Computer Vision Toolbox™ Release Notes

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Computer Vision Toolbox™ Release Notes

Contents

R2022a

R2019a

R2016a

R2015b

R2022a

Ground Truth Labeling

Labeler Enhancements: Labeling interactions and other enhancements

The following table describes enhancements for these labeling apps:

- **Image Labeler**
- **Video Labeler**
- **Ground Truth Labeler**
- **Lidar Labeler**

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Draw, visualize, and export semantic labels in the point cloud.	No	No	No	Yes
Create training data for object detection in point clouds by using the `lidarObjectDetectorTrainingData` function.	No	No	No	Yes
Import multiframe DICOM images.	Yes	No	No	No
Create 3-D line ROI for point cloud data.	No	No	Yes	Yes
Create voxel ROI for point cloud data.	No	No	No	Yes
Show or hide pixel labels in a labeled image or video.	Yes	Yes	Yes	No

Feature Detection and Extraction

Functionality Being Removed or Changed: Support for GPU removed for detectFASTFeatures function

The `detectFASTFeatures` function no longer supports GPU. You can use the `detectHarrisFeatures` function on the GPU to detect corner points instead. Though `detectHarrisFeatures` does not provide identical results, it might be suitable based on the application.

Recognition, Object Detection, and Semantic Segmentation

Deep Learning Instance Segmentation: Train Mask R-CNN networks

The `trainMaskRCNN` function trains a Mask R-CNN network represented by a `maskrcnn` object. You can perform transfer learning on a pretrained Mask R-CNN network. This functionality requires the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation and a Deep Learning Toolbox™ license.

YOLO v4 Object Detector: Detect objects in image using YOLO v4 deep learning network

The `yolov4ObjectDetector` object creates a YOLO v4 object detector to detect objects in images. You can either create a pretrained YOLO v4 object detector or a custom YOLO v4 object detector. Then, use the `detect` object function to detect objects in a test image by using the trained YOLO v4 object detector. To train a YOLO v4 object detection network, use the `trainYOLOv4ObjectDetector` function.

Create Pretrained YOLO v4 Object Detector

- To create a pretrained YOLO v4 object detector, use the pretrained DarkNet-53 and tiny YOLO v4 deep learning networks in the Computer Vision Toolbox Model for YOLO v4 Object Detection support package. These networks are trained on the COCO data set.

  You can download the Computer Vision Toolbox Model for YOLO v4 Object Detection from the Add-On Explorer. For more information, see “Get and Manage Add-Ons”.

Create Custom YOLO v4 Object Detector

- You can create a custom YOLO v4 object detector by using an untrained or pretrained YOLO v4 deep learning network.
- You can also create a custom YOLO v4 object detector by using a base network for feature extraction. The `yolov4ObjectDetector` object adds detection heads to the feature extraction layers of the base network and creates a YOLO v4 object detector. You must specify the source layers to which to add the detection heads.

  The “Object Detection Using YOLO v4 Deep Learning” example shows how to create and train a custom YOLO v4 object detector for performing object detection.

SSD Object Detector Enhancements: Create pretrained or custom SSD object detector

Starting in R2022a, use the `ssdObjectDetector` object to create a pretrained or custom SSD object detection network. Then, use the `detect` object function to detect objects in a test image by using the trained SSD object detector. To train a SSD object detector, use the `trainSSDObjectDetector` function.

Text Detection: Detect natural scene texts using CRAFT deep learning model

The `detectTextCRAFT` function detects text in natural scene images by using a pretrained character region awareness for text detection (CRAFT) model. The pretrained CRAFT model can detect text in these nine languages: Chinese, Japanese, Korean, Italian, English, French, Arabic, German, and Bangla (Indian).

Automated Visual Inspection: Use anomaly detection and classification techniques

These examples show how to use anomaly detection and classification techniques.

