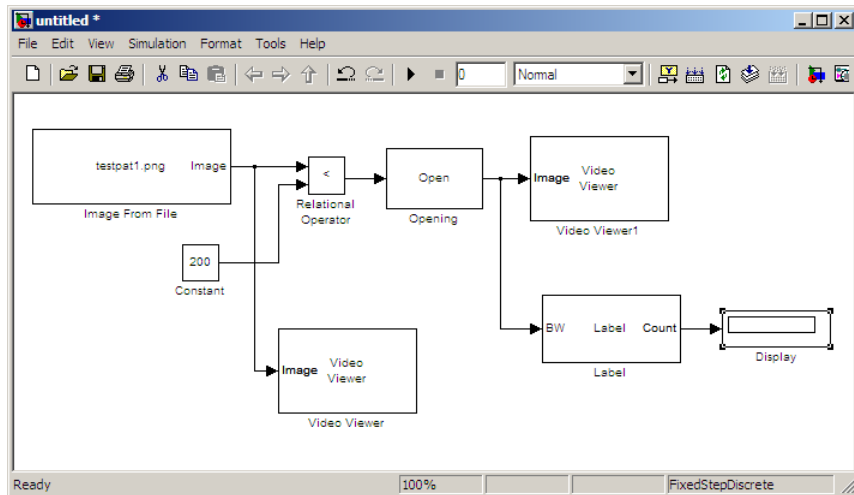
The `strel` function creates a circular STREL object with a radius of 5 pixels. When working with the Opening block, pick a STREL object that fits within the objects you want to keep. It often takes experimentation to find the neighborhood or STREL object that best suits your application.

- 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.



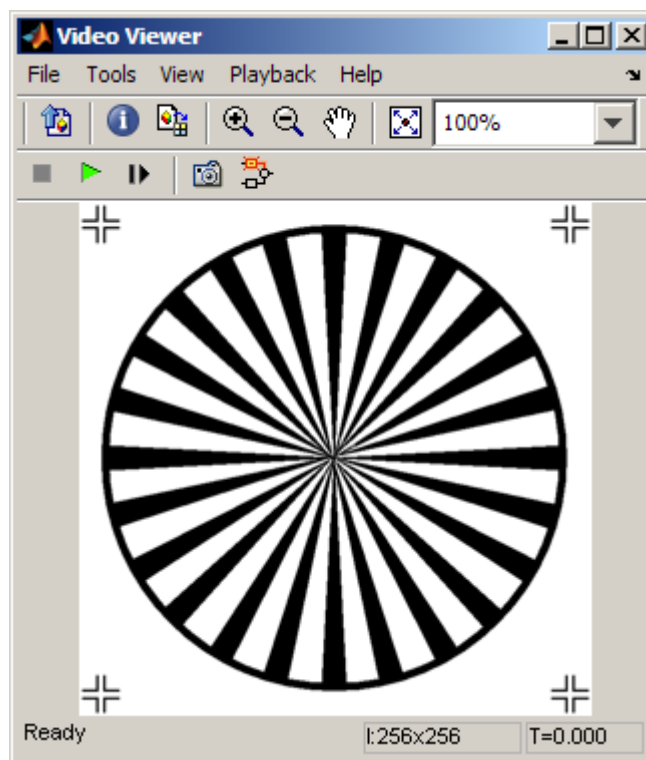
10 The Display block displays the number of spokes in the input image. Use the default parameters.

11 Connect the block as shown in the following figure.

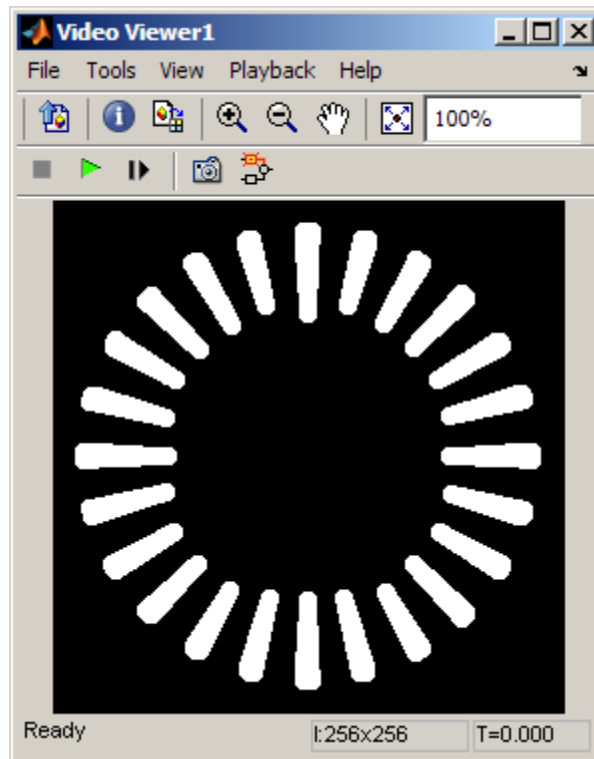


- 12** Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
- **Solver** pane, **Stop time** = 0
 - **Solver** pane, **Type** = Fixed-step
 - **Solver** pane, **Solver** = discrete (no continuous states)
- 13** Run the model.

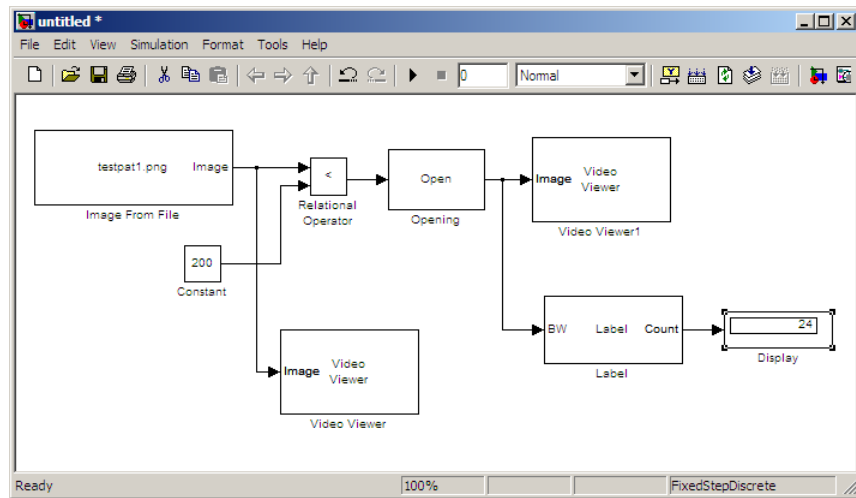
The original image appears in the Video Viewer1 window. To view the image at its true size, right-click the window and select **Set Display To True Size**.



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 *Video and Image Processing Blockset 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 Signal Processing Blockset. For more information about STREL objects, see `strel` in the Image Processing Toolbox documentation.

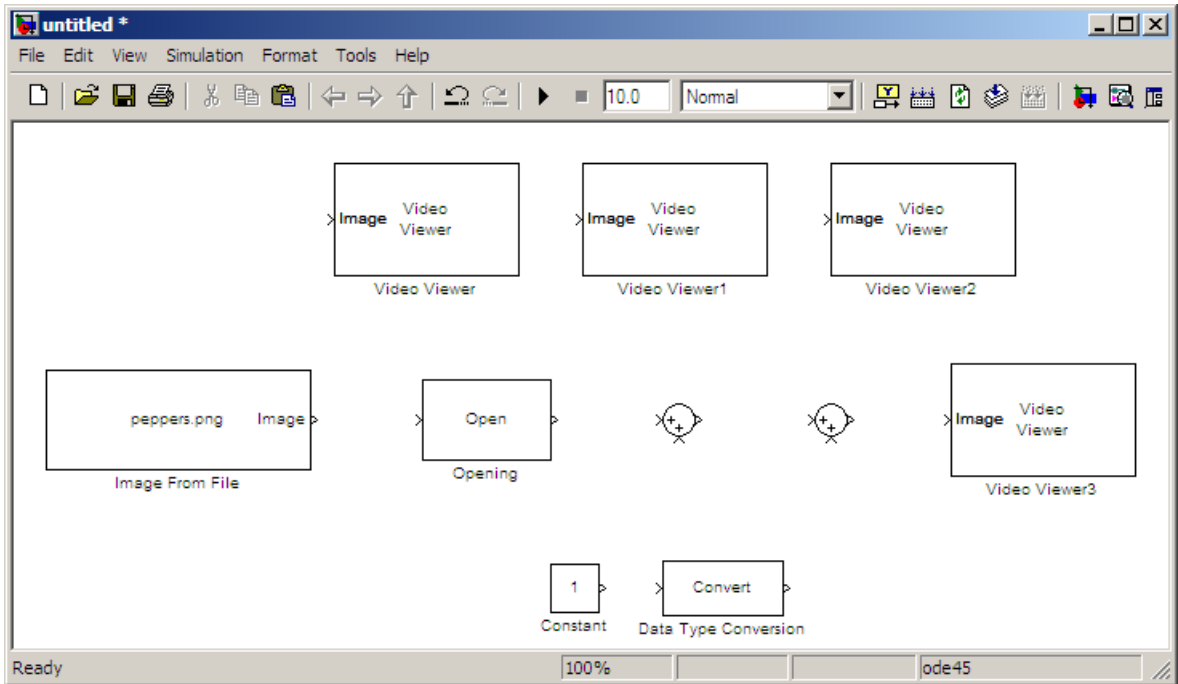
Correcting for Nonuniform Illumination

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.

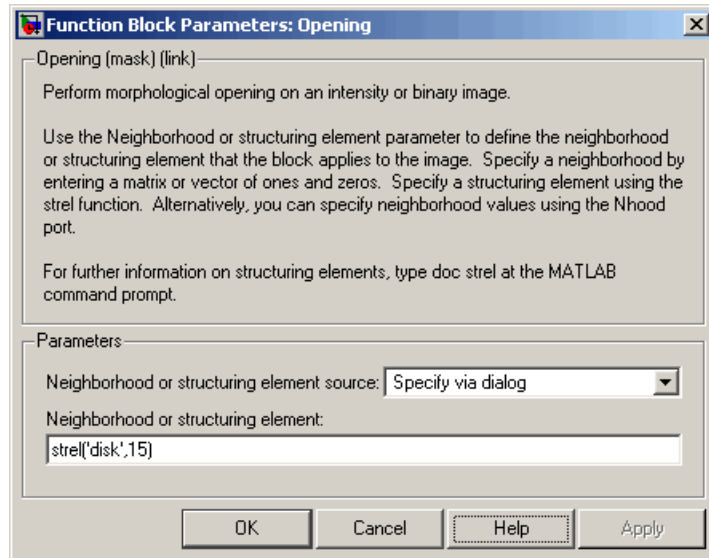
Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Opening	Video and Image Processing Blockset > Morphological Operations	1
Video Viewer	Video and Image Processing Blockset > Sinks	4
Constant	Simulink > Sources	1
Sum	Simulink > Math Operations	2
Data Type Conversion	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)`.



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.

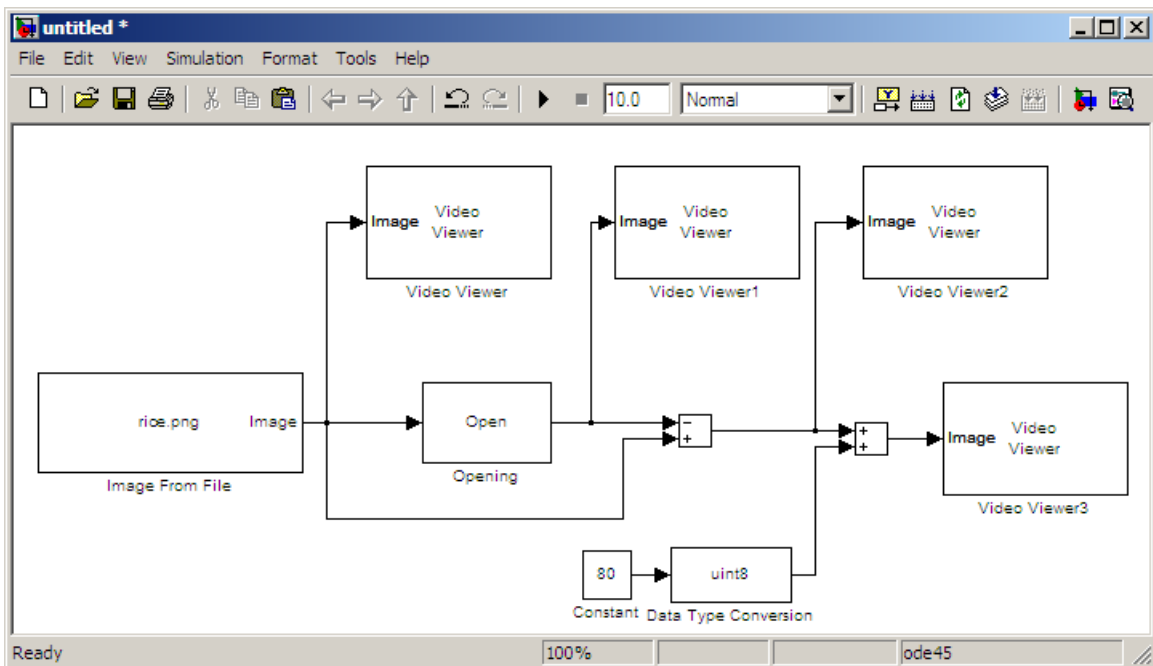
- 6 Use the Video Viewer1 block to view the background estimated by the Opening block. Accept the default parameters.
- 7 Use the first Sum block to subtract the estimated background from the original image. Set the block parameters as follows:
 - **Icon shape** = rectangular
 - **List of signs** = - +
- 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.

