Release Notes for Computer Vision System Toolbox™

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Release Notes for Computer Vision System Toolbox™

Contents

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2012b

Version: 5.1
New Features: Yes
Bug Fixes: Yes

Kalman filter and Hungarian algorithm for multiple object tracking

The `vision.KalmanFilter` object is designed for object tracking. You can use it to predict an object's future location, to reduce noise in the detected location, or to help associate multiple objects with their corresponding tracks. The `configureKalmanFilter` function helps you to set up the Kalman filter object.

The `assignDetectionsToTrack` function assigns detections to tracks in the context of multiple object tracking using the James Munkres' variant of the Hungarian assignment algorithm. The function also determines which tracks are missing, and which detections should begin a new track.

Image and video annotation for detected or tracked objects

The `insertObjectAnnotation` function inserts labels and corresponding circles or rectangles into an image or video to easily display tracked objects. You can use it with either a grayscale or true color image input.

Kanade-Lucas-Tomasi (KLT) point tracker

The `vision.PointTracker` object tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature tracking algorithm. You can use the point tracker for video stabilization, camera motion estimation, and object tracking.

HOG-based people detector

The `vision.PeopleDetector` object detects people in an input image using the Histogram of Oriented Gradient (HOG) features and a trained Support Vector Machine (SVM) classifier. The object detects unoccluded people in an upright position.

Video file reader support for H.264 codec (MPEG-4) on Windows 7

This release adds H.264 codec (MPEG-4) video formats for Windows 7 operating systems.

Show matched features display

The `showMatchedFeatures` function displays corresponding feature points. It displays a falsecolor overlay of two images with a color-coded plot of the corresponding points connected by a line.

Matching methods added for match features function
Compatibility Considerations: Yes

This release enhances the `matchFeatures` function for applications in computing the fundamental matrix, stereo vision, registration, and object detection. It provides three different matching methods: simple threshold match, unique matches, and unambiguous matches.

Compatibility Considerations

The new implementation of `matchFeatures` uses different default value for the `method` parameter. If you need the same results as those produced by the previous implementation, set the `Method` parameter with syntax:

```
matchFeatures(FEATURES1, FEATURES2, 'Method',
'NearestNeighbor_old', ...).
```

Kalman filter for tracking tutorial

The Kalman filter is a popular tool for object tracking. The “Using Kalman Filter for Object Tracking” example helps you to understand how to setup and use the `vision.KalmanFilter` object and the `configureKalmanFilter` function to track objects.

Motion-based multiple object tracking example

The “Motion-Based Multiple Object Tracking” example shows you how to perform automatic detection and motion-based tracking of moving objects in a video from a stationary camera.

Face detection and tracking examples

The “Face Detection and Tracking” example shows you how to automatically detect and a track a face. The “Face Detection and Tracking Using the KLT Algorithm” example uses the Kanade-Lucas-Tomasi (KLT) algorithm and shows you how to automatically detect a face and track it using a set of feature points.

Stereo image rectification example

This release enhances the Stereo Image Rectification example. It uses SURF feature detection with the `estimateFundamentalMatrix`, `estimateUncalibratedRectification`, and `detectSURFFeatures` functions to compute the rectification of two uncalibrated images, where the camera intrinsics are unknown.

System object tunable parameter support in code generation

You can change tunable properties in user-defined System objects at any time, regardless of whether the object is locked. For System objects predefined in the software, the object must be locked. In previous releases, you could tune System object properties only for a limited number of predefined System objects in generated code.

`save` and `load` for System objects

You can use the `save` method to save System objects to a MAT file. If the object is locked, its state information is saved, also. You can recall and use those saved objects with the `load` method.

You can also create your own `save` and `load` methods for a System object you create. To do so, use the `saveObjectImpl` and `loadObjectImpl`, respectively, in your class definition file.

Save and restore `SimState` not supported for System objects

Compatibility Considerations: Yes

The Save and Restore Simulation State as SimState option is no longer supported for any System object in a MATLAB Function block. This option was removed because it prevented parameter tunability for System objects, which is important in code generation.

Compatibility Considerations

If you need to save and restore simulation states, you may be able to use a corresponding Simulink block, instead of a System object.

R2012a

Version: 5.0
New Features: Yes
Bug Fixes: Yes

Dependency on DSP System Toolbox and Signal Processing Toolbox Software Removed

Compatibility Considerations: Yes

The DSP System Toolbox™ and Signal Processing Toolbox™ software are no longer required products for using Computer Vision System Toolbox™ software. As a result, a few blocks have been modified or removed.

Audio Output Sampling Mode Added to the From Multimedia File Block

The From Multimedia File block now includes a new parameter, which allows you to select frame- or sample-based audio output. If you do not have a DSP System Toolbox license and you set this parameter for frame-based processing, your model will return an error. The Computer Vision System Toolbox software uses only sample-based processing.

Kalman Filter and Variable Selector Blocks Removed from Library

This release removes the Kalman Filter and Variable Selector Blocks from the Computer Vision System Toolbox block library.

Compatibility Considerations

To use these blocks or to run a model containing these blocks, you must have a DSP System Toolbox license.

2-D Median and 2-D Histogram Blocks Replace Former Median and Histogram Blocks

The Median and Histogram blocks have been removed. You can replace these blocks with the 2-D Median and the 2-D Histogram blocks.

Compatibility Considerations

Replace these blocks in your models with the new 2-D blocks from the Computer Vision System Toolbox library.

Removed Sample-based Processing Checkbox from 2-D Maximum, 2-D Minimum, 2-D Variance, and 2-D Standard Deviation Blocks

This release removes the **Treat sample-based row input as a column** checkbox from the 2-D Maximum, 2-D Minimum, 2-D Variance, and 2-D Standard Deviation statistics blocks.