- The “Classify Defects on Wafer Maps Using Deep Learning” example classifies manufacturing defects on wafer maps.
- The “Detect Image Anomalies Using Explainable One-Class Classification Neural Network” example detects and localizes anomalies using single-class classification neural network.
- The “Detect Image Anomalies Using Pretrained ResNet-18 Feature Embeddings” example trains a similarity-based anomaly detector using one-class learning of feature embeddings extracted from a pretrained ResNet-18 convolutional neural network.

For more details, see “Getting Started with Anomaly Detection Using Deep Learning”.

Bounding Box Coordinates: Data augmentation for object detection using spatial coordinates

The `bboxresize`, `bboxcrop`, `bboxwarp`, and `showShape` functions assume the input bounding box coordinates for axis-aligned rectangles are specified in spatial coordinates and return the transformed bounding boxes in spatial coordinates.

Use non-integer coordinates to specify a bounding box using the `bboxresize`, `bboxcrop`, `bboxwarp`, or `showShape` function.

Functionality being removed or changed

anchorBoxLayer object being removed in future release
Still runs

The `anchorBoxLayer` object will be removed in a future release. Use `ssdObjectDetector` object to specify the anchor boxes for training a single shot detector (SSD) multibox object detection network, instead.

ssdLayers function being removed in future release
Still runs

The `ssdLayers` function will be removed in a future release. Use the `ssdObjectDetector` object to create a SSD multibox object detector, instead.

pixelLabelImageDatastore object being removed in future release
Errors

The `pixelLabelImageDatastore` will be removed in a future release. Use the `imageDatastore` and `pixelLabelDatastore` objects and the `combine` function instead.

Support for LayerGraph object as input to trainSSDObjectDetector function being removed in future release
Still runs

Starting in R2022a, use of `LayerGraph` object to specify SSD object detection network as input to the `trainSSDObjectDetector` function is not recommended and will be removed in a future release.

If your SSD object detection network is a `LayerGraph` object, configure the network as a `ssdObjectDetector` object by using the `ssdObjectDetector` function. Then, use the `ssdObjectDetector` object as input to the `trainSSDObjectDetector` function for training.

Experiment Manager Example: Find optimal training options

The **Experiment Manager** app enables you to create deep learning experiments to train object detectors under multiple initial conditions and compare the results. The “Train Object Detectors in Experiment Manager” example uses the **Experiment Manager** to find optimal training options for object detectors.

Multiclass Object Detection Example: Train multiclass object detector using deep learning

The “Multiclass Object Detection Using YOLO v2 Deep Learning” example shows you how to train a YOLO v2 multiclass object detector.

Datastore Support: Use datastores with ACF and cascade object detectors

Use datastores with the `trainACFObjectDetector` and `trainCascadeObjectDetector` functions, as well as with the detect object function of the `acfObjectDetector` object and the `step` object function of the `vision.CascadeObjectDetector` object

Structure from Motion and Visual SLAM

Stereo Vision Rectification Parameters: Access stereo camera rectification parameters

Use these enhancements to the `rectifyStereoImages` and `reconstructScene` functions to access stereo camera rectification parameters.

- Use the `rectifyStereoImages` function can now return a reprojection matrix, for reprojecting a 2-D point in a disparity map to a 3-D point. The function can also return the camera projection matrices for the rectified cameras from a a pair of stereo images. You can use these matrices to project 3-D world points from the coordinate system of the primary camera into the image plane of the rectified images.
- Use the `reconstructScene` function can now reconstruct a 3-D scene from a disparity map and a reprojection matrix.

Bundle Adjustment Data Management: Integration of bundle adjustment functions with data management objects

Use these bundle adjustment enhancements with data management objects:

- Use the `bundleAdjustment` function to refine world points in a `worldpointset` and to view poses in an `imageviewset`.
- Use the `bundleAdjustmentStructure` function to refine world points in a `worldpointset`.