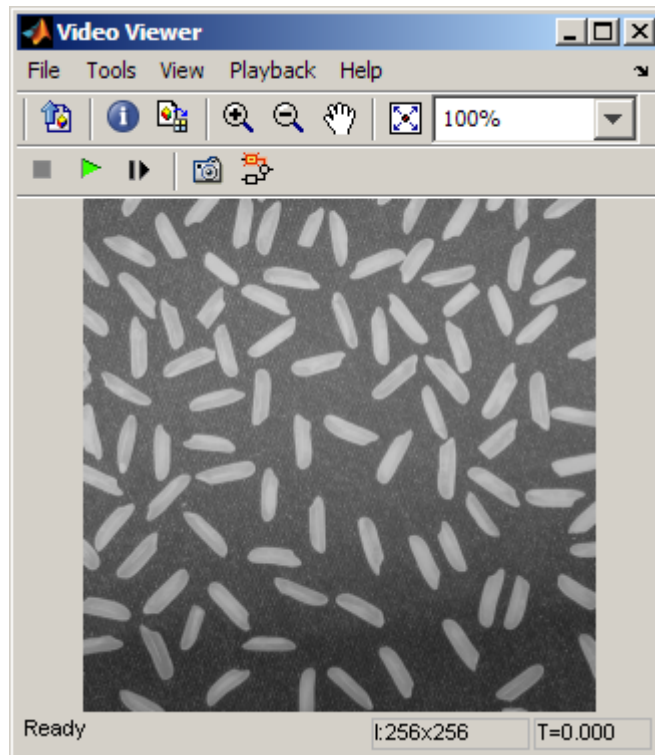


14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

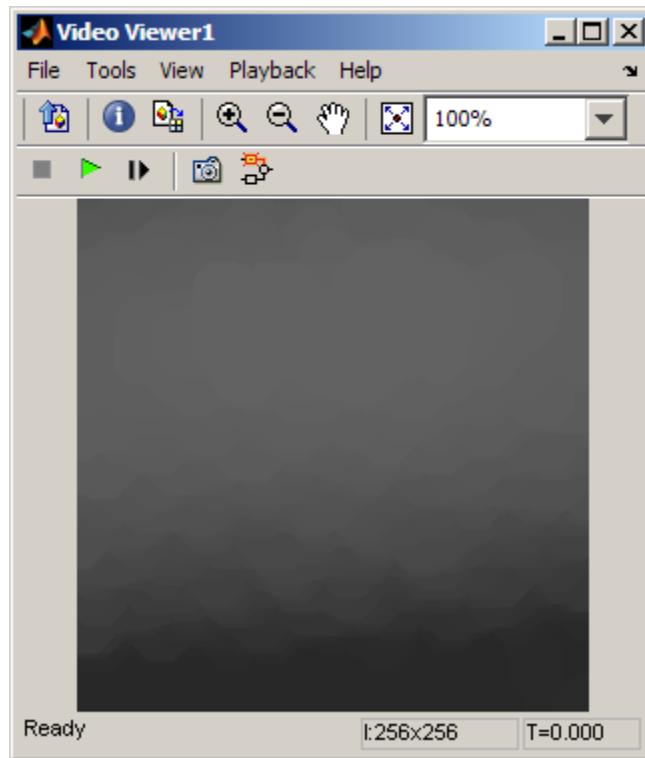
- **Solver** pane, **Stop time** = 0
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = discrete (no continuous states)

15 Run the model.

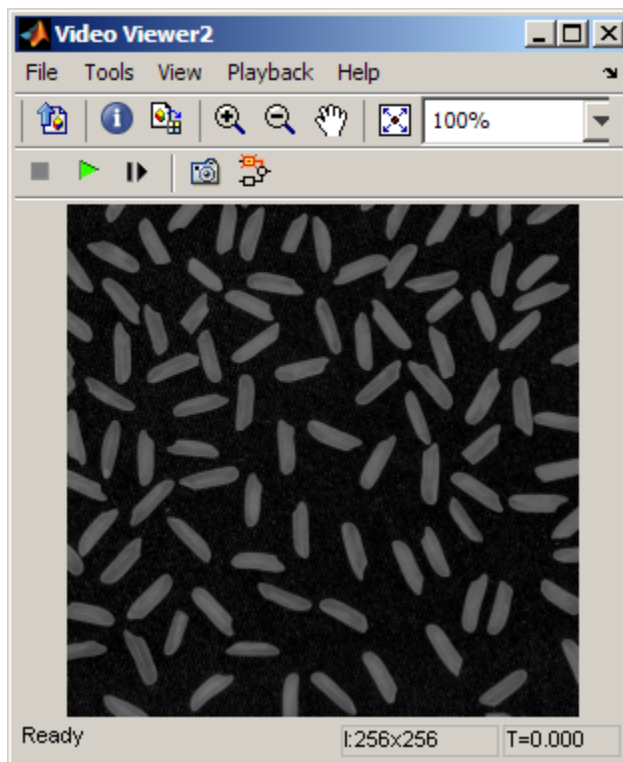
The original image appears in the Video Viewer window.



The estimated background appears in the Video Viewer1 window.

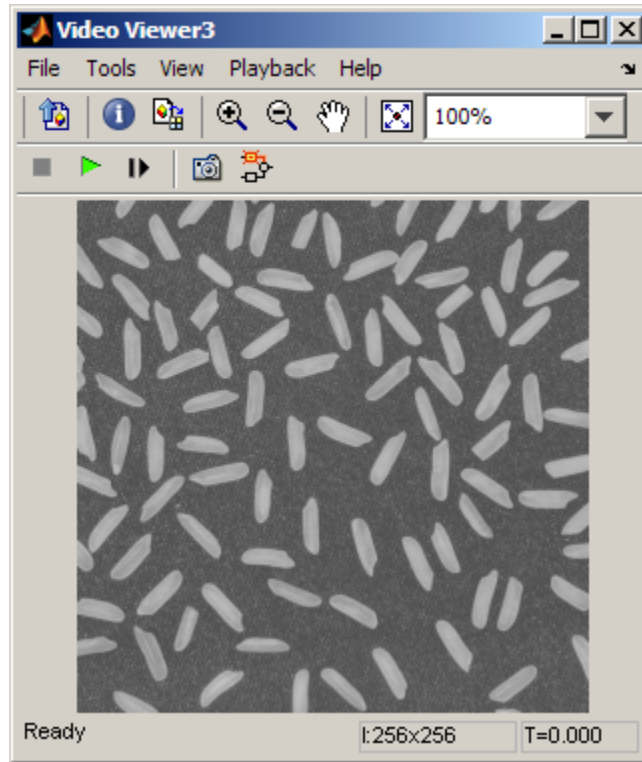


The image without the estimated background appears in the Video Viewer2 window.



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 block reference page in the *Video and Image Processing Blockset Reference*. 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.

Example Applications

- “Pattern Matching” on page 8-2
- “Motion Compensation” on page 8-10
- “Image Compression” on page 8-12

Pattern Matching

In this section...

“Overview of Pattern Matching” on page 8-2

“Tracking an Object Using Correlation” on page 8-2

Overview of Pattern Matching

Pattern matching can be used to recognize and/or locate specific objects in an image. It can be accomplished using several techniques, one of which is correlation. Correlation provides a direct measure of the similarity between two images. Though sensitive to the scaling or rotation of objects, normalized correlation is robust to changes in lighting.

Tracking an Object Using Correlation

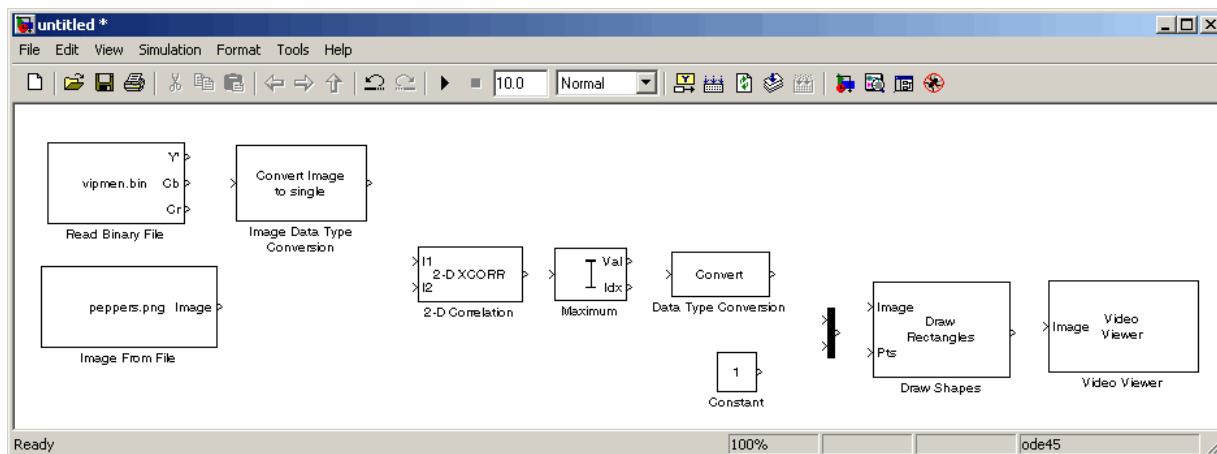
In this example, you use the 2-D Correlation, Maximum, and Draw Shapes blocks to find and indicate the location of a sculpture in each video frame:

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

Block	Library	Quantity
Read Binary File	Video and Image Processing Blockset > Sources	1
Image Data Type Conversion	Video and Image Processing Blockset > Conversions	1
Image From File	Video and Image Processing Blockset > Sources	1
2-D Correlation	Video and Image Processing Blockset > Statistics	1
Maximum	Video and Image Processing Blockset > Statistics	1
Draw Shapes	Video and Image Processing Blockset > Text & Graphics	1

Block	Library	Quantity
Video Viewer	Video and Image Processing Blockset > Sinks	1
Data Type Conversion	Simulink > Signal Attributes	1
Constant	Simulink > Sources	1
Mux	Simulink > Signal Routing	1

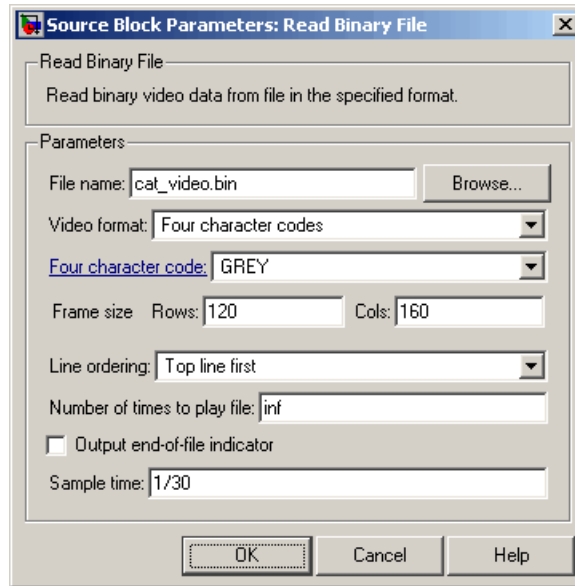
2 Position the blocks as shown in the following figure.