Compatibility Considerations

Modify your code accordingly.

New Viola-Jones Cascade Object Detector

The `vision.CascadeObjectDetector` System object uses the Viola-Jones algorithm to detect objects in an image. This detector includes Haar-like features and a cascade of classifiers. The cascade object detector is pretrained to detect faces, noses and other objects.

New MSER Feature Detector

The `detectMSERFeatures` function detects maximally stable extremal regions (MSER) features in a grayscale image. You can use the `MSERRegions` object, returned by the function, to manipulate and plot MSER features.

New CAMShift Histogram-Based Tracker

The `vision.HistogramBasedTracker` System object uses the continuously adaptive mean shift (CAMShift) algorithm for tracking objects. It uses the histogram of pixel values to identify the object.

New Integral Image Computation and Box Filtering

The `integralKernel` object with the `integralImage` and `integralFilter` functions use integral images for filtering an image with box filters. The speed of the filtering operation is independent of the filter size, making it ideally suited for fast analysis of images at different scales.

New Demo to Detect and Track a Face

This release provides a new demo, Face Detection and Tracking. This example shows you how to develop a simple face tracking system by detecting a face, identifying its facial features, and tracking it.

Improved MATLAB Compiler Support

MATLAB® Compiler™ now supports `detectSURFFeatures` and disparity functions.

Code Generation Support

The `vision.HistogramBasedTracker` and `vision.CornerDetector` System objects now support code generation. See About MATLAB Coder for more information about code generation.

Conversion of Error and Warning Message Identifiers

Compatibility Considerations: Yes

This release changes error and warning message identifiers.

Compatibility Considerations

If you have scripts or functions using message identifiers that have changed, you must update the code to use the new identifiers. Typically, you use message identifiers to turn off specific warning messages. You can also use them in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the <'XXXXX:old:ID'> identifier has changed to <'new:similar:ID'>. If your code checks for <'XXXXX:old:ID'>, you must update it to check for <'new:similar:ID'> instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error that appears at the MATLAB prompt, run the following command just after you see the error.

```
exception = MException.last;
MSGID = exception.identifier;
```

Note Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code without producing a warning.

System Object Updates

Compatibility Considerations: Yes

Code Generation for System Objects

System objects defined by users now support C code generation. To generate code, you must have the MATLAB Coder™ product.

New System Object Option on File Menu

The File menu on the MATLAB desktop now includes a **New > System object** menu item. This option opens a System object class template, which you can use to define a System object class.

Variable-Size Input Support for System Objects

System objects that you define now support inputs that change size at runtime.

Data Type Support for User-Defined System Objects

System objects that you define now support all MATLAB data types as inputs and outputs.

New Property Attribute to Define States

R2012a adds the new `DiscreteState` attribute for properties in your System object class definition file. Discrete states are values calculated during one step of an object's algorithm that are needed during future steps.

New Methods to Validate Properties and Get States from System Objects

The following methods have been added:

- `validateProperties` – Checks that the System object is in a valid configuration. This applies only to objects that have a defined `validatePropertiesImpl` method
- `getDiscreteState` – Returns a `struct` containing a System object's properties that have the `DiscreteState` attribute

matlab.system.System changed to matlab.System

The base System object class name has changed from `matlab.system.System` to `matlab.System`.

Compatibility Considerations

The previous `matlab.system.System` class will remain valid for existing System objects. When you define new System objects, your class file should inherit from the `matlab.System` class.

R2011b

Version: 4.1
New Features: Yes
Bug Fixes: Yes

Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms

Compatibility Considerations: Yes

Conventions for indexing, spatial coordinates, and representation of geometric transforms have been changed to provide improved interoperability with the Image Processing Toolbox™ product.

Running your Code with New Conventions

How to run code	Solution
Written with R2011b or later (New User)	You can safely ignore the warning, and turn it off. Your code will use the one-based [x y] coordinate system. To turn the warning off, place the following command in your startup.m file: `warning('off','vision:transition:usesOldCoordinates')`
Written prior to R2011b	To run your pre-R2011b code using the zero-based [row column] conventions, invoke `vision.setCoordinateSystem('RC')` command prior to running your code. Support for the pre-R2011b coordinate system will be removed in a future release. You should update your code to use R2011b coordinate system conventions. To turn the warning off, place the following command in your startup.m file: `warning('off','vision:transition:usesOldCoordinates')`

One-Based Indexing

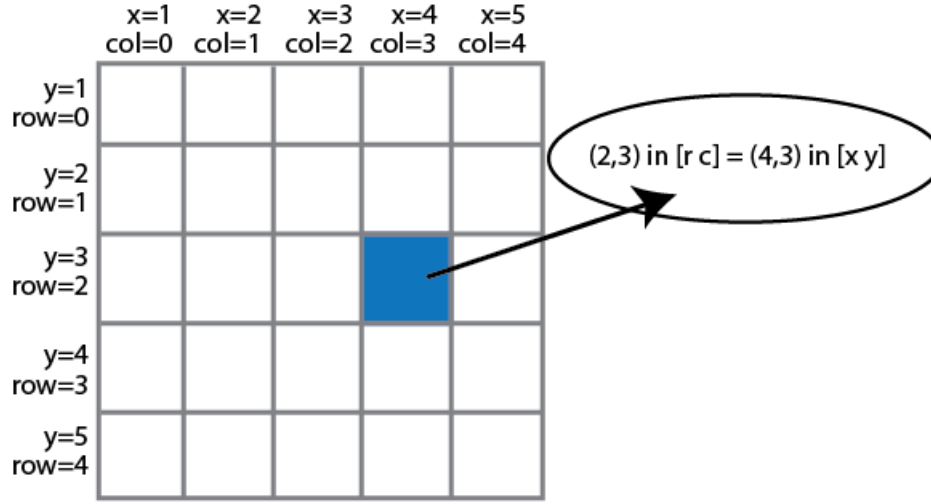
The change from zero-based to one-based indexing simplifies the ability to blend Image Processing Toolbox functionality with Computer Vision System Toolbox algorithms and visualization functions.