Visual SLAM Example: Process image data from RGB-D camera to build dense map

The “Visual SLAM with an RGB-D Camera” example shows you how to process image data from an RGB-D camera to build a dense map of an indoor environment and estimate the trajectory of the camera. The example uses ORB-SLAM2, which is a feature-based vSLAM algorithm that supports RGB-D cameras.

Functionality Being Removed or Changed: Support for GPU removed for disparityBM function

The `disparityBM` function no longer supports GPU. You can use the enhanced `disparitySGM` function on the GPU instead. Results are not identical using the `disparitySGM` function, but the semi-global algorithm used in the `disparitySGM` function is normally recommended over the block-matching algorithm in the `disparityBM` function .

Point Cloud Processing

Point Cloud Preprocessing: Preserve organized structure of point cloud

Use the PreserveStructure name-value argument of the pcdownsample and pcdenoise functions to preserve the organized structure of a point cloud. Additionally, the pcdownsample function now returns the linear indices of points in the downsampled point cloud.

Velodyne Organized Point Cloud Support: Velodyne file reader can return organized point clouds

Use the OrganizePoints property of the velodyneFileReader object to return an organized point cloud.

Point Cloud: uint16 data type support for intensity

The pointCloud object now supports the uint16 data type for the Intensity property.

Code Generation, GPU, and Third-Party Support

OpenCV Interface: Integrate OpenCV version 4.5.0 projects with MATLAB

Integrate OpenCV projects with MATLAB using OpenCV version 4.5.0.

Generate C and C++ Code Using MATLAB Coder: Support for functions

These functions now supports portable C code generation for non-host target platforms.

- `pcdownsample` (only when using the grid average downsample method)
- `pcmerge`
- `pcregisterndt`

These functions and objects now support C/C++ code generation for both host and non-host target platforms.

- `pointTrack`
- `worldToImage`
- `scanContextLoopDetector`
- `pcregistericp`
- `pointsToWorld`
- `selectStrongestBboxMulticlass`
- `undistortPoints`
- `worldpointset`
- `bboxOverlapRatio`
- `copy` function of the `pointCloud` object.
- `pcviewset` object and its functions, excluding the `optimizePoses` function.
- `imageviewset` object and its functions, excluding the `optimizePoses`.
- The `optimizePoses` function of the `imageviewset` and `pcviewset` objects support code generation for host target platforms.
- `yolov3ObjectDetector` object and its `detect` function.
- `yolov4ObjectDetector` object and its `detect` function.

Generate CUDA code for NVIDIA GPU Coder: Support for functions

These functions now support code generation using GPU Coder™.

- `yolov3ObjectDetector` object and its `detect` function.
- `yolov4ObjectDetector` object and its `detect` function.

Functionality being removed or changed

GPU support removed for OpenCV interface with MATLAB

Errors

The MATLAB® functions in the Computer Vision Toolbox Interface for OpenCV in MATLAB support package no longer supports GPU.

Computer Vision with Simulink

Computer Vision with Simulink: Specify image data type in Simulink model

Prior to R2022a, Computer Vision Toolbox supported importing an image into a Simulink® model only as matrix data. In some cases, Simulink implemented the imported image as separate signals (for example, separate R, G, and B signals). In other cases, it was implemented as a single signal. Simulink did not have a universal standard to model the image data across Simulink blocks and MATLAB functions. You needed the Computer Vision Toolbox Interface for OpenCV in Simulink to import the images specified as `Simulink.ImageType` data type in your OpenCV code.

In R2022a, you can now use Computer Vision Toolbox to directly specify an image of the `Simulink.ImageType` data type and generate code for the model. Specify this data type for signals and other data in your model. The `Simulink.ImageType` data type is an encapsulated object that defines an image with fixed meta-attributes specific to this data type.

To specify a signal as the `Simulink.ImageType` data type for a supported block, on the **Signal Attributes** tab, specify **Data type**, or the equivalent parameter for the block, as `Simulink.ImageType(480,640,3,'ColorFormat','RGB','Layout','ColumnMajor','ClassUnderlying','uint8')`. This specification assigns a default `Simulink.ImageType` data type to the signal. You can also use the Data Type Assistant to customize the image attributes. For more information, see “Specify an Image Data Type” (Simulink).