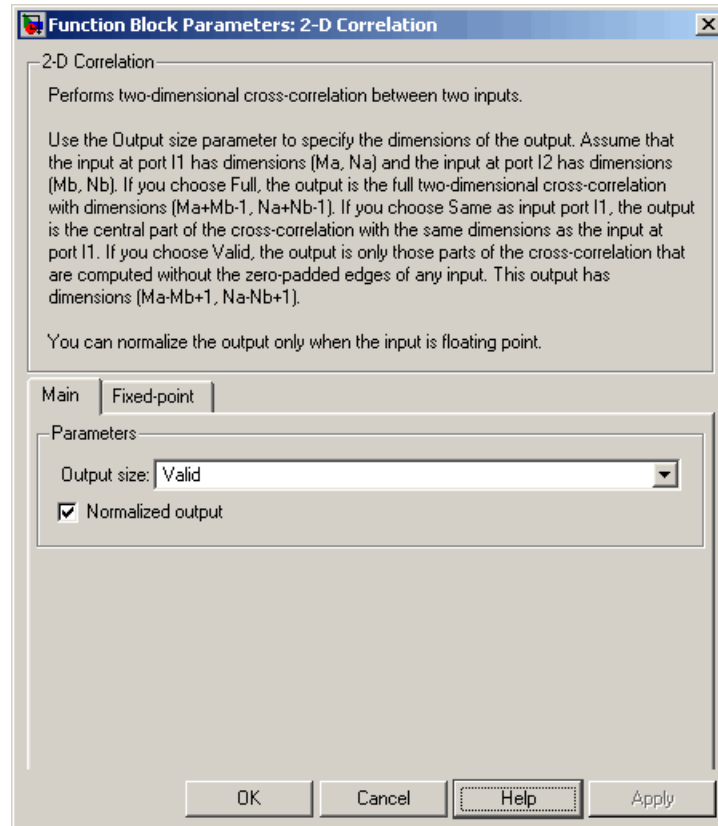
You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

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



- 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.



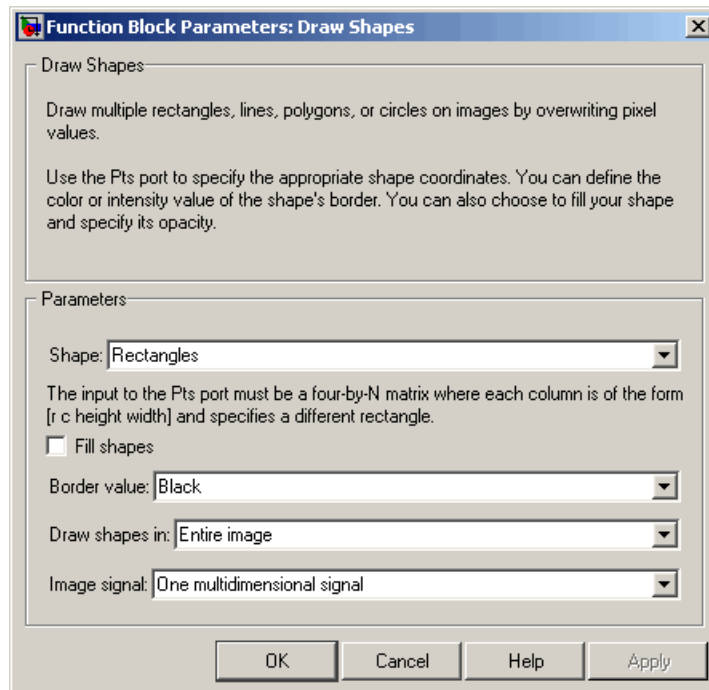
Because you chose **Valid** for the **Output size** parameter, the block outputs only those parts of the correlation that are computed without the zero-padded edges of any input.

- 7 Use the 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.

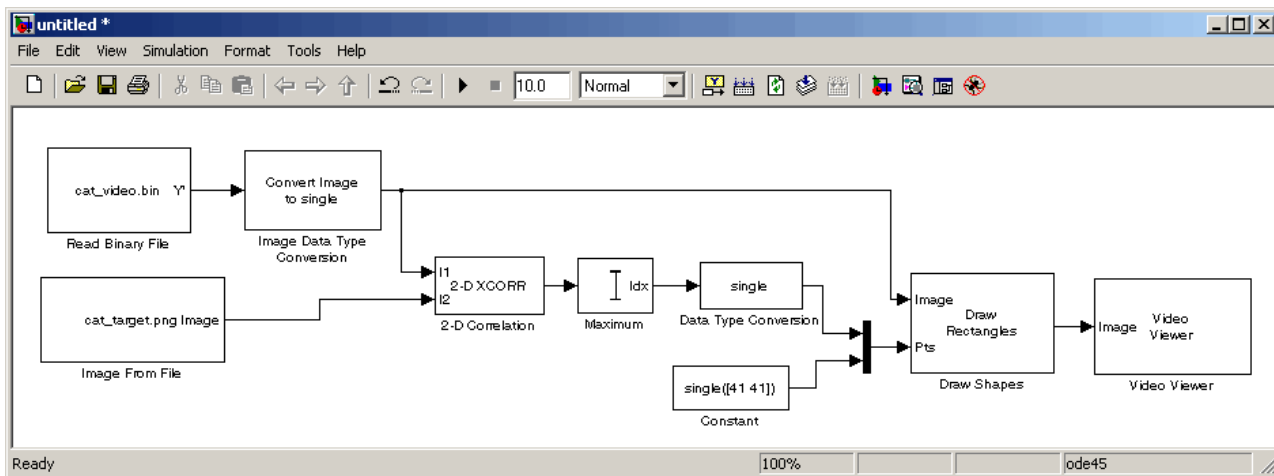


- 12 Use the Video Viewer block to display the video stream with the ROI displayed on it. Accept the default parameters.

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.

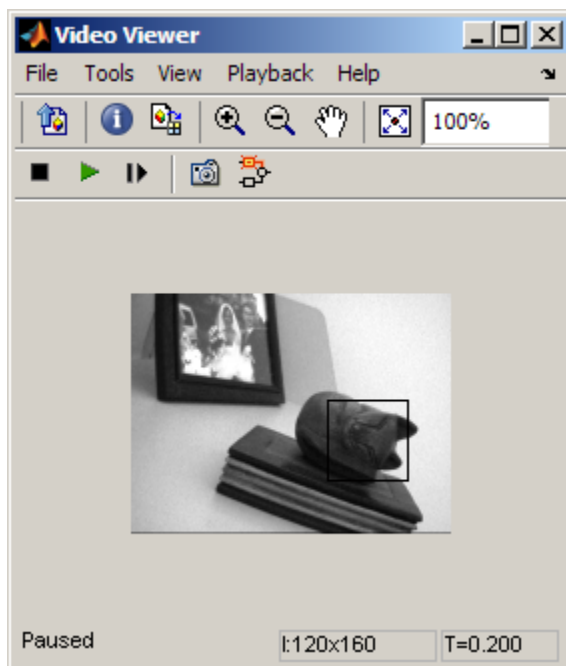


14 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:

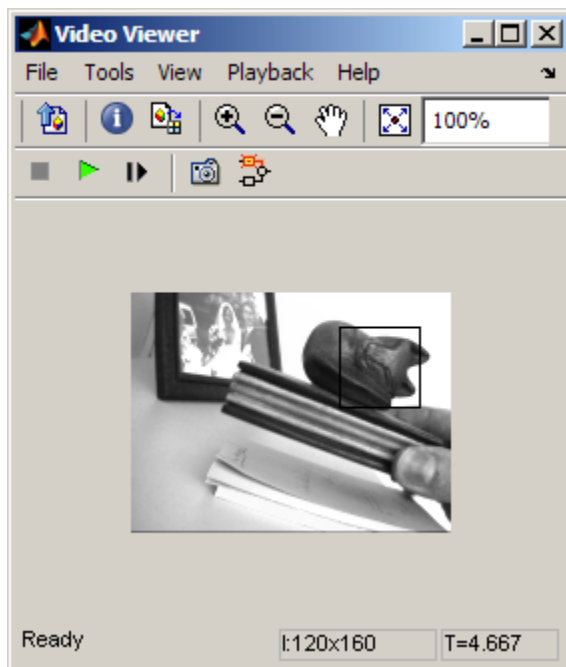
- **Solver** pane, **Stop time** = inf
- **Solver** pane, **Type** = Fixed-step
- **Solver** pane, **Solver** = Discrete (no continuous states)

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.



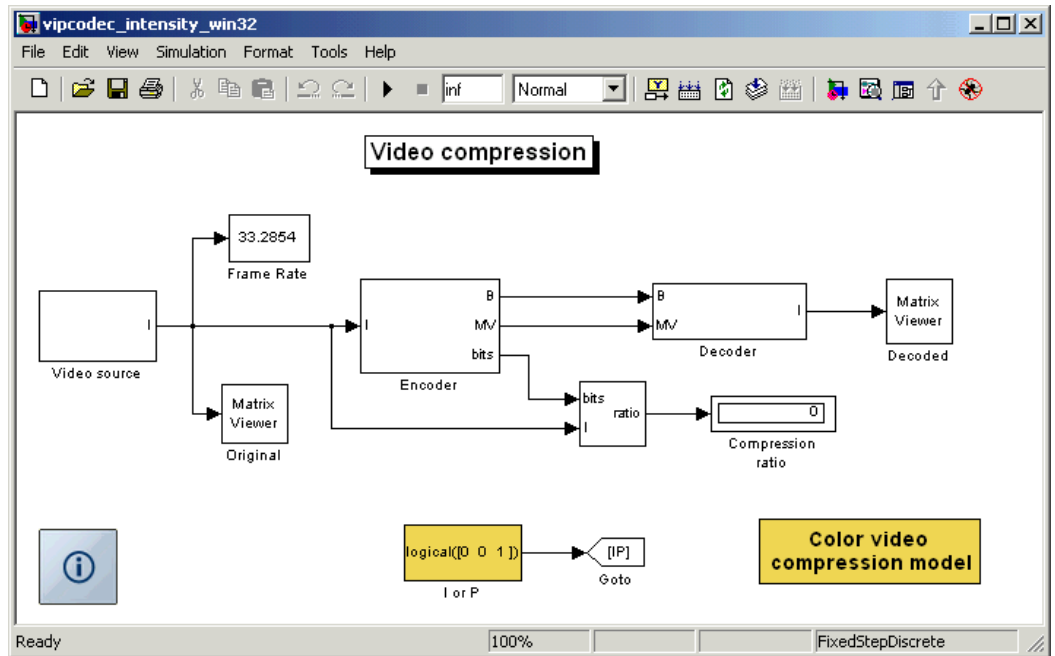
In this example, you used the 2-D Correlation, 2-D Maximum, and Draw Shapes blocks to track the motion of an object in a video stream. For more information about these blocks, see the 2-D Correlation, Maximum, and Draw Shapes block reference pages in the *Video and Image Processing Blockset Reference*.

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.

Motion Compensation

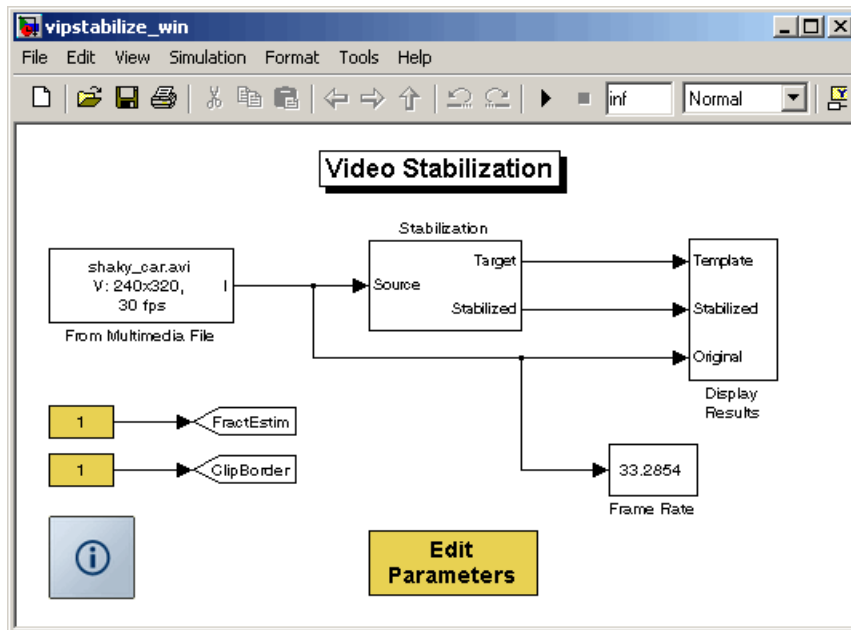
Motion compensation is a set of techniques that take advantage of redundancy in consecutive video frames. These techniques are used in video processing applications such as video compression and video stabilization. For both of these applications, motion compensation is a two-step process of detection and compensation. The detection step results in the specification of a motion vector that relates two consecutive video frames. For video compression, the compensation step involves using the motion vector to predict the current video frame from the previous frame and encoding the prediction residual. For video stabilization, the compensation step involves translating the current frame in the opposite direction of the motion vector to stabilize the video sequence.