Coordinate System Convention

Image locations in the Computer Vision System Toolbox are now expressed in [x y] coordinates, not in [row column]. The orientation of matrices containing image locations has changed. In previous releases, the orientation was a 2-by-*N* matrix of zero-based [row column] point coordinates. Effective in R2011b, the orientation is an *M*-by-2 matrix of one-based [x y] point coordinates. Rectangular ROI representation changed from [r c height width] to [x y width height].

Example: Convert a point represented in the [r c] coordinate system to a point in the [x y] coordinate system

Convert your data to be consistent with MATLAB and the Image Processing Toolbox coordinate systems by switching the order indexing and adding `1` to each dimension. The *row* index dimension corresponds to the *y* index, and the *column* index corresponds to the *x* index. The following figure shows the equivalent row-column and x-y coordinates for a pixel location in an image.



The following MATLAB code converts point coordinates from an [r c] coordinate system to the [x y] coordinate system:

```
ptsRC = [2 0; 3 5]    % Two RC points at [2 3] and [0 5]
ptsXY = fliplr(ptsRC'+1) % RC points converted to XY
```

Example: Convert a bounding box represented in the [r c] coordinate system to the [x y] coordinate system

```
% Two bounding boxes represented as [r c height width]
% First box is [2 3 10 5] and the second box is[0 5 15 10]
   bboxRC = [2 0; 3 5; 10 15; 5 10]
% Convert the boxes to XY coordinate system format [x y width heigth]
   bboxXY = [fliplr(bboxRC(1:2,:)'+1) fliplr(bboxRC(3:4,:)')]
```

Example: Convert an affine geometric transformation matrix represented in the [r c] coordinate system to the [x y] coordinate system

```
% Transformation matrix [h1 h2 h3; h4 h5 h6] represented in RC coordinates
   tformRC = [5 2 3; 7 8 13]
% Transformation matrix [h5 h2; h4 h1; h6 h3] represented in XY coordinates
   temp = rot90(tformRC,3);
   tformXY = [flipud(temp(1:2,:)); temp(3,:)]
```

Note: You cannot use this code to remap a projective transformation matrix. You must derive the tformXY matrix from your data.

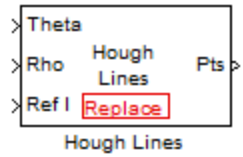
See Expressing Image Locations for an explanation of pixel and spatial coordinate systems.

Migration to [x y] Coordinate System

By default, all Computer Vision System Toolbox blocks, functions, and System objects are set to operate in the [x y] coordinate system. Use the `vision.setCoordinateSystem` and `vision.getCoordinateSystem` functions to help you migrate your code, by enabling you to revert to the previous coordinate system until you can update your code. Use `vision.setCoordinateSystem('RC')` call to set the coordinate system back to the zero-based [r c] conventions .

For Simulink users, blocks affected by the [x y] coordinate system should be replaced with blocks of the same name from the Vision library. Old blocks are

marked with a red “Replace” badge. The following figure shows the Hough Lines block, as it would appear with the Replace badge, indicating that it should be replaced with the Hough Lines block from the R2011b version.



Support for the pre-R2011b coordinate system will be removed in a future release.

Updated Blocks, Functions, and System Objects

The following table provides specifics for the functions, System objects, and blocks that were affected by this update:

Functions	Description of Update	Prior to R2011b	R2011b
`epipolarLine`	The output *A*,*B*,*C* line parameters were changed to work with [x y] one-based coordinates.	*A**row + *B**col + *C*	*A**x + *B**y + *C*
	Accepts Fundamental matrix in [x y] format.		
`estimateFundamentalMatrix`	Adjusted to format of fundamental matrix. Modified to work with points expressed in [x y] one-based coordinates.	[r;c] 2-by-*N* zero-based points.	[x y] *M*-by-2 one-based points.
		Fundamental matrix formatted points for [r;c] zero-based coordinates.	Fundamental matrix formatted to work with [x y] one-based coordinates.

Functions	Description of Update	Prior to R2011b	R2011b
`estimateUncalibratedRectification`	Fundamental matrix, matched points, and output projective transformation	Fundamental matrix formatted only for [r;c] 2-by-*N* zero-based points	Fundamental matrix formatted [x y] *M*-by-2 one-based points
`extractFeatures`	Converted to accept [x y] coordinates	[r;c] 2-by-*N* zero-based points.	[x y] *M*-by-2 one-based points.
`isEpipoleInImage`	Adjusted Fundamental matrix format. Converted to [x y] coordinates.	Fundamental matrix formatted only for zero-based [r;c] coordinate system.	Fundamental matrix formatted only for one-based, [x y] coordinate system.
`lineToBorderPoints`	The input *A*,*B*,*C* line parameters were changed to work with [x y] coordinates.	*A**row + *B**col + *C*, where A,B, and C are represented in a 3-by-*N* matrix of [r;c] zero-based points.	*A**x + *B**y + *C*, where A,B, and C are represented in an *M*-by-3 matrix of [x y] one-based points.
	Output intersection points converted to [x y] one-based	The function returned the intersection points in an 4-by-*M* matrix	The function returns the intersection points in an *M*-by-4 matrix
`matchFeatures`	Converted the Index Pairs matrix to match orientation of the POINTS with [x y] one-based coordinates.	The function returns the output Index Pairs in a 2-by-*M* [r c] zero-based format.	The function returns the output Index Pairs in a *M*-by-2 [x y] one-based format.
	Changed orientation of input feature vectors.	Input feature vectors stored in columns.	Input feature vectors stored in rows.