To integrate images of the `Simulink.ImageType` data type in existing image processing algorithms that operate on matrix data only, use the new To Simulink Image and From Simulink Image blocks to convert the image data to and from the `Simulink.ImageType` data type, respectively. For more information, see “Track Marker Using Simulink Images”.

If your model includes image signals of the `Simulink.ImageType` data type, you can use the Color Space Conversion block to convert color information between color spaces by setting the **Conversion** parameter to `R'G'B' to intensity` or one of these new options:

- `R’G’B’ to B’G’R’`
- `B’G’R’ to R’G’B’`
- `B’G’R’ to intensity`

R2021b

Version: 10.1

New Features

Ground Truth Labeling

Labeler Enhancements: Labeling interactions and other enhancements

The following table describes enhancements for these labeling apps:

- **Image Labeler**
- **Video Labeler**
- **Ground Truth Labeler**
- **Lidar Labeler** (Lidar Toolbox)

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Show or hide labels and sublabels of type `Rectangle`, `Line`, `Polygon`, and `Projected cuboid` in a labeled image or video.	Yes	Yes	Yes	No
Show or hide labels of type `Cuboid` in a labeled point cloud or point cloud sequence.	No	No	Yes	Yes
View and edit cuboid ROI labels using top, side, and front 2-D view projections by selecting **Projected View**.	No	No	Yes	Yes
Segment ground from lidar data using the simple morphological filter (SMRF) algorithm. For more information about the algorithm parameters, see the `segmentGroundSMRF` (Lidar Toolbox) function.	No	No	Yes (only with Lidar Toolbox™ license)	Yes
Extract video scenes and corresponding labels from a `groundTruth` or `groundTruthMultisignal` object.	No	Yes	Yes	No
Digital Imaging and Communication in Medicine (DICOM) image format.	Yes	No	No	No

Feature Detection and Extraction

SIFT Feature Detector: Scale-invariant feature transform detection and feature extraction

Use the `detectSIFTFeatures` function to detect scale-invariant features. You can use the `SIFTPoints` object to store SIFT feature keypoints. In R2021b, the `extractFeatures` function now also has the SIFT descriptor.

Recognition, Object Detection, and Semantic Segmentation

Experiment Manager App Support: Track progress of deep learning object detector training