The Video and Image Processing Blockset software contains a video compression demo model that you can open by typing `vipcodec` at the MATLAB command prompt.



This demo model detects motion by analyzing how much objects move between consecutive video frames. The model aligns two sequential video frames, subtracts them, and codes the residual.

The Video and Image Processing Blockset software also contains a video stabilization demo model that you can open by typing `vipstabilize` at the MATLAB command prompt.



The demo illustrates a motion stabilization technique based on the sum of absolute differences (SAD) method. It applies the SAD technique to remove unwanted translational camera motions and generate a stabilized video.

Image Compression

In this section...

“Overview of Image Compression” on page 8-12

“Compressing an Image” on page 8-12

“Viewing the Compressed Image” on page 8-18

Overview of Image Compression

The following sections use a two-part example to show how to build a Simulink model that is capable of image compression. In the first part of the example, the input image is divided into blocks and the two-dimensional DCT is computed for each block. The DCT coefficients are then quantized, coded, and transmitted. The receiver in the second part of the example decodes the quantized DCT coefficients, computes the inverse two-dimensional DCT of each block, and then puts the blocks back together into a single image. Although there is some loss of quality in the reconstructed image, it is recognizable as an approximation of the original image.

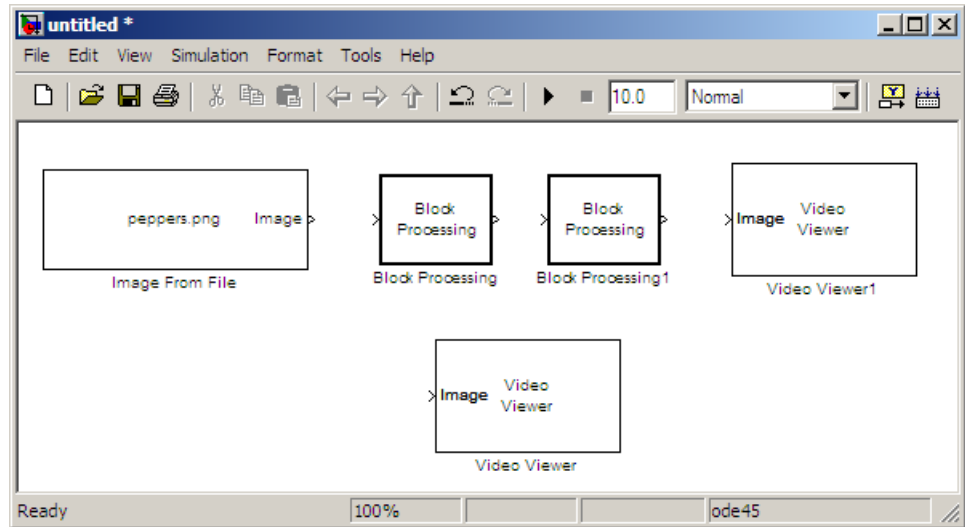
Compressing an Image

You can use image compression to reduce the size of an image before you transmit it. The compressed image retains many of the original image’s features but requires less bandwidth. In this topic, you use the 2-D DCT and Selector blocks to compress an intensity image:

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

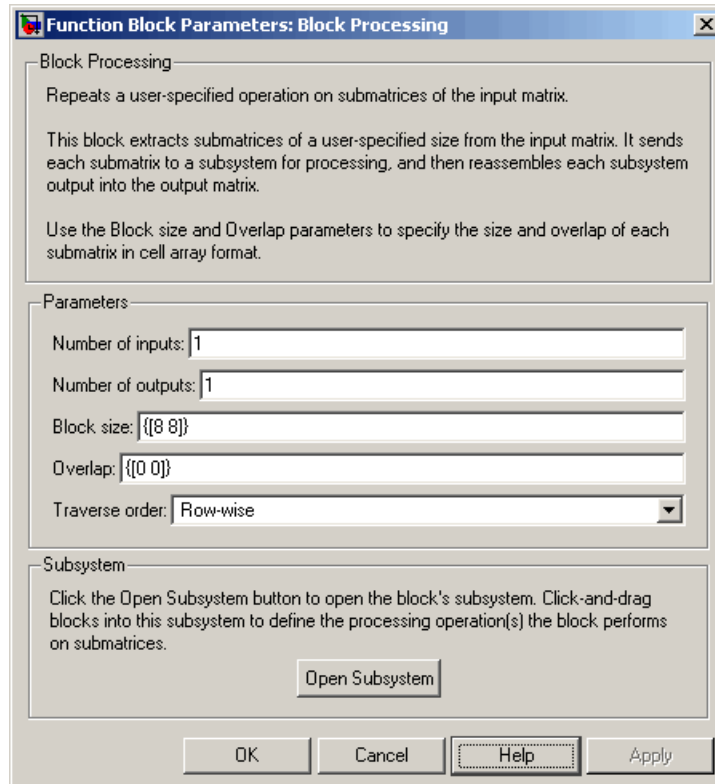
Block	Library	Quantity
Image From File	Video and Image Processing Blockset > Sources	1
Block Processing	Video and Image Processing Blockset > Utilities	2
Video Viewer	Video and Image Processing Blockset > Sinks	2

2 Position the blocks as shown in the following figure.

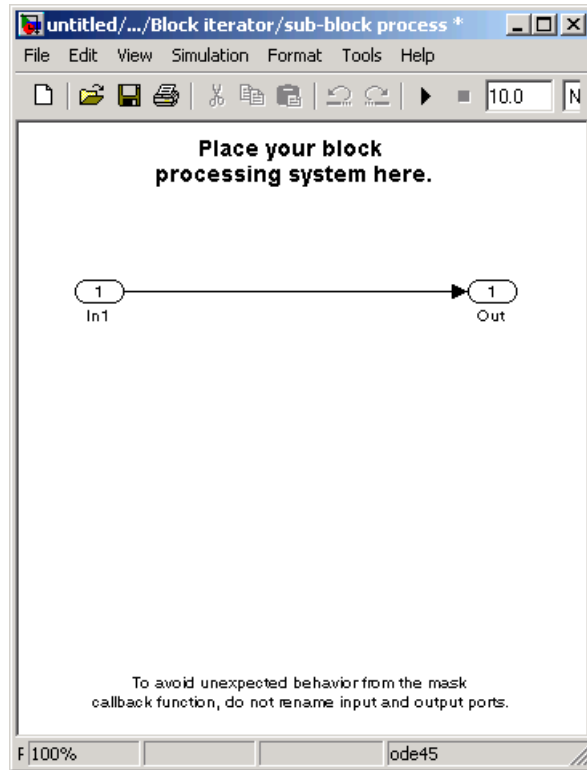


You are now ready to set your block parameters by double-clicking the blocks, modifying the block parameter values, and clicking **OK**.

- 3 Use the Image From File block to import the intensity image into your model. Set the block parameters as follows:
 - **Main** pane, **File name** = cameraman.tif
 - **Data Types** pane, **Output data type** = double
- 4 Use the Video Viewer block to view the original intensity image. This image is a 256-by-256 matrix of 8-bit unsigned integer values. Accept the default parameters.
- 5 The first Block Processing block represents the transmission portion of the block diagram. This block sends 8-by-8 submatrices of the original matrix to the block's subsystem for processing. Use this block when you want to perform block-based processing on large input images. To view the subsystem, double-click the block and click **Open Subsystem**.



The Block Processing block's subsystem opens.

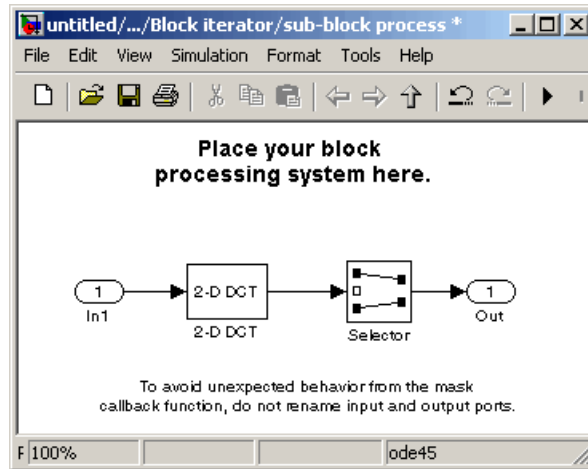


You can drag blocks into this subsystem to process the submatrices.

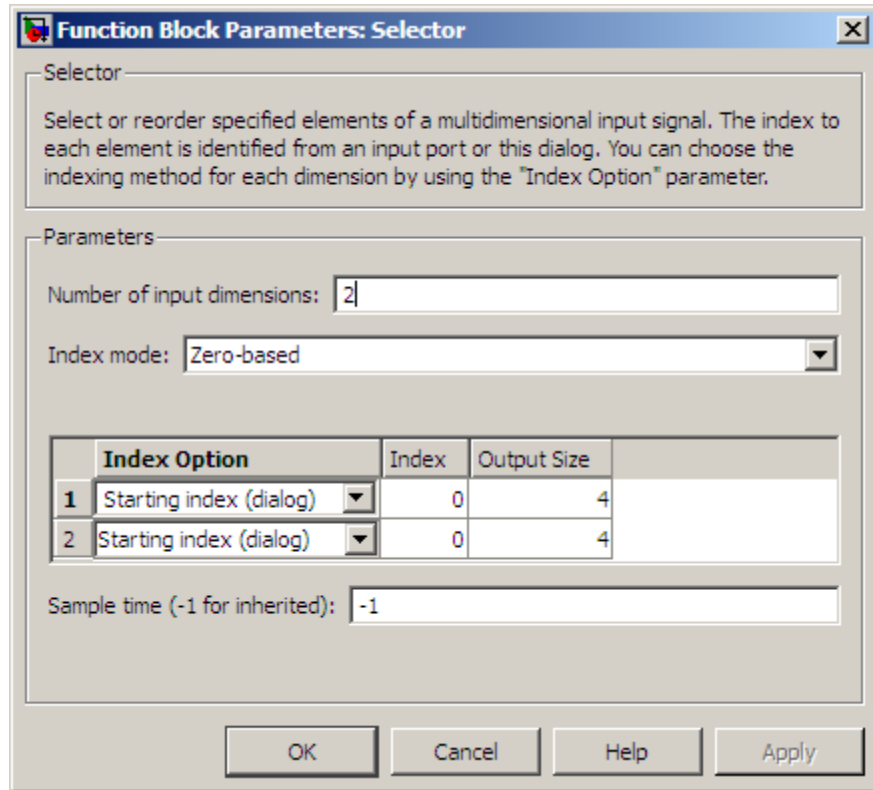
6 Add the following blocks to your subsystem.

Block	Library	Quantity
2-D DCT	Video and Image Processing Blockset > Transforms	1
Selector	Simulink > Signal Routing	1

7 Connect the blocks as shown in the following figure.



- 8 The 2-D DCT block takes the two-dimensional DCT of each submatrix. This process puts most of the energy in the image into the upper left corner of the resulting matrix. Use the default parameters.
- 9 Use the Selector block to extract the upper left corner of the submatrix. Set the block parameters as follows:
 - **Number of input dimensions** = 2
 - **Index mode** = Zero-based
 - 1
 - **Index Option** = Starting index (dialog)
 - **Index** = 0
 - **Output Size** = 4
 - 2
 - **Index Option** = Starting index (dialog)
 - **Index** = 0
 - **Output Size** = 4



You are using the Selector block to compress the image by extracting the upper left corner of the submatrix, which contains the high energy image coefficients. You want to transmit only this portion of the submatrix because it requires less bandwidth than transmitting the entire submatrix.