System Objects	Description of Update	Prior to R2011b	R2011b
`vision.AlphaBlender`	Converted Location property to take [x y] coordinate location.	Location format in [r;c] zero-based coordinates.	Location format in [x y] one-based coordinates.
`vision.BlobAnalysis`	Centroid and Bounding Box formats converted to [x y] coordinate system.	Centroid format in 2-by-*M* [r1 r2; c1 c2] zero-based coordinates.	Centroid format in *M*-by-2 of format [x1 y1 x2 y2] one-based coordinates.
		Bounding Box format in 4-by-*N* zero-based matrix [r;c;height;width].	Bounding Box format in *M*-by-4 one-based matrix [x y width height].
`vision.BoundaryTracer`	Converted to accept and output [x y] one-based points.	2-by-*N* matrix of [r c] zero-based coordinates.	*M*-by-2 matrix of [x y] one-based coordinates.
`vision.CornerDetector`	Corner locations converted to [x y] coordinate system.	Corner location in a 2-by-*N* set of [r c] zero-based coordinates.	Corner locations in an *M*-by-2 one-based [x y] coordinates.
`vision.GeometricScaler`	Converted ROI input to [x y] coordinate one-based system.	Shape in [r c height width] zero-based matrix.	Shape in [x y width height] one-based matrix.

System Objects	Description of Update	Prior to R2011b	R2011b
`vision.GeometricTransformer`	Converted transformation matrix format to support changed ROI [x y] one-based coordinate system format.	Transformation matrix formatted only for zero-based [r;c] coordinate system.	Takes one-based, [x y] coordinate format for Transformation matrix.
		ROI format in [r;c;height;width] zero-based format.	ROI format in [x y width height] one-based format.
`vision.GeometricTransformEstimation`	Changed formatting for input points.	Input points: [r1 r2;c1 c2].	Input points: [x1 y1; x2 y2].
	Converted transformation matrix to [x y] one-based coordinate system.	Transformation matrix formatted only for zero-based [r;c] coordinate system.	Transformation matrix format matches Image Processing Toolbox format.
`vision.HoughLines`	Converted format for lines to [x y] one-based coordinate system.	Output: [r11 r21; c11 c21; r12 r22; c12 c22].	Output: [x11 y11 x12 y12; x21 y21 x22 y22].
		Size of output in a 4-by-*N* zero-based matrix.	Size of the output in *M*-by-4 one-based matrix.
`vision.LocalMaximaFinder`	Converted format for Maxima locations	2-by-*N* zero-based [r c] coordinates.	*M*-by-2 one-based [x y] coordinates.
`vision.MarkerInserter`	Converted format for locations.	2-by-*N* zero-based [r c] coordinates.	*M*-by-2, one-based [x y] coordinates.

System Objects	Description of Update	Prior to R2011b	R2011b

`vision.Maximum`
`vision.Mean`
`vision.Minimum`
`vision.StandardDeviation`
`vision.Variance`
	Converted formats for line and rectangle inputs.	Line: [r1 c1 r2 c2 r3 c3].	Line: [x1 y1 x2 y2 x3 y3].

		Rectangle: [r c height width].	Rectangle: [x y width height].
`vision.ShapeInserter`	Converted format for rectangles, lines, polygons, and circles to [x y] one-based format.	Rectangle: [r; c; height; width] zero-based format.	Rectangle: [x y width height] one-based format.
		Line: [r1 c1 r2 c2] zero-based format.	Line: [x1 y1 x2 y2] one-based format.
		Polygon: 4-by-*M* zero-based matrix.	Polygon: *M*-by-4 one-based matrix.
		Circle: [r c radius] zero-based format.	Circle: [x y radius] one-based format.
	Input image intensity values converted to [x y] one-based format.	*N*-by-*M* and *N*-by-*M*-by-*P* [r c] zero-based format.	*M*-by-*N* and *M*-by-*N*-by-*P* [x y] one-based format.
`vision.TemplateMatcher`	Converted Location and ROI format to [x y] one-based coordinate system.	Location output: [r; c] zero-based format.	Location output: [x y] one-based format.
		ROI: [r c height width] zero-based format.	ROI processing: [x y width height] one-based format.

System Objects	Description of Update	Prior to R2011b	R2011b
`vision.TextInserter`	Converted location and color orientation.	2-by-*N* zero-based [r;c] locations.	*M*-by-2 [x y] one-based locations.
		numColorPlanes-by-*M* zero-based format.	*M*-by-*numColorPlanes* one-based format.

Blocks	Description of Update	Prior to R2011b	R2011b
Apply Geometric Transformation	Converted Transformation matrix format to support changed ROI [x y] one-based coordinate system format.	Transformation matrix formatted only for zero-based [r;c] coordinate system.	Takes one-based, [x y] coordinate format for Transformation matrix.
		ROI format in [r;c;height;width] zero-based format.	ROI format in [x y width height] one-based format.
Blob Analysis	Centroid and Bounding Box formats converted to [x y] coordinate system.	Centroid format in 2-by-*M* [r1 r2; c1 c2] zero-based coordinates.	Centroid format in *M*-by-2 of format [x1 y1 x2 y2] one-based coordinates.
		Bounding Box format in 4-by-*N* zero-based matrix [r;c;height;width].	Bounding Box format in *M*-by-4 one-based matrix [x y width height].
Compositing	Converted Location property to takes [x y] coordinate location.	Location format in [r;c] zero-based coordinates.	Location format in [x y] one-based coordinates.