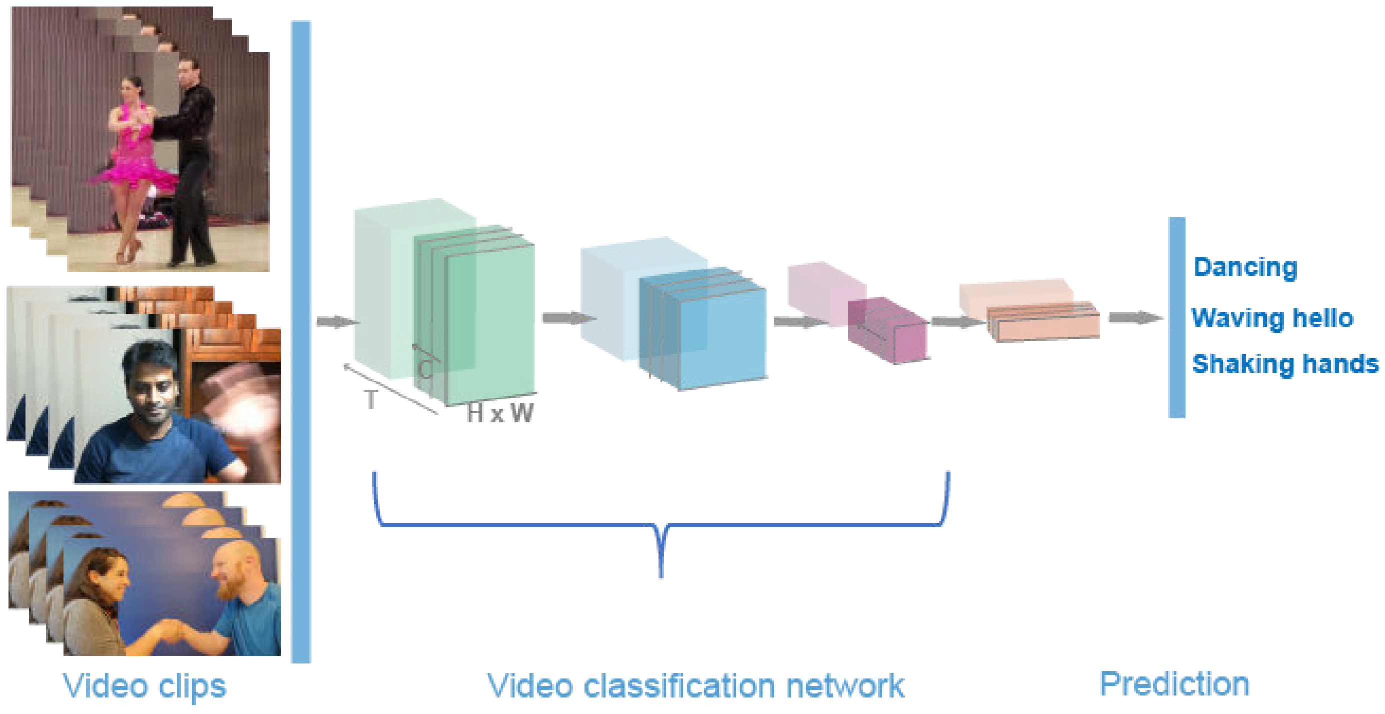
Use the **Experiment Manager** (Deep Learning Toolbox) app to track the progress of deep learning object detector training. Set the `ExperimentMonitor` name-value argument in the Computer Vision Toolbox training functions to specify an `experiments.Monitor` (Deep Learning Toolbox) object to use with the app. You can use the app with these training functions:

- `trainRCNNObjectDetector`
- `trainFastRCNNObjectDetector`
- `trainFasterRCNNObjectDetector`
- `trainSSDObjectDetector`
- `trainYOLOv2ObjectDetector`

Deep Learning Activity Recognition: Video classification using deep learning

Analyze, classify, and track activity contained in visual data sources, such as a video stream, using deep learning techniques. Use the Inflated-3D, SlowFast, or R(2+1)D video classifiers with their supporting software packages.

You can install software packages from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run these functionalities, you will require the Deep Learning Toolbox.



- The `inflated3dVideoClassifier` model contains two subnetworks: the video network and the optical flow network. These networks are trained on the Kinetics-400 data set with RGB data and optical flow data, respectively. This functionality requires the Computer Vision Toolbox Model for Inflated-3D Video Classification.
- The `slowFastVideoClassifier` model is pretrained on the Kinetics-400 data set which contains the residual network ResNet-50 model as the backbone architecture with slow and fast pathways. This functionality requires the Computer Vision Toolbox Model for SlowFast Video Classification.
- The `r2plus1dVideoClassifier` model is pretrained on the Kinetics-400 data set which contains 18 spatio-temporal (ST) residual layers. This functionality requires the Computer Vision Toolbox Model for R(2+1)D Video Classification.

For more details, see Getting Started with Video Classification Using Deep Learning.

Create Training Data for Video Classifier: Extract video clips for labeling and training workflow

Use the `sceneTimeRanges` and `writeVideoScenes` functions to extract video clips from labels from ground truth data.

Deep Learning Instance Segmentation: Create and configure pretrained Mask R-CNN neural networks

Create a Mask R-CNN instance segmentation object detector using the `maskrcnn` object. This network is trained on the COCO dataset with a ResNet-50 network as the feature extractor. You can detect objects in an image using the pretrained network or you can configure the network to perform transfer learning. This functionality requires the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation and Deep Learning Toolbox.