10 Close the subsystem and the Block Processing dialog box.

You have now configured the Block Processing and 2-D DCT blocks to compress an image for transmission. In "Viewing the Compressed Image" on page 8-18, you use the 2-D IDCT block to transform the image back to the time domain. Then, you view the compressed image.

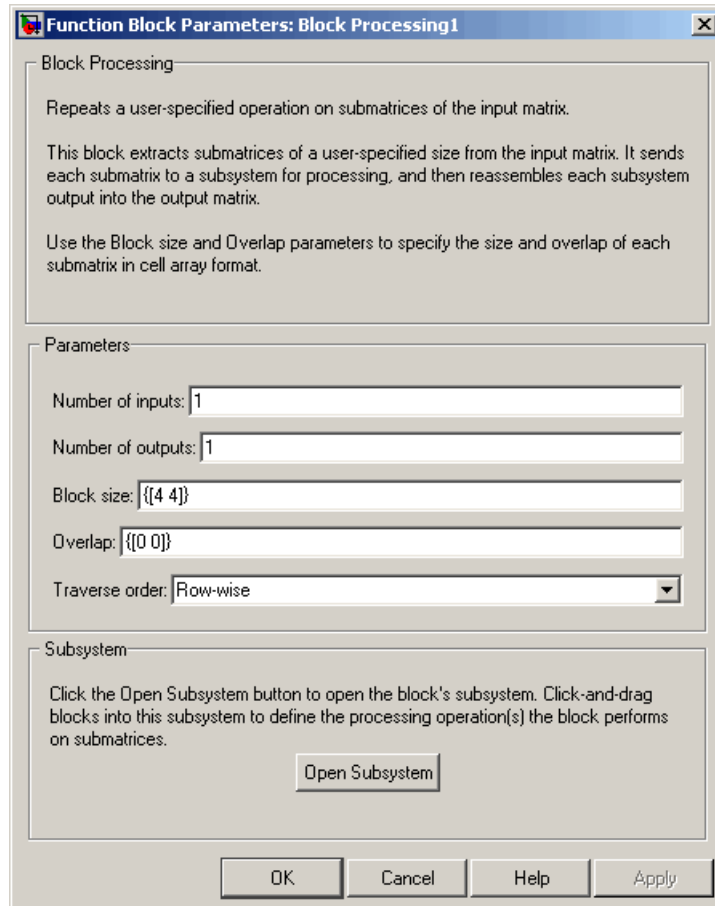
Viewing the Compressed Image

In “Compressing an Image” on page 8-12, you compressed an image using the 2-D DCT and Selector blocks. Now, you can use the 2-D IDCT block to transform the image back to the time domain and view the result:

- 1** If you have not already done so, set the Image From File block to import the intensity image into your model. Set the block parameters as follows:
 - **Main** pane, **File name** = cameraman.tif
 - **Data Types** pane, **Output data type** = double
- 2** If the model you created in “Compressing an Image” on page 8-12 is not open on your desktop, you can open an equivalent model by typing

```
doc_compressing_an_image
```

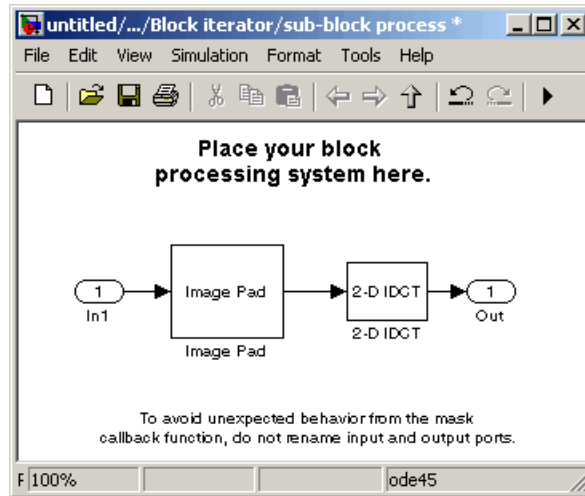
at the MATLAB command prompt.
- 3** Use the Block Processing1 block to set the size of the submatrices that the block passes to the subsystem. Set the **Block size** parameter to {[4 4]}.



- 4** Open the block's subsystem by clicking **Open Subsystem**, and add the following blocks to it.

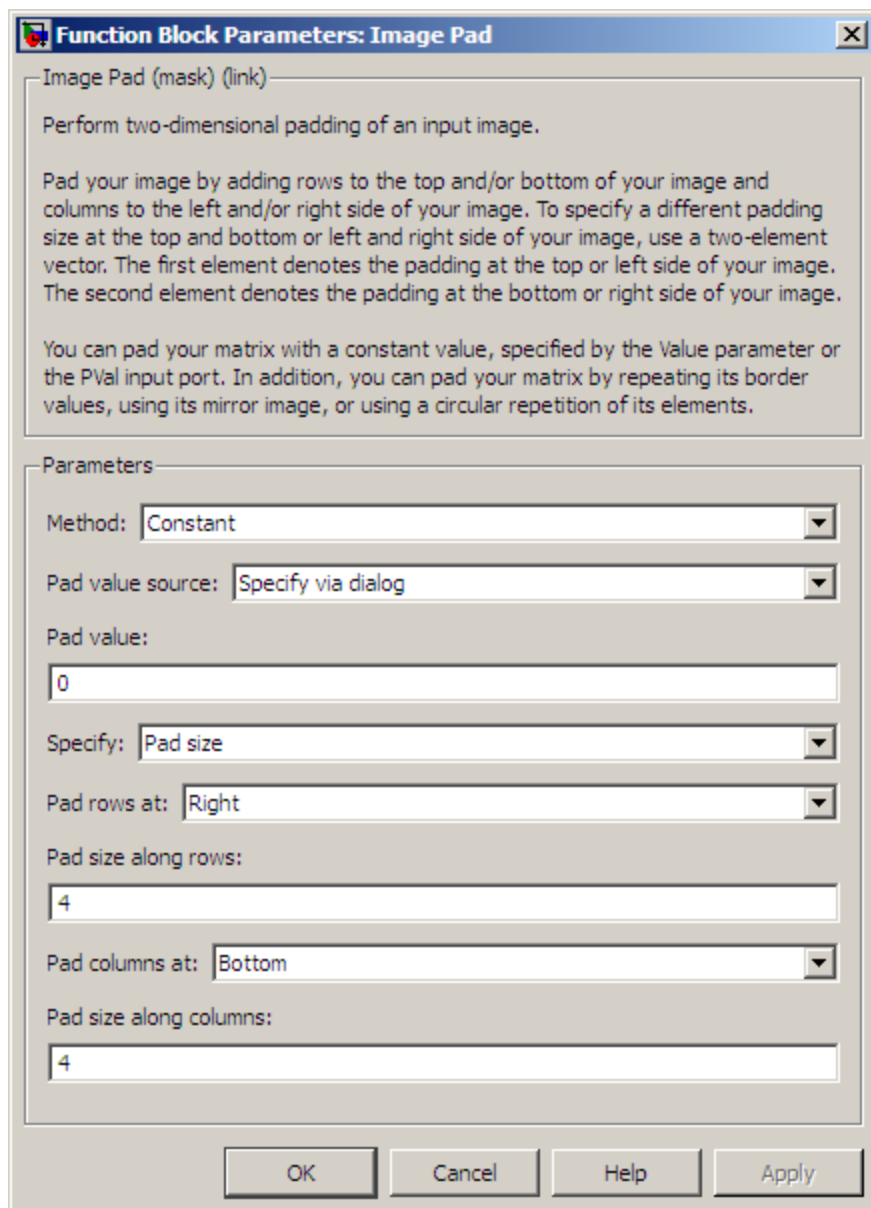
Block	Library	Quantity
Image Pad	Video and Image Processing Blockset > Utilities	1
2-D IDCT	Video and Image Processing Blockset > Transforms	1

5 Connect the blocks as shown in the following figure.



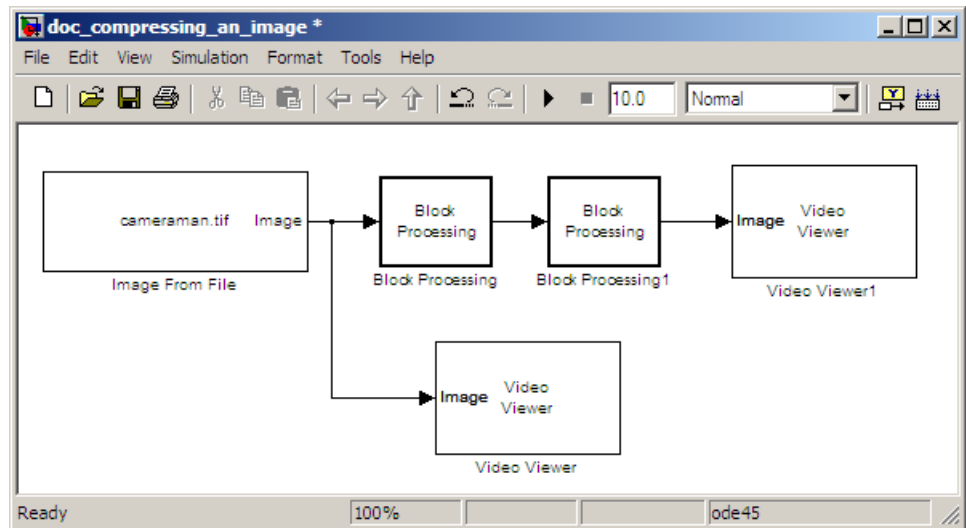
6 Use the Image Pad block to zero pad the 4-by-4 submatrix back to its original 8-by-8 size. Set the block parameters as follows:

- Pad rows at = Right
- Pad size along rows = 4
- Pad columns at = Bottom
- Pad size along columns = 4



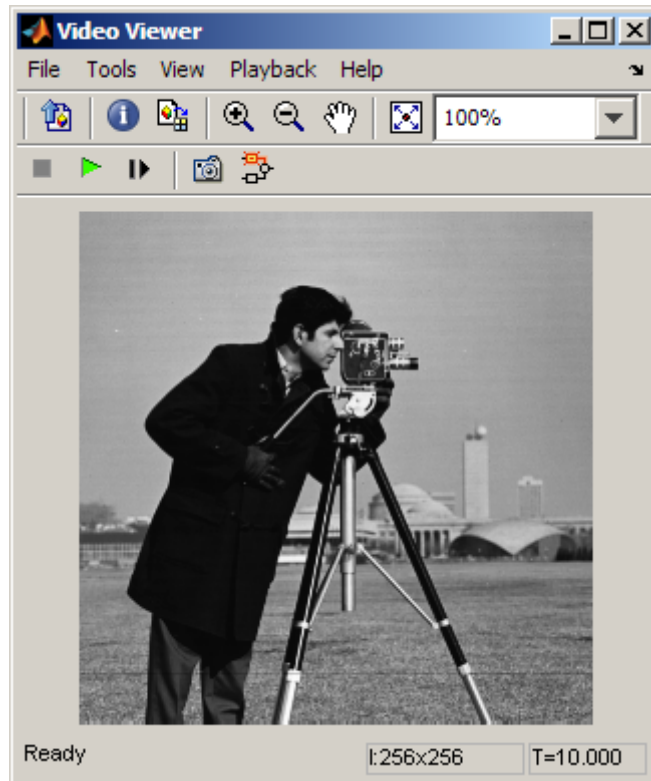
Because zeros are replacing the low energy transform coefficients, the output image is an approximation of the original image.

- 7 The 2-D IDCT block takes the inverse two-dimensional DCT of the submatrices. Accept the default parameters.
- 8 Close the subsystem and the Block Processing1 dialog box.
- 9 Use the Video Viewer1 block to view the compressed image. Accept the default parameters.
- 10 Connect the blocks as shown in the following figure.