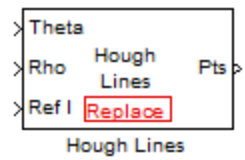
Blocks	Description of Update	Prior to R2011b	R2011b
Corner Detection	Corner locations converted to [x y] coordinate system.	Corner location in a 2-by-*N* set of [r c] zero-based coordinates.	Corner locations in an *M*-by-2 one-based [x y] coordinates.
Draw Markers	Converted format for locations.	2-by-*N* zero-based [r c] coordinates.	*M*-by-2, one-based [x y] coordinates.
Draw Shapes	Converted format for rectangles, lines, polygons, and circles to [x y] one-based format.	Rectangle: [r; c; height; width] zero-based format.	Rectangle: [x y width height] one-based format.
		Line: [r1 c1 r2 c2] zero-based format.	Line: [x1 y1 x2 y2] one-based format.
		Polygon: 4-by-*M* zero-based matrix.	Polygon: *M*-by-4 one-based matrix.
		Circle: [r c radius] zero-based format	Circle: [x y radius] one-based format.
Estimate Geometric Transformation	Converted formatting for input points.	Input points: [r1 r2;c1 c2].	Input points: [x1 y1; x2 y2].
	Converted Transformation matrix to [x y] one-based coordinate system.	Transformation: T=[t22 t12 t32; t21 t11 t31; t23 t13 t33].	Transformation matrix format matches Image Processing Toolbox format.

Blocks	Description of Update	Prior to R2011b	R2011b
Find Local Maxima	Converted format for Maxima locations	2-by-*N* zero-based [r c] coordinates.	*M*-by-2 one-based [x y] coordinates.
Hough Lines	Converted format for lines to [x y] one-based coordinate system.	Output: [r11 r21; c11 c21; r12 r22; c12 c22].	Output: [x11 y11 x12 y12; x21 y21 x22 y22].
		Size of output in a 4-by-*N* zero-based matrix.	Size of the output in *M*-by-4 one-based matrix.
Template Matching	Converted Location and ROI format to [x y] one-based coordinate system.	Location output: [r; c] zero-based format.	Location output: [x y] one-based format.
		ROI: [r c height width] zero-based format.	ROI processing: [x y width height] one-based format.
Insert Text	Converted location and color orientation.	2-by-*N* zero-based [r;c] locations.	*M*-by-2 [x y] one-based locations.
		numColorPlanes-by-*N* zero-based format.	*M*-by-*numColorPlanes* one-based format.
2-D Maximum2-D Mean2-D Minimum2-D Standard Deviation2-D Variance	Converted formats for line and rectangle ROIs.	Line: [r1 c1 r2 c2 r3 c3].	Line: [x1 y1 x2 y2 x3 y3].
		Rectangle: [r c height width].	Rectangle: [x y width height].

Blocks	Description of Update	Prior to R2011b	R2011b
Resize	Converted ROI input to [x y] coordinate one-based system.	Shape in [r c height width] zero-based matrix.	Shape in [x y width height] one-based matrix.
Trace Boundary	Converted to accept and output [x y] one-based points.	2-by-*N* matrix of [r c] zero-based coordinates.	*M*-by-2 matrix of [x y] one-based coordinates.

Compatibility Considerations

Blocks affected by the [x y] coordinate system should be replaced with blocks of the same name from the Vision library. Old blocks are marked with a red "Replace" badge. The following figure shows a block which was affected by the coordinate system change:



Adjust your model and data as necessary. All functions and System objects are updated to use the one-based [x y] convention.

By default, all Computer Vision System Toolbox blocks, functions, and System objects are set to operate in the [x y] coordinate system. Use the `vision.setCoordinateSystem` and `vision.getCoordinateSystem` functions to help migrate your code containing System objects and functions to the [x y] coordinate system. Use `vision.setCoordinateSystem('RC')` call to temporarily set the coordinate system to old conventions.

When you invoke an affected block, object, or function, a one time, per MATLAB session, warning appears.

See the section, Expressing Image Locations for a description of the coordinate systems now used by the Computer Vision System Toolbox product.

New SURF Feature Detection, Extraction, and Matching Functions

This release introduces a new Speeded Up Robust Features (SURF) detector with functions supporting interest feature detection, extraction and matching. The `detectSURFFeatures` function returns information about SURF features detected in a grayscale image. You can use the `SURFPoints` object returned by the `detectSURFFeatures` function to manipulate and plot SURF features.

New Disparity Function for Depth Map Calculation

The new `disparity` function provides the disparity map between a pair of stereo images. You can use the `disparity` function to find relative depth of the scene for tasks such as, segmentation, robot navigation, or 3-D scene reconstruction.

Added Support for Additional Video File Formats for Non-Windows Platforms

The From Multimedia File block and the `vision.VideoFileReader` now support many compressed video file formats on Linux® and Macintosh OS X platforms.

Variable-Size Support for System Objects

Computer Vision System Toolbox System objects support inputs that change their size at run time.

New Demo to Retrieve Rotation and Scale of an Image Using Automated Feature Matching

This release provides a new demo, Finding the Rotation and Scale of an Image Using Automated Feature Matching. This demo shows you how to use the `vision.GeometricTransformEstimator` System object and the new `detectSURFFeatures` function to find the rotation angle and scale factor of a distorted image.

Apply Geometric Transformation Block Replaces Projective Transformation Block

The Projective Transformation block will be removed in a future release. It is recommended that you replace this block with the combination of Apply Geometric Transformation and the Estimate Geometric Transformation blocks to apply projective or affine transform to an image.