For more details, see Getting Started with Mask R-CNN for Instance Segmentation.

Deep Learning ROI Pooling: Nonquantized ROI pooling

Use the `roialign` function for nonquantized ROI pooling of `dlarray` (Deep Learning Toolbox) data.

Deep Learning Object Detector Block: Simulate and generate code for deep learning object detectors in Simulink

Simulate and generate code for deep learning object detectors in Simulink. The `Analysis & Enhancement` block library now includes the Deep Learning Object Detector block. This block predicts bounding boxes, class labels, and scores for the input image data by using a specified trained object detector. This block enables you to load a pretrained object detector into the Simulink model from a MAT file or from a MATLAB function.

For more information about working with the Deep Learning Object Detector block, see Lane and Vehicle Detection in Simulink Using Deep Learning (Deep Learning Toolbox). To learn more about generating code for Simulink models containing the Deep Learning Object Detector block, see Code Generation for a Deep Learning Simulink Model that Performs Lane and Vehicle Detection (GPU Coder).

Pretrained Deep Learning Models on GitHub: Perform object detection and segmentation using latest pretrained models on GitHub

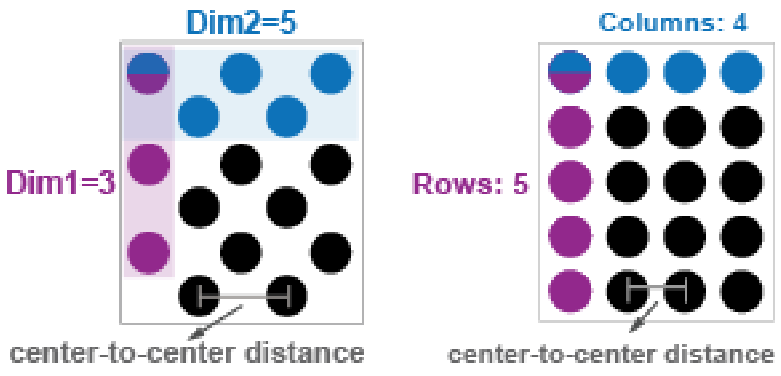
The MATLAB Deep Learning GitHub repository provides the latest pretrained deep learning networks to download and use for performing object detection and segmentation. For example:

- To perform object detection by using the pretrained you-only-look-once (YOLO) v4 deep learning network, see the Object Detection Using Pretrained YOLO v4 Deep Learning Network GitHub repository. You can download the model and use it to perform out-of-the-box inference.
- To perform scene text detection by using a pretrained character region awareness for text detection (CRAFT) model, see the Pretrained Character Region Awareness For Text Detection Model GitHub repository.
- To perform segmentation by using a pretrained DeepLabv3+ network, see the Pretrained DeepLabv3+ Semantic Segmentation Network GitHub repository.

Camera Calibration

Camera Calibration: Circle grid calibration pattern detection

Support is available for asymmetric and symmetric circle grid patterns in the **Camera Calibrator** and **Stereo Camera Calibrator** apps or programmatically when you use the `detectCircleGridPoints` and `generateCircleGridPoints` functions.



For more information about camera calibration, using the calibration app, and circle grid patterns, see Using the Single Camera Calibrator App, Select Calibration Pattern and Set Properties, and Calibration Patterns.

Camera Calibration: Custom pattern detection

The **Camera Calibrator** and the **Stereo Camera Calibrator** apps support calibration using custom patterns. You can create a custom detector and load it into the app as one of the selectable pattern choices. To see how to use a custom calibration pattern, see the Camera Calibration Using AprilTag Markers example.

Rigid 3-D Support: Pass rigid 3-D transformation object to calibration functions

Pass a `rigid3d` object as an input to the `cameraPoseToExtrinsics` and `extrinsicsToCameraPose` functions to return the output transformations as `rigid3d` objects.