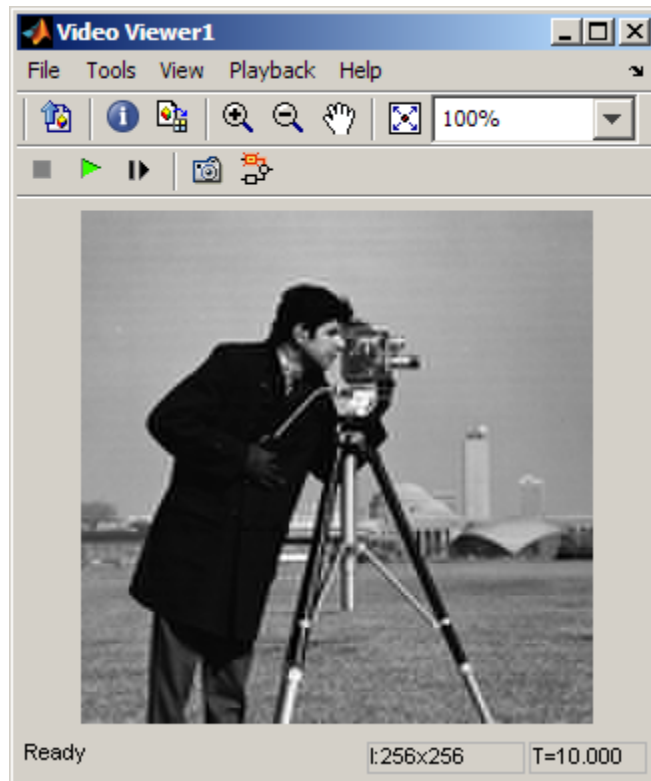


- 11 Set the configuration parameters. Open the Configuration dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Set the parameters as follows:
 - Solver pane, **Stop time** = 0
 - Solver pane, **Type** = Fixed-step
 - Solver pane, **Solver** = Discrete (no continuous states)
- 12 Run the model.

The original image is displayed in the Video Viewer window.



The compressed image is displayed in the Video Viewer1 window. The compressed image is not as clear as the original image. However, it still contains many of its features.



In this example, you used the 2-D DCT, Image Pad 2-D IDCT, and Block Processing blocks to compress an image. For more information on these blocks, see the 2-D DCT, Image Pad, 2-D IDCT, and Block Processing block reference pages in the *Video and Image Processing Blockset Reference*. For information on the Selector block, see the Simulink documentation. For more information on sharpening an image, see “Sharpening and Blurring an Image” on page 4-30.

Getting Started with System Objects

- “What Are System Objects?” on page 9-2
- “Setting Up and Running System Objects” on page 9-3
- “Using System Objects with the Embedded MATLAB Subset” on page 9-9

What Are System Objects?

System objects are MATLAB object-oriented implementations of algorithms. They extend MATLAB by enabling you to model dynamic systems represented by time-varying algorithms. System objects are well integrated into the MATLAB language, regardless of whether you are writing simple functions, working interactively in the command window, or creating large applications.

In contrast to MATLAB functions, System objects automatically manage state information, data indexing, and buffering, which is particularly useful for iterative or stream data processing. This enables efficient processing of long data sets. System objects support fixed-point arithmetic and C-code generation from MATLAB and Simulink. With System objects, you can optionally generate code to target the desktop or external hardware. System objects are part of the Embedded MATLAB[®] subset, and therefore can be used in Simulink[®] models via the Embedded MATLAB function block.

Setting Up and Running System Objects

In this section...

“Creating an Instance of a System Object” on page 9-3

“Using Methods to Run System Objects ” on page 9-6

“Finding Help and Demos for System Objects” on page 9-8

Creating an Instance of a System Object

You must create an instance of a System object before using it. You can create the object at the MATLAB command line or within a program file. The general syntax for creating an instance of a System object with default property values is:

```
<handleName> = <packagename>.<ObjectName>
```

where:

- `handleName` is a MATLAB variable that holds the handle pointing to the created object. System objects are handle objects and follow handle semantics (e.g., when you call a method using the handle, it affects the original object, not a copy of that object). See “The Handle Superclass” for information on handle objects. See “Value or Handle Class — Which to Use” in the MATLAB user documentation for information on object handles.
- `packagename` is the package that contains the particular object. *Packages* are libraries of System objects. For example, these packages implement object versions of associated algorithms.
 - `signalblks` – a package that corresponds to the Signal Processing Blockset
 - `video` – a package that corresponds to the Video and Image Processing Blockset
- `ObjectName` is the particular object in the package.

This example creates a digital filter object, with default property values, from the `signalblks` package:

```
h = signalblks.DigitalFilter
```

Your command-line code and programs can pass MATLAB variables into and out of System objects.

Understanding System Object Modes

System objects are in one of two modes: *unlocked* or *locked*. After you create an instance of an object and until it starts processing data, that object is in unlocked mode. You can change any of its properties as desired.

When the object begins processing data, it initializes and is locked. When the object is locked, you cannot change the number of inputs or outputs or the value of any nontunable property. You also cannot change the input data type, dimensions, or complexity of any tunable or nontunable property. These restrictions allow the object to maintain states and allocate memory appropriately. The typical way in which an object becomes locked is when the `step` method is called on that object. See “Changing Properties While Running System Objects” on page 9-5 for information on tunable and nontunable properties.

Viewing System Object Properties

System objects have properties that configure the object. You use the default values or set each property to a specific value. The combination of a property and its value is referred to as a *property-value pair*. You can display the list of relevant properties and their current values for an object by using the object handle only, `<handleName>`. Some properties are relevant only when you set another property or properties to particular values. If a property is not relevant, it does not display.

To display a particular property value, use the handle of the created object followed by the property name: `<handleName>.<propertyName>`.

The following code gets the `TransferFunction` property value for the previously created `DigitalFilter` object:

```
h.TransferFunction
```

Setting System Object Property Values

You set the property values of a System object to model the desired algorithm. To set a property when you first create the object, use property-value pair syntax. For properties that allow a specific set of string values, you can use tab completion to select from a list of valid values.

```
<handleName> = <packageName>.<objectName>(property1,value1,...
    property2,value2...)

h1 = signalblks.DigitalFilter('TransferFunction','FIR (all zeros)')
```

To set a property after you create an object, use either of the following syntaxes:

```
<handleName>.<propertyName> = <propertyValue>
```

```
h1.TransferFunction = 'FIR (all zeros)'
```

or

```
set(<handleName>,<propertyName>,<propertyValue>)
```

```
set(h1,'TransferFunction','FIR (all zeros)')
```

You can enter property-value pairs in any order, except if you are using value-only inputs. Some object properties have no useful default values or are specified every time you create an instance of an object. For these properties, you can specify only the value without specifying the corresponding property name.

If you use value-only inputs, those inputs must be in a specific order. Refer to the object reference page for details. For example, `h2 = signalblks.FIRDecimator(3,fir1(20,0.5))` specifies the decimation factor as 3 and the numerator as `fir1(20,0.5)`.

Changing Properties While Running System Objects

When an object is in locked mode, it is processing data and you can only change the values of properties that are *tunable*. To determine if a particular System object property is tunable, see the corresponding reference page or use `help.<packageName>.<objectName>.<propertyName>`. For information

on locked and unlocked modes, see “Understanding System Object Modes” on page 9-4.

For most objects, if you change a nontunable property while the object is in locked mode, the object unlocks, loses its state information, and stops processing. For a locked object, if you change the data type, dimensions or complexity of an input or tunable property, the object reinitializes the next time you call the step method. See the object’s reference page for more information.

Using Methods to Run System Objects

After you create a System object, you use various object methods to obtain information from the object or have it process data. As an extension of MATLAB classes, most System objects include a standardized set of methods. Some of these methods only apply to particular objects (see “Common Methods” on page 9-7). All methods that are applicable to an object are described in the reference pages for that object.

System object method names begin with a lowercase letter and class and property names begin with an uppercase letter. The syntax for using methods is `<methodName> (<handleName>)`.

Understanding the Advantages of Using Methods

System objects use two commands to process data—a constructor to create the object and a method to run data through the object. This separation of declaration from execution lets you create multiple, persistent, reusable instances of an object, each with different settings. Using this approach avoids repeated input validation and verification, allows for easy use within a programming loop, and improves overall performance. MATLAB functions must validate parameters every time you call the function.

These advantages make System objects particularly well suited for processing streaming data, where segments of a continuous data stream are processed iteratively. This ability to process streaming data provides the advantage of not having to hold large amounts of data in memory. Use of streaming data also allows you to use simplified programs that use loops efficiently.

Common Methods

System objects support the following methods, each of which is described in a method reference page associated with the object. In cases where a method is not applicable to a particular object, calling that method has no effect on the object.

- **step** – Initializes needed resources, processes inputs to the object based on the current object states and properties, returns outputs, and updates the object states. After you call the **step** method, you cannot change any input specifications (i.e., dimensions, data type, complexity). During execution, you can change only tunable properties. The **step** method returns regular MATLAB variables.

Note For System objects that perform calculations on data (and not just rearrange data), if you pass non-floating point data into that object, it outputs a fixed-point numeric `fi` object, even if you specify the output data type to be the same as the input data type.

When you create a source object, you specify whether **step** processes the data as samples or as frames. If the object is a source, **step** produces outputs but has no inputs. If the object is a sink, **step** requires inputs, but produces no outputs. For all other objects, **step** has both inputs and outputs, `[out1,...,outN] = step(h,in1,...,inM)`.

- **reset** – Resets the internal states of the object to the initial values for that object.
- **getNumInputs** – Returns the number of inputs expected by the **step** method. This number varies for an object depending on whether any properties enable additional inputs.
- **getNumOutputs** – Returns the number of outputs from the **step** method. This number varies for an object depending on whether any properties enable additional outputs.
- **isDone** – Applies only to source objects with end-of-data capability. **isDone** returns logical `true` when the most recent **step** call reaches the end-of-data state. This method returns `false` if either end-of-data is not reached or if the source object does not have end-of-data capability.

- `close` – Applies to sink and source objects only. `close` releases any special resources allocated by the object, such as file handles and device drivers.

Finding Help and Demos for System Objects

Refer to the following resources for more information about System objects:

- `help <packagename>` – Lists all System objects in the package, organized by category.
- `help <packagename>.<ObjectName>` — Displays help for the object.
- `doc <packagename>.<ObjectName>` – Displays the reference page for the object, including its properties.
- `help <packageName>.<ObjectName>.<PropertyName>` – Displays help for the property
- `help <packageName>.<ObjectName>.helpFixedPoint` – Displays a list of fixed-point properties for the object.
- `help <packagename>.<ObjectName>.<methodName>` – Displays the method reference page for the object.
- Demos – Provides System object related demos. To view demos, go to online Help contents for the associated product blockset. Under Demos, select MATLAB demos.
- *Object-Oriented Programming* in the MATLAB user documentation – Provides general information about working with objects.

Using System Objects with the Embedded MATLAB Subset

In this section...

“Considerations for Using System Objects with the Embedded MATLAB Subset” on page 9-9

“Using System Objects with Embedded MATLAB Coder” on page 9-11

“Using System Objects with the Embedded MATLAB Function Block” on page 9-12

“Using System Objects with Embedded MATLAB MEX” on page 9-12

Considerations for Using System Objects with the Embedded MATLAB Subset

You can use System objects in code generated using the Embedded MATLAB subset, which is part of the MATLAB language. To generate code, you must also have Simulink and Real-Time Workshop products. Embedded MATLAB lets you generate efficient code for deployment in embedded systems. It also accelerates fixed-point algorithms. System objects support code generation using the Embedded MATLAB function block in Simulink and using the Embedded MATLAB coder function.

For general information on using Embedded MATLAB, see

- Working with the Embedded MATLAB Subset.
- *Embedded MATLAB Getting Started Guide*.
- *Real-Time Workshop Embedded Coder Getting Started Guide*.

You can customize your generated code by using a configuration object, which is described in “Configuring Your Environment for Code Generation”.

The following example, which uses System objects, shows the key factors to consider when you write MATLAB code to be generated using Embedded MATLAB.

```
function lmssystemidentification
% LMSSYSTEMIDENTIFICATION System identification using
```

```
% LMS adaptive filter
%#eml

% Declare System objects as persistent to generate code
% using Embedded MATLAB.

persistent hlms hfilt;

% Initialize persistent System objects in Embedded MATLAB
% only once. Do this with 'if isempty(persistent variable).'
% This condition will be false after the first time.

if isempty(hlms)

    % Create LMS adaptive filter used for system
    % identification. Pass property value arguments
    % as constructor arguments. Property values must
    % be constants during compile time.

    hlms = signalblks.LMSFilter(11, 'StepSize', 0.01);

    % Create system (an FIR filter) to be identified.

    hfilt = signalblks.DigitalFilter(...
        'TransferFunction', 'FIR (all zeros)', ...
        'Numerator', fir1(10, .25));
end

x = randn(1000,1); % Input signal
d = step(hfilt, x) + 0.01*randn(1000,1); % Desired signal
[~,~,w] = step(hlms, x, d); % Filter weights

% Declare functions called into MATLAB that do not generate
% code as extrinsic.

eml.extrinsic('stem');

stem([get(hfilt, 'Numerator').', w]);
end
```



```
% To compile this function use emlc lmssystemidentification.  
% This produces a mex file with the same name in the current  
% directory.
```

Review the following considerations when you create code that includes System objects for use with the Embedded MATLAB subset .