Trace Boundaries Block Replaced with Trace Boundary Block

Compatibility Considerations: Yes

This release provides a replacement block for the Trace Boundaries block. The Trace Boundary block now returns variable size data. See Working with Variable-Size Signals for more information about variable size data.

Note Unlike the Trace Boundaries block, the new Trace Boundary block only traces a single boundary.

The Trace Boundaries block will be removed in a future release.

Compatibility Considerations

The new Trace Boundary block no longer provides the **Count** output port that the older Trace Boundaries block provided. Instead, the new Trace Boundary block and the corresponding `vision.BoundaryTracer` System object now return variable size data.

FFT and IFFT Support for Non-Power-of-Two Transform Length with FFTW Library

The 2-D FFT and 2-D IFFT blocks and the `vision.IFFT` and `vision.FFT` System objects include the use of the FFTW library. The blocks and objects now support non-power-of-two transform lengths.

vision.BlobAnalysis Count and Fill-Related Properties Removed

Compatibility Considerations: Yes

The blob analysis System object now supports variable-size outputs. Therefore, the `Count` output, and the `NumBlobsOutputPort`, `FillEmptySpaces`, and `FillValues` properties related to fixed-size outputs, were removed from the object.

Compatibility Considerations

Remove these properties from your code, and update accordingly. If you require an explicit blob count, call `size` on one of the object's outputs, such as `AREA`.

vision.CornerDetector Count Output Removed

Compatibility Considerations: Yes

The corner detector System object now supports variable-size outputs. Therefore, the `Count` output related to fixed-size outputs, were removed from the object.

Compatibility Considerations

Update your code accordingly. If you require an explicit count, call `size` on the object `METRIC` output.

vision.LocalMaximaFinder Count Output and CountDataType Property Removed

Compatibility Considerations: Yes

The local maxima finder System object now supports variable-size outputs. Therefore, the `Count` output, and the `CountDataType` property related to fixed-size outputs, were removed from the object.

Compatibility Considerations

Remove the property from your code, and update accordingly.

vision.GeometricTransformEstimator Default Properties Changed

Compatibility Considerations: Yes

The following default property values for the `vision.GeometricTransformEstimator` System object have been changed to provide more reliable outputs.

Property	Default Value	
	From	To
`Transform`	`Projective`	`Affine`
`AlgebraicDistanceThreshold`	`1.5`	`2.5`
`PixelDistanceThreshold`	`1.5`	`2.5`
`NumRandomSamplings`	`100`	`500`
`MaximumRandomSamples`	`200`	`1000`

Compatibility Considerations

The effect of these changes make the object's default-value computations more reliable. If your code relies on the previous default values, you might need to update the affected property values.

Code Generation Support

The `vision.IFFT` System object now supports code generation. See About MATLAB Coder for more information about code generation.

vision.MarkerInserter and vision.ShapeInserter Properties Not Tunable

Compatibility Considerations: Yes

The following `vision.MarkerInserter` and `vision.ShapeInserter` properties are now nontunable:

- `FillColor`
- `BorderColor`

When objects are locked (for instance, after calling the `step` method), you cannot change any nontunable property values.

Compatibility Considerations

Review any code that changes any `vision.MarkerInserter` or `vision.ShapeInserter` property value after calling the `step` method. You should update the code to use property values that do not change.

Custom System Objects

You can now create custom System objects in MATLAB. This capability allows you to define your own System objects for time-based and data-driven algorithms, I/O, and visualizations. The System object API provides a set of implementation and service methods that you incorporate into your code to implement your algorithm. See Define New System Objects in the DSP System Toolbox documentation for more information.

System Object DataType and CustomDataType Properties Changes

Compatibility Considerations: Yes

When you set a System object, fixed-point `<xxx>DataType` property to `` `Custom' ``, it activates a dependent `Custom<xxx>DataType` property. If you set that dependent `Custom<xxx>DataType` property before setting its `<xxx>DataType` property, a warning message displays. `<xxx>` differs for each object.

Compatibility Considerations

Previously, setting the dependent `Custom<xxx>DataType` property would automatically change its `<xxx>DataType` property to `` `Custom' ``. If you have code that sets the dependent property first, avoid warnings by updating your code. Set the `<xxx>DataType` property to `` `Custom' `` before setting its `Custom<xxx>DataType` property.

Note If you have a `Custom<xxx>DataType` in your code, but do not explicitly update your code to change `<xxx>DataType` to `` `Custom' ``, you may see different numerical output.

R2011a

Version: 4.0
New Features: Yes
Bug Fixes: Yes

Product Restructuring

The Video and Image Processing Blockset has been renamed to Computer Vision System Toolbox. This product restructuring reflects the broad expansion of computer vision capabilities for the MATLAB and Simulink® environments. The Computer Vision System Toolbox software requires the Image Processing Toolbox and DSP System Toolbox software.

You can access archived documentation for the Video and Image Processing Blockset™ products on the MathWorks website.

System Object Name Changes

Package Name Change

The System object package name has changed from video to vision. For example, `video.BlobAnalysis` is now `vision.BlobAnalysis`.

Object Name Changes

The 2D System object names have changed. They no longer have 2D in the name and now use the new package name.

Old Name	New Name
video.Autocorrelator2D	`vision.Autocorrelator`
video.Convolver2D	`vision.Convolver`
video.Crosscorrelator2D	`vision.Crosscorrelator`
video.DCT2D	`vision.DCT`
video.FFT2D	`vision.FFT`
video.Histogram2D	`vision.Histogram`
video.IDCT2D	`vision.IDCT`
video.IFFT2D	`vision.IFFT`
video.MedianFilter2D	`vision.MedianFilter`

New Computer Vision Functions

Extract Features

The `extractFeatures` function extracts feature vectors, also known as descriptors, from an image.