OpenCV Camera Parameters: Relay camera intrinsics and stereo parameters to and from OpenCV

Use OpenCV camera intrinsics and stereo parameters with Computer Vision Toolbox by using these functions:

- `cameraIntrinsicsFromOpenCV`
- `cameraIntrinsicsToOpenCV`
- `stereoParametersFromOpenCV`
- `stereoParametersToOpenCV`

Structure from Motion and Visual SLAM

Bundle Adjustment Solver: Specify optimization solver

Use the `Solver` name-value argument of the `bundleAdjustment` function to select between the `preconditioned-conjugate-gradient` solver for high sparsity, and the `sparse-linear-algebra` solver. High sparsity indicates that each camera view observes only a small selection of world points.

Rigid 3-D Support: Pass 3-D rigid transformation object to camera parameter functions

Pass a `rigid3d` object as an input to the `pointsToWorld`, `cameraMatrix`, `stereoParameters`, and `worldToImage` functions and objects.

Image View Set Support: Find views and view connections

Use the `findView` and `findConnection` object functions of the `imageviewset` object to find views and connections in image view sets.

Bag of Features: Support for binary features

You can now use the custom feature extraction function `extractorFcn` to return a `binaryFeatures` object.

Use the `TreeProperties` property of the `bagOfFeatures` object to control the amount of vocabulary in the tree at successive levels.

The `TreeProperties` property replaces the `VocabularySize` property of the `bagOfFeatures` object. `VocabularySize` continues to work.

Bag of Features Search Index: Support for Visual Simultaneous Localization and Mapping (vSLAM) loop closure detection

Enhancements have been made to the `invertedImageIndex` object and `retrieveImages` to support using bag-of-features for loop closure detection.

- Use the `addImages` object function of the `invertedImageIndex` object to add an image to the image index with an image identifier.
- Use the `addImageFeatures` object function of the `invertedImageIndex` to add the features of an image to the image index with an image identifier.
- Use the `Metric` name-value argument of `retrieveImages` to set the similarity metric for ranking image retrieval results.

Point Cloud Processing

Point Cloud Simultaneous Localization and Mapping (SLAM): Detect loop closures

Use the `scanContextLoopDetector` object to manage the database of scan context descriptors and detect loop closures. For details, see Implement Point Cloud SLAM in MATLAB.

Point Cloud View Set Support: Find views and view connections

Use the `findView` and `findConnection` object functions of the `pcviewset` object to find views and connections in point cloud view sets.

Multiquery Radius Search: Optimized radius search for point cloud segmentation

Use the `ParallelNeighborSearch` name-value argument of the `pcsegdist` function, to improve segmentation speed.

Point Cloud Viewers: Modify background color programmatically

Use the `BackgroundColor` name-value argument to modify the axis, figure, and title background colors for the `pcshow`, `pcshowpair`, and `pcplayer` functions.

Code Generation, GPU, and Third-Party Support

Generate C and C++ Code Using MATLAB Coder: Support for functions

The `readAprilTag` and `readBarcode` functions now support code generation for host target platforms.

These functions now support code generation for both host and nonhost target platforms:

- `fisheyeIntrinsics`
- `fisheyeParameters`
- `pcalign`

Generate C and C++ Code Using MATLAB Coder: Compiler links to OpenCV libraries

You can generate C code for your target by using any of the supported functions. The support enables you to build the application for your target using a C++ compiler. The C++ compiler links to OpenCV libraries that you provide for the particular target. You can also build a standalone application by using the `packNGo` function and setting the `packType` name-value argument to `"hierarchical"`.