- Assign System objects to persistent variables.
- Initialize System objects once by embedding the object handles in an if statement with a call to `isempty()`.
- Call the constructor exactly once for any instance of a System object.
- Arguments to System object constructors must be compile-time constants.
- Use the object constructor to set System object properties because Embedded MATLAB does not allow you to use dot notation. Do not set any properties during code generation. You can use `get` to display properties.
- Set System object properties using parameter-value pairs only. Do not use value-only inputs.
- Ensure that input to a System object is consistent with the object size, type, and complexity.
- Do not set System objects to become outputs from the Embedded MATLAB function block or from a MEX function generated by Embedded MATLAB.
- Do not pass a System object as an example input argument to a function being compiled with Embedded MATLAB Coder.
- Do not pass a System object from within Embedded MATLAB to functions declared as extrinsic (i.e., functions called in interpreted mode) using `eml.extrinsic`. Do not return System objects from any extrinsic functions.

Using System Objects with Embedded MATLAB Coder

Embedded MATLAB Coder (`emlc`) is a Real-Time Workshop function that converts MATLAB code into C-code. You can include System objects in a MATLAB program in the same way you include any other program elements. For more information on Embedded MATLAB Coder, see “Converting MATLAB Code to C/C++ Code” and “Generating C Code Using `emlc`”.

Using System Objects with the Embedded MATLAB Function Block

Using the Embedded MATLAB Function block, you can include a MATLAB language function in a Simulink model. This model can then generate embeddable code using the Embedded MATLAB subset. You can include any System object in the Embedded MATLAB function block. System objects provide higher level algorithms for code generation than do most associated blockset blocks. For more information about the Embedded MATLAB Function block, see [Using the Embedded MATLAB Function Block](#) and the [Embedded MATLAB Function block reference page](#) in the Simulink documentation.

Using System Objects with Embedded MATLAB MEX

You can use System objects with Embedded MATLAB MEX (`emlmex`), which is particularly useful if you are using System objects that include fixed-point support. `emlmex` converts MATLAB code to C-MEX code which is optimized specifically to accelerate fixed-point algorithms to compiled C-code speed. For more information, see “Working with Embedded MATLAB MEX” in the [Embedded MATLAB language subset documentation](#).

Using Video and Image Processing System Objects

- “What Are Video and Image Processing System Objects?” on page 10-2
- “Generating Code for Video and Image Processing System Objects” on page 10-3
- “Working with Fixed-Point Data” on page 10-5
- “Example: Using System Objects in Video and Image Processing Applications: Marking a Region of Interest” on page 10-9

What Are Video and Image Processing System Objects?

Video and image processing System objects are object-oriented implementations of video and image processing algorithms. This set of System objects is organized in a single package, `video`. Many of these objects correspond to block algorithms in the Video and Image Processing Blockset. A key difference between blocks and System objects is that you include blocks in Simulink models whereas you include System objects in programs or MATLAB command-line code.

Video and image processing System objects provide these advantages.

- Support for code generation of algorithms in MATLAB (see “Generating Code for Video and Image Processing System Objects” on page 10-3).
- Additional support for fixed-point-capable algorithms in MATLAB (see “Working with Fixed-Point Data” on page 10-5).

Information in the following topics applies to System objects. These topics refer to models and blocks, but the information and concepts apply to System objects, too.

- Chapter 2, “Importing and Exporting Images and Video” – batch processing, live video, and multimedia files
- Chapter 3, “Viewing Video” – video files and video file frames
- Chapter 8, “Example Applications” — pattern matching, motion compression, and image compression

Generating Code for Video and Image Processing System Objects

These video and image processing System objects support code generation in MATLAB via Embedded MATLAB Coder (emlc), which requires Simulink and Real-Time Workshop. See “Using System Objects with the Embedded MATLAB Subset” on page 9-9 for information on generating code.

Video and Image Processing Code Generation Support

```
video.AlphaBlender
video.Autocorrelator2D
video.Autothresher
video.BlobAnalysis
video.ChromaResampler
video.ColorSpaceConverter
video.ConnectedComponentLabeler
video.Convolver2D
video.Crosscorrelator2D
video.DCT2D
video.Deinterlacer
video.DemosaicInterpolator
video.EdgeDetector
video.GammaCorrector
video.GeometricRotator
video.GeometricTranslator
video.Histogram2D
video.HistogramEqualizer
video.HoughTransform
video.IDCT2D
video.ImageComplementer
video.ImageDataTypeConverter
video.LocalMaximaFinder
video.MarkerInserter
video.Maximum
video.Mean
video.Median
```

```
video.Minimum  
video.MorphologicalClose  
video.MorphologicalDilate  
video.MorphologicalErode  
video.MorphologicalOpen  
video.PSNR  
video.Pyramid  
video.ShapeInserter  
video.StandardDeviation  
video.TraceBoundaries  
video.Variance
```

Working with Fixed-Point Data

Working with Fixed-Point Data

- “Getting Information About Fixed-Point System Objects” on page 10-5
- “Displaying Fixed-Point Properties” on page 10-6
- “Setting System Object Fixed-Point Properties” on page 10-7

Getting Information About Fixed-Point System Objects

General information about using fixed-point data processing is in “Working with Fixed-Point Data” in the Signal Processing Blockset documentation. System objects that support fixed-point data processing have fixed-point properties, which you can display for a particular object by typing `video.<ObjectName>.helpFixedPoint` at the command line.

See “Displaying Fixed-Point Properties” on page 10-6 to set the display of System object fixed-point properties.

The following video and image processing System objects support fixed-point data processing.

Video and Image Processing Fixed-Point Data Processing Support

```
video.AlphaBlender
video.Autocorrelator2D
video.Autothresher
video.BlobAnalysis
video.BlockMatcher
video.ContrastAdjuster
video.Convolver2D
video.CornerDetector
video.Crosscorrelator2D
video.DCT2D
video.Deinterlacer
video.DemosaicInterpolator
video.EdgeDetector
video.FFT2D
```

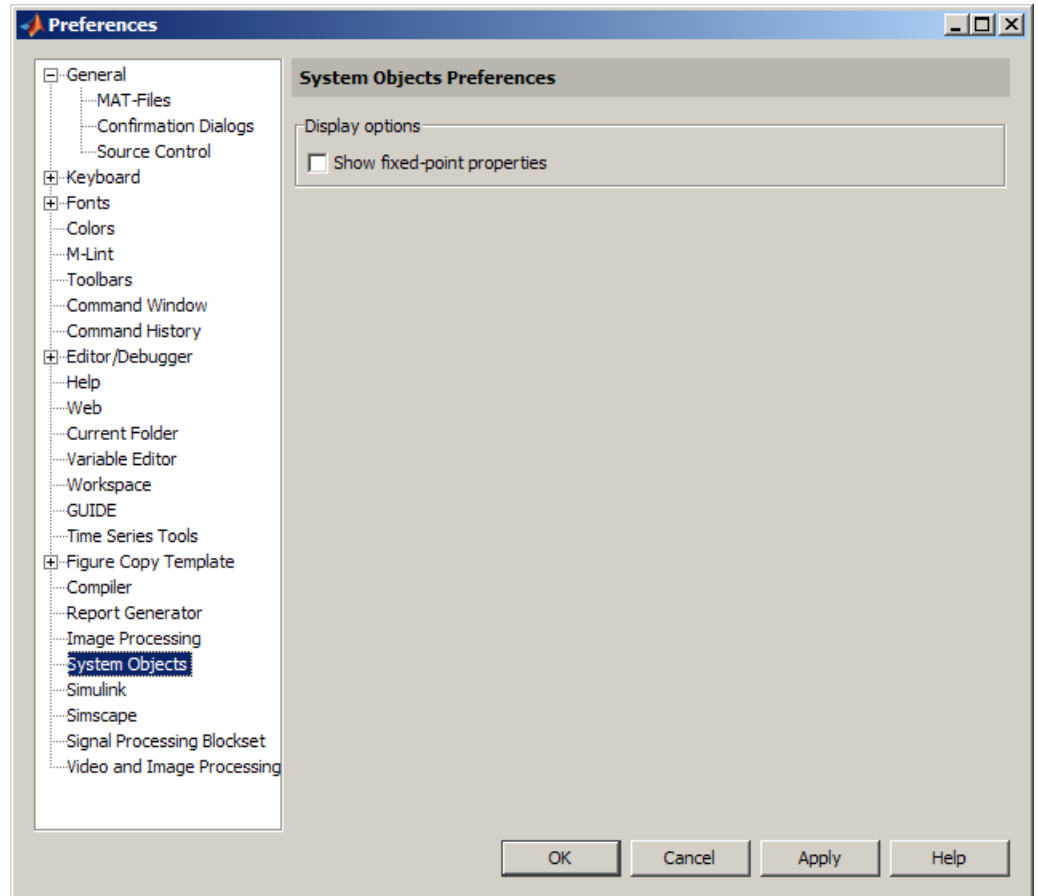
```
video.GeometricRotator
video.GeometricScaler
video.GeometricTranslator
video.Histogram2D
video.HoughLines
video.HoughTransform
video.IDCT2D
video.IFFT2D
video.ImageDataTypeConverter
video.ImageFilter
video.MarkerInserter
video.Maximum
video.Mean
video.Median
video.MedianFilter2D
video.Minimum
video.OpticalFlow
video.PSNR
video.Pyramid
video.SAD
video.ShapeInserter
video.Variance
```

Displaying Fixed-Point Properties

You can control whether the software displays fixed-point properties with either of the following commands:

- `matlab.system.ShowFixedPointProperties`
- `matlab.system.HideFixedPointProperties`

at the MATLAB command line. These commands set the **Show fixed-point properties** display option. You can also set the display option directly via the MATLAB preferences dialog box. Select **File > Preferences** on the MATLAB desktop, and then select **System Objects**. Finally, select or deselect **Show fixed-point properties**.



If an object supports fixed-point data processing, its fixed-point properties are active regardless of whether they are displayed or not.

Setting System Object Fixed-Point Properties

A number of properties affect the fixed-point data processing used by a System object. Objects perform fixed-point processing and use the current fixed-point property settings when they receive fixed-point input.

You change the values of fixed-point properties in the same way as you change any System object property value. See “Setting System Object Property

Values” on page 9-5. You also use the Fixed-Point Toolbox™ `numericType` object to specify the desired data type as fixed-point, the signedness, and the word- and fraction-lengths.

In the same way as for blocks, the data type properties of many System objects can set the appropriate word lengths and scalings automatically by inheriting via the internal rule (see “Inherit via Internal Rule” for how this rule applies to different types of blocks and objects).

In most cases, if you have not set the property that activates a dependent property and you attempt to change that dependent property, a warning message displays. As a convenience, if you set a dependent, fixed-point, `Custom<xxx>DataType` property before setting the `<xxx>DataType` property, the System object automatically sets `<xxx>DataType` for you to activate the dependent property. `<xxx>` differs for each object. For example, for the `video.EdgeDetector` object, setting `CustomProductDataType` to `numericType(1,16,15)` automatically sets `ProductDataType` to `'Custom'`.

Example: Using System Objects in Video and Image Processing Applications: Marking a Region of Interest

The following example shows you how to define a region of interest (ROI) using System objects to draw a green rectangle on a video stream.

- 1 Create a `MultimediaFileReader` System object and, set its properties.

This System object enables you to work with video data from an AVI file.

At the MATLAB command line, type the following code:

```
hVideoIn = video.MultimediaFileReader
```

The `MultimediaFileReader` System Component exists in both Signal Processing Blockset and Video and Image Processing Blockset software. The difference between them, are the default settings for video or audio file formats.

- 2 Create a `ShapeInserter` System object to draw a green rectangle on the video.

At the MATLAB command line, type the following code:

```
hRect = video.ShapeInserter;  
hRect.BorderColor = 'Custom';  
hRect.CustomBorderColor = [0 1 0];
```

- 3 Create a `DeployableVideoPlayer` System object so you can see the result of your processing. In this example, the System object sends the video to your default video device and places the video window near the center of your screen.