Feature Matching

The `matchFeatures` function takes a pair of feature vectors, as returned by the `extractFeatures` function, and finds the features which are most likely to correspond.

Uncalibrated Stereo Rectification

The `estimateUncalibratedRectification` function returns projective transformations for rectifying stereo images.

Determine if Image Contains Epipole

The `isEpipoleInImage` function determines whether an image contains an epipole. This function supports the `estimateUncalibratedRectification` function.

Epipolar Lines for Stereo Images

The `epipolarLine` computes epipolar lines for stereo images.

Line-to-Border Intersection Points

The `lineToBorderPoints` function calculates the location of the point of intersection of line in an image with the image border. This function supports the `epipolarLine` function.

New Foreground Detector System Object

The `vision.ForegroundDetector` object computes a foreground mask using Gaussian mixture models (GMM).

New Tracking Cars Using Gaussian Mixture Models Demo

The new Tracking Cars Using Gaussian Mixture Models demo illustrates the use of Gaussian mixture models for detection and tracking of cars. The algorithm detects and tracks the cars in a video by separating them from their background.

Expanded To Video Display Block with Additional Video Formats

The To Video Display block now supports 4:2:2 YCbCr video input format.

New Printing Capability for the mplay Function and Video Viewer Block

You can now print the display information from the GUI interface of the mplay function and the Video Viewer block.

Improved Display Updates for mplay Function, Video Viewer Block and vision.VideoPlayer System Object

R2011a introduces the capability to improve the performance of `mplay`, the Video Viewer block and the `vision.VideoPlayer` System object by reducing the frequency with which the display updates. You can now choose between this new enhanced performance mode and the old behavior. By default, all scopes operate in the new enhanced performance mode.

Improved Performance of FFT Implementation with FFTW library

The 2-D FFT, 2-D IFFT blocks include the use of the FFTW library.

Variable Size Data Support

The Resize block now supports variable size data. See Working with Variable-Size Signals for more information about variable size data.

System Object Input and Property Warnings Changed to Errors

Compatibility Considerations: Yes

When a System object is locked (e.g., after the `step` method has been called), the following situations now produce an error. This change prevents the loss of state information.

- Changing the input data type
- Changing the number of input dimensions
- Changing the input complexity from real to complex
- Changing the data type, dimension, or complexity of tunable property
- Changing the value of a nontunable property

Compatibility Considerations

Previously, the object issued a warning for these situations. The object then unlocked, reset its state information, relocked, and continued processing. To update existing code so that it does not error, use the `release` method before changing any of the items listed above.

System Object Code Generation Support

The following System objects now support code generation:

- `vision.GeometricScaler`
- `vision.ForegroundDetector`

MATLAB Compiler Support for System Objects

The Computer Vision System Toolbox supports the MATLAB Compiler for all objects except `vision.VideoPlayer`. With this capability, you can use the MATLAB Compiler to take MATLAB files, which can include System objects, as input and generate standalone applications.

R2010a MAT Files with System Objects Load Incorrectly

Compatibility Considerations: Yes

If you saved a System object to a MAT file in R2010a and load that file in R2011a, MATLAB may display a warning that the constructor must preserve the class of the returned object. This occurs because an aspect of the class definition changed for that object in R2011a. The object's saved property settings may not restore correctly.

Compatibility Considerations

MAT files containing a System object saved in R2010a may not load correctly in R2011a. You should recreate the object with the desired property values and save the MAT file.

Documentation Examples Renamed

Compatibility Considerations: Yes

In previous releases, the examples used throughout the Video and Image Processing Blockset™ documentation were named with a `doc_` prefix. In R2011a, this changed to a `ex_` prefix. For example, in R2010b, you could launch an example model using the Video Viewer block by typing `doc_thresholding` at the MATLAB command line. To launch the same model in R2011a, you must type `ex_thresholding` at the command line.

Compatibility Considerations

You can no longer launch Video and Image Processing Blockset™ documentation example models using the `doc_` prefix name. To open these models in R2011a, you must replace the `doc_` prefix in the model name with `ex_`.

R2010b

Version: 3.1
New Features: Yes
Bug Fixes: Yes

New Estimate Fundamental Matrix Function for Describing Epipolar Geometry

New `Estimate Fundamental Matrix` function for describing epipolar geometry. Epipolar geometry applies to the geometry of stereo vision, where you can calculate depth information based on corresponding points in stereo image pairs. The function supports the generation of embeddable C code.

New Histogram System Object Replaces Histogram2D Object

Compatibility Considerations: Yes

The new `video.Histogram` System object replaces the video.Histogram2D System object. The name change was made to align this object with its corresponding block.

Compatibility Considerations

The `video.Histogram2D` System object now issues a warning. Update code that uses the 2D-Histogram object to use the new Histogram object.

New System Object release Method Replaces close Method

Compatibility Considerations: Yes

The `close` method has been replaced by the new `release` method, which unlocks the object and releases memory and other resources, including files, used by the object. The new `release` method includes the functionality of the old `close` method, which only closed files used by the object.

Compatability Considerations

The `close` method now issues a warning. Update code that uses the `close` method to use the new `release` method.

Expanded Embedded MATLAB Support

Embedded MATLAB® now supports the generation of embeddable C code for two Image Processing Toolbox functions and additional Video and Image Processing Blockset System objects. The generated C code meets the strict memory and data type requirements of embedded target environments. Video and Image Processing Blockset provides Embedded MATLAB support for these Image Processing Toolbox functions. See Code Generation for details, including limitations.