At the MATLAB command line, type the following code:

```
hVideoOut = video.DeployableVideoPlayer;  
hVideoOut.WindowLocation = [400 400];
```

Note The `DeployableVideoPlayer` System object is only supported on Windows platforms.

After you create all your System objects, use them to annotate your video data by calling them within a processing loop. The following code is an example of a processing loop:

```
pts = [10 5 80 90];

while ~isDone(hVideoIn)
    % Get video data from the file, and store the data
    % in the variable videoData.
    videoData = step(hVideoIn);

    % Add the rectangle to the video, and send it to the display.
    imageRect = step(hRect, videoData, pts);
    step(hVideoOut, imageRect);
end

% Close the file, free memory and release the video device
close(hVideoIn);
close(hVideoOut);
```

The `step` method takes the input to each System object and computes an output. It also updates the System object states. In the last two lines, the `close` method releases the file handles and the hardware connections.

The Deployable Video Player window displays the resulting video. The last frame of the video appears in the following figure.



After you set up the basic example, you can try either of the following tasks:

- Changing the tunable parameters
- Making the rectangle move around the video display

A

- Accelerator mode 1-28
- adding periodic noise to a signal 4-45
- adjusting
 - intensity image contrast 4-54
 - RGB image contrast 4-61
- Adobe Acrobat Reader 1-12
- algorithms
 - bicubic interpolation 6-4
 - bilinear interpolation 6-3
 - nearest neighbor interpolation 6-2
- angles
 - rotation 6-6
- annotating
 - AVI files 2-22
- arrays
 - interpretation of 1-15
- artifacts
 - in an image 4-45
- audio
 - exporting to multimedia file 2-38
- Autothreshold block
 - to perform thresholding 5-7
- AVI files
 - annotating 2-22
 - cropping 2-30
 - exporting 2-18
 - importing 2-14
 - saving to multiple files 2-30
 - splitting 2-30
 - viewing 2-14

B

- background
 - estimation 7-10
 - pixels 4-2
 - user's expected 1-12
- batch processing 2-2
- bicubic interpolation 6-4

- bilinear interpolation 6-3
- binary
 - conversion from intensity 5-2
 - images 1-15
- blurring images 4-30
- Boolean matrices 1-15
- boundaries
 - of objects 4-2
- boundary artifacts 4-45
- brightening images 7-10

C

- capabilities of
 - Video and Image Processing Blockset software 1-2
- changing
 - image size 6-14
 - intensity image contrast 4-54
 - RGB image contrast 4-61
- chapter descriptions 1-13
- chroma components
 - of images 5-19
- chroma resampling 5-19
- chrominance resampling 5-19
- close method 9-8
- code generation
 - video and image processing objects 10-3
- codecs
 - supported by Microsoft Windows Media Player 2-8
- color
 - definition of 1-16
- color space conversion 5-14
- colormaps 1-16
- column-major format 1-26
- compensation
 - for motion 8-10
- compression
 - of images 8-12

- of video 8-10
- concepts
 - description of 1-15
- Configuration dialog box 1-26
- continuous rotation 6-6
- contrast
 - increasing 2-11
- controlling video duration 1-27
- conventions
 - column-major format 1-26
- conversion
 - color space 5-14
 - intensity to binary 5-2
 - R'G'B' to intensity 5-14
- correction
 - of uneven lighting 7-10
- correlation
 - used in object tracking 8-2
- counting objects 7-3
- cropping
 - AVI files 2-30
 - images 6-20
- D**
- data type support 1-33
- data types 1-16
- definition of
 - intensity and color 1-16
- demos
 - in the Help browser 1-5
 - on MATLAB Central 1-10
 - on the Web 1-9
 - Periodic noise reduction 4-45
 - Video compression 8-10
 - Video stabilization 8-10
- dependencies
 - on Windows dynamic libraries 1-31
- detection of
 - edges 4-2

- lines 4-9
- dilation 7-2
- DirectX 2-8
- dlls
 - dependencies on 1-31
- documentation
 - on the Web 1-11
 - on your system 1-11
 - PDF 1-12
 - printing 1-12
 - viewing 1-11
- downsampling
 - chroma components 5-19
- DVD installation 1-3
- dynamic range 1-16

E

- edge
 - pixels 4-2
 - thinning 4-2
- edge detection 4-2
- electrical interference 4-45
- erosion 7-2
- estimation
 - of image background 7-10
- executables
 - running 1-31
- exporting
 - AVI files 2-18
 - multimedia files 2-11

F

- feature extraction
 - finding angles between lines 4-18
 - finding edges 4-2
 - finding lines 4-9
- filtering
 - median 4-39

- operations 7-2
- finding
 - angles between lines 4-18
 - edges of objects 4-2
 - histograms of images 4-73
 - lines in images 4-9
- fixed point
 - System object preferences 10-6
 - System object processing 10-5
- fixed point properties
 - System objects 10-7
- form of objects 7-2
- frequency distribution
 - of elements in an image 4-73
- fspecial function 4-30

G

- gamma correction 5-14
- geometric transformation 6-1
- getNumInputs method 9-7
- getNumOutputs method 9-7
- gradient components
 - of images 4-2

H

- Help browser
 - demos 1-5
 - documentation 1-11
- histograms
 - of images 4-73

I

- image compression 8-12
- image credits 1-34
- image data
 - storage of 1-26
- image rotation 6-6
- image sequence processing 2-2

- image types 1-15
- images
 - binary 1-15
 - boundary artifacts 4-45
 - brightening 7-10
 - correcting for uneven lighting 7-10
 - counting objects in 7-3
 - cropping 6-20
 - filtering of 7-2
 - finding angles between lines 4-18
 - finding edges in 4-2
 - finding histograms of 4-73
 - finding lines in 4-9
 - gradient components 4-2
 - intensity 1-16
 - intensity to binary conversion 5-2
 - labeling objects in 7-3
 - lightening 7-10
 - noisy 4-39
 - periodic noise removal 4-45
 - removing salt and pepper noise 4-39
 - resizing of 6-14
 - RGB 1-16
 - rotation of 6-6
 - segmentation of 7-2
 - sharpening and blurring 4-30
 - true-color 1-16
 - types of 1-15
- importing
 - AVI files 2-14
 - multimedia files 2-8
- improvement
 - of performance 1-28
- increasing video contrast 2-11
- installation
 - DVD 1-3
 - Video and Image Processing Blockset
 - software 1-3
 - Web download 1-3
- intensity

- conversion from R'G'B' 5-14
- conversion to binary 5-2
- definition of 1-16
- images 1-16
- intensity images
 - adjusting the contrast of 4-54
- interference
 - electrical 4-45
- interpolation
 - bicubic 6-4
 - bilinear 6-3
 - examples 6-2
 - nearest neighbor 6-2
 - overview 6-2
- interpretation of
 - matrices 1-15
- irregular illumination 7-10
- isDone method 9-7

K

- key blockset concepts 1-15
- knowledge
 - user's expected 1-12

L

- labeling objects 7-3
- lightening images 7-10
- location of
 - lines 4-9
 - object edges 4-2
 - objects in an image 8-2
- locked vs. unlocked mode 9-4
- luma components
 - applying highpass filter 4-30
 - applying lowpass filter 4-30
 - of images 5-19
- luminance 5-19

M

- matching
 - patterns in an image 8-2
- MATLAB Central
 - demos 1-10
- matrices
 - interpretation of 1-15
- measurement operations 7-2
- median filtering 4-39
- methods
 - interpolation 6-2
 - sum of absolute differences (SAD) 8-10
 - thresholding 7-10
- Microsoft Windows Media Player 2-8
- modes
 - Normal and Accelerator 1-28
- morphology 7-1
 - opening 7-3
 - overview 7-2
 - STREL object 7-3
- motion compensation 8-10
- motion detection 8-10
- multimedia files
 - exporting 2-11
 - exporting audio and video 2-38
 - importing 2-8
 - viewing 2-8

N

- nearest neighbor interpolation 6-2
- noise
 - adding to a signal 4-45
- noise removal
 - periodic 4-45
 - salt and pepper 4-39
- nonuniform illumination
 - correcting for 7-10
- Normal mode 1-28

O

- object boundaries 4-2
- object extraction 7-2
- object tracking
 - using correlation 8-2
- objects
 - delineating 7-10
 - location of 8-2
- opening 7-3
- operations
 - morphological 7-1
 - thresholding 5-2
- organization of the chapters 1-13
- overview of
 - documentation 1-13
 - interpolation 6-2
 - morphology 7-2
 - Video and Image Processing Blockset software 1-2

P

- padding 4-45
- pattern matching 8-2
- performance
 - improving 1-28
- periodic noise
 - removal 4-45
- preferences 10-6
- printing
 - PDF documentation 1-12
- processing
 - in real time 1-29
- product demos 1-5
- products
 - related 1-4
 - required 1-4
- property values 9-5

R

- R'B'G'
 - conversion to intensity 5-14
- real-time processing 1-29
- reception
 - of an RGB image 5-19
- reconstruction
 - of images 8-12
- reduction
 - of image size 6-14
- region of interest
 - cropping to 6-20
 - delineating 10-9
 - System object example 10-9
 - visualizing 8-2
- related products 1-4
- relational operators
 - to perform thresholding 5-2
- removal of
 - periodic noise 4-45
 - salt and pepper noise 4-39
- required products 1-4
- resampling
 - chroma 5-19
- reset method 9-7
- resizing
 - images 6-14
- RGB images 1-16
 - adjusting the contrast of 4-61
- rotation
 - continual 6-6
 - of an image 6-6

S

- salt and pepper noise removal 4-39
- sample time 1-26
- saving
 - to multiple AVI files 2-30
- scaling 1-16

- data types 4-2
 - sectioning
 - AVI files 2-30
 - segmentation operations 7-2
 - sequence
 - of images 2-2
 - setting
 - configuration parameters 1-26
 - simulation time 1-27
 - shape of objects 7-2
 - sharpening images 4-30
 - shrinking
 - image size 6-14
 - simulation time 1-27
 - Simulink Solver 1-26
 - Sobel kernel 4-2
 - splitting
 - AVI files 2-30
 - stabilization
 - of video 8-10
 - step method 9-7
 - storage of image data 1-26
 - streaming data
 - using System objects 9-6
 - STREL object 7-3
 - sum of absolute differences (SAD) method 8-10
 - summary of morphology 7-2
 - System object
 - close method 9-8
 - code generation
 - video and image processing objects 10-3
 - creating an instance 9-3
 - description 9-2
 - fixed point 10-5
 - video and image processing objects 10-5
 - getNumInputs method 9-7
 - getNumOutputs method 9-7
 - isDone method 9-7
 - locked vs. unlocked mode 9-4
 - methods 9-6
 - preferences 10-6
 - properties 9-4
 - property values 9-5
 - reset method 9-7
 - step method 9-7
 - tunable property 9-5
 - using with Embedded MATLAB 9-9
 - value-only input 9-5
 - video and image processing 10-2
 - video and image processing example 10-9
- ## T
- techniques
 - motion compensation 8-10
 - sum of absolute differences (SAD) 8-10
 - thresholding 7-10
 - thresholding operation 5-2
 - with uneven lighting 5-7
 - thresholding techniques 7-10
 - tracking
 - of an object 8-2
 - transformation
 - geometric 6-1
 - transmission
 - of an RGB image 5-19
 - trimming
 - images 6-20
 - true size 2-14
 - true-color images 1-16
 - tunable 9-5
 - tutorials 1-13
 - types of images 1-15
- ## U
- uneven lighting
 - correcting for 7-10

V

- value-only input 9-5
- vectors
 - motion 8-10
- video
 - adjusting display size 2-14
 - annotating AVI files at separate locations 2-26
 - annotating AVI files with video frame numbers 2-22
 - duration 1-27
 - exporting from AVI file 2-18
 - exporting from multimedia file 2-11
 - importing from AVI file 2-14
 - importing from multimedia file 2-8
 - increasing the contrast of 2-11
 - interpretation of 1-16
 - speed of 2-8
 - stabilization 8-10

- video compression and stabilization 8-10

- viewing

- AVI files 2-14
- compressed images 8-18
- demos 1-5
- documentation 1-11
- multimedia files 2-8

- `vip_rt.dll` 1-31

W

- Web

- demos 1-9
- documentation 1-11
- download 1-3

- Windows dynamic libraries

- dependencies on 1-31

- Windows platforms 2-8