Supported Image Processing Toolbox Functions

```
label2rgb
fspecial
```

Supported System objects

Video and Image Processing Blockset objects now support code generation:

```
video.CornerDetector
video.GeometricShearer
video.Histogram
video.MorpologicalBottomHat
video.MorpologicalTopHat
video.MultimediaFileReader
video.MultimediaFileWriter
```

Data Type Assistant and Ability to Specify Design Minimums and Maximums Added to More Fixed-Point Blocks

The following blocks now offer a **Data Type Assistant** to help you specify fixed-point data types on the block mask. Additionally, you can now enable simulation range checking for certain data types on these blocks. To do so, specify appropriate minimum and maximum values on the block dialog box. The blocks that support these features are:

- 2-D DCT
- 2-D FFT
- 2-D IDCT
- 2-D IFFT
- 2-D FIR Filter

For more information on these features, see the following sections in the Simulink documentation:

- Using the Data Type Assistant
- Signal Ranges

Data Types Pane Replaces the Data Type Attributes and Fixed-Point Panes on Fixed-Point Blocks

In previous releases, some fixed-point blocks had a **Data type attributes** pane, and others had a **Fixed-point** pane. The functionality of these panes remains the same, but the pane now appears as the **Data Types** pane on all fixed-point Computer Vision System Toolbox blocks.

Enhanced Fixed-Point and Integer Data Type Support with System Objects

Compatibility Considerations: Yes

For nonfloating point input, System objects now output the data type you specify. Previously, the output was always a fixed-point, numeric `fi` object.

Compatability Considerations

Update any code that takes nonfloating point input, where you expect the object to output a `fi` object.

Variable Size Data Support

Several Video and Image Processing Blockset blocks now support changes in signal size during simulation. The following blocks support variable size data as of this release:

PSNR	2-D Correlation
Median Filter	2-D Convolution
Block Processing	2-D Autocorrelation
Image Complement	Deinterlacing
Gamma Correction	

See Working with Variable-Size Signals for more information about variable size data.

Limitations Removed from Video and Image Processing Blockset Multimedia Blocks and Objects

Support for reading interleaved AVI data and reading AVI files larger than 2GB on UNIX platforms. Previously, this was only possible on Windows platforms. The following blocks and System objects have the limitation removed:

From Multimedia File block
`video.MultimediaFileReader` System object

Support for writing AVI files larger than 2GB on UNIX platforms, which was previously only possible on Windows platforms. The following blocks and System objects have the limitation removed:

To Multimedia File block
`video.MultimediaFileWriter` System object

R2010a

Version: 3.0
New Features: Yes
Bug Fixes: Yes

New System Objects Provide Video and Image Processing Algorithms for use in MATLAB

System Objects are algorithms that provide stream processing, fixed-point modeling, and code generation capabilities for use in MATLAB programs. These new objects allow you to use video and image processing algorithms in MATLAB, providing the same parameters, numerics and performance as corresponding Video and Image Processing Blockset blocks. System objects can also be used in Simulink models via the Embedded MATLAB Function block.

Intel Integrated Performance Primitives Library Support Added to 2-D Correlation, 2-D Convolution, and 2-D FIR Filter Blocks

The 2-D Correlation, 2-D Convolution, and 2-D FIR Filter blocks are now taking advantage of SSE Intel instruction set and multi-core processor capabilities for double and single data types.

Variable Size Data Support

Several Video and Image Processing Blockset blocks now support changes in signal size during simulation. The following blocks support variable size data as of this release:

2-D FFT	Hough Transform
2-D FIR Filter	Image Data Type Conversion
Apply Geometric Transformation	Image Pad
Autothreshold	Insert Text
Bottom-hat	Label
Chroma Resampling	2-D Maximum
Closing	2-D Mean
Color Space Conversion	
Compositing	2-D Minimum
Contrast Adjustment	Opening
Dilation	Rotate
Edge Detection	2-D Standard Deviation
Erosion	Template Matching
Estimate Geometric Transformation	To Video Display
Find Local Maxima	Top-hat
Frame Rate Display	2-D Variance
Gaussian Pyramid	Video Viewer

See Working with Variable-Size Signals for more information about variable size data.

Expanded From and To Multimedia File Blocks with Additional Video Formats

The To Multimedia File and From Multimedia File blocks now support 4:2:2 YCbCr video formats.

The To Multimedia File block now supports `WMV`, `WMA`, and `WAV` file formats on Windows® platforms. This block now supports broadcasting `WMV` and `WMA` streams over the network.

New Simulink Demos

The Video and Image Processing Blockset contain new and enhanced demos.

New Modeling a Video Processing System for an FPGA Target Demo

This demo uses the Video and Image Processing Blockset in conjunction with Simulink HDL Coder™ to show a design workflow for generating Hardware Design Language (HDL) code suitable for targeting video processing application on an FPGA. The demo reviews how to design a system that can operate on hardware.

New System Object Demos

New Image Rectification Demo

This demo shows how to rectify two uncalibrated images where the camera intrinsics are unknown. Rectification is a useful procedure in many computer vision applications. For example, in stereo vision, it can be used to reduce a 2-D matching problem to a 1-D search. This demo is a prerequisite for the Stereo Vision demo.

New Stereo Vision Demo

This demo computes the depth map between two rectified stereo images using block matching, which is the standard algorithm for high-speed stereo vision in hardware systems. It further explores dynamic programming to improve accuracy, and image pyramiding to improve speed.

New Video Stabilization Using Point Feature Matching

This demo uses a point feature matching approach for video stabilization, which does not require knowledge of a feature or region of the image to track. The demo automatically searches for the background plane in a video sequence, and uses its observed distortion to correct for camera motion. This demo presents a more advanced algorithm in comparison to the existing Video Stabilization demo in Simulink.

SAD Block Obsoleted

The new Template Matching block introduced in the previous release, supports Sum of Absolute Differences (SAD) algorithm. Consequently, the SAD Block has been obsoleted.