Generate CUDA code for NVIDIA GPUs using GPU Coder

These Computer Vision Toolbox functions now support code generation using the GPU Coder:

- `pcnormals`
- `pctransform`
- `pcdenoise`
- `pcfitplane`
- `pcmapndt`
- `pcmerge`

Computer Vision Toolbox Interface for OpenCV in MATLAB (September 2021, Version 21.2): Call OpenCV functions from MATLAB

Computer Vision Toolbox Interface for OpenCV in MATLAB provides a prebuilt MATLAB interface to OpenCV library. You can directly call OpenCV functions from MATLAB without having to write a C++ MEX file. The prebuilt interface provides these utility functions for moving data between OpenCV and MATLAB:

- `createMat`
- `createUMat`
- `getBasePtr`
- `getImage`
- `keyPointsToStruct`
- `rectToBbox`

Computer Vision Toolbox Interface for OpenCV in Simulink: Specify image data type in Simulink model

Prior to R2021a, the Computer Vision Toolbox Interface for OpenCV in Simulink only supported importing an image into a Simulink model as matrix data. In some cases, Simulink implemented the imported image as separate signals (for example, separate R, G, and B signals). In other cases, it was implemented as a single signal. Simulink did not have a universal standard to model the image data across Simulink blocks and MATLAB functions.

In R2021b, you can now use the Computer Vision Toolbox Interface for OpenCV in Simulink to specify an image of the `Simulink.ImageType` data type and generate code for the model. Specify this data type for signals and other data in the model. The `Simulink.ImageType` data type is an encapsulated object that defines an image with fixed meta-attributes specific to this data type.

When importing OpenCV code into a Simulink model, you can use the **OpenCV Importer** app to configure default values for the `Simulink.ImageType` data type. Specify these configurations on the Create Simulink Library page:

- Select **Configure library to use Simulink.ImageType signals**.
- Specify **Default Color Format of Simulink.ImageType signal** as `BGR`, `RGB`, or `Grayscale`.
- Specify **Default Array layout of Simulink.ImageType signal** as `Row-major` or `Column-major`.

The **OpenCV Importer** app generates library and ToOpenCV and FromOpenCV blocks inside the subsystem and configures them to use the `Simulink.ImageType` data type.

To integrate images of `Simulink.ImageType` data type in existing image processing algorithms that operate on matrix data only, use the new blocks Image To Matrix and Matrix To Image to convert the image data. For more information, see Convert Between Simulink Image Type and Matrices.

You can configure the From Multimedia File block to generate signals of the `Simulink.ImageType` data type. When the **Output color format** parameter is set to `RGB` or `Intensity`, specify the **Image signal** parameter as `Simulink image signal`.

The Video Viewer block supports visualization for `Simulink.ImageType` signals.

These blocks support simulation and code generation of a `Simulink.ImageType` object:

Block Library	Block Name
Sources	• Ground • Inport • Outport

Block Library	Block Name
Signal Routing	• Goto • From • Data Store Read • Data Store Write • Data Store Memory • Switch • Multiport Switch • Merge • Variant Source • Variant Merge (internal block added during code generation) • Mux • Demux • Vector Concatenate, Matrix Concatenate • Selector
Sink	• Terminator
Ports & Subsystems	• Subsystem, Atomic Subsystem, CodeReuse Subsystem • Enabled Subsystem • Triggered Subsystem • Function-Call Subsystem • If • Switch Case • Resettable Subsystem • For Iterator Subsystem • Model
Discrete	• Unit Delay
Signal Attributes	• Signal Conversion • Signal Specification
User-Defined Functions	• Initialize Function, Reset Function, and Terminate Function • Simulink Function block with side I/O and arguments

R2021a

Camera Calibration: Checkerboard detector for fisheye camera calibration and partial checkerboard detection

- Calibrate cameras with up to a 195 degree field of view (FOV) using the **Camera Calibrator** app. Perform programmatic calibration of wide field of view cameras by enabling the `HighDistortion` name-value argument of the `detectCheckerboardPoints` function.
- All functions in the single camera calibration workflow now support partially detected checkerboards: `detectCheckerboardPoints`, `estimateCameraParameters`, `estimateFisheyeParameters`, and `showExtrinsics`. Perform detection of partial checkerboards by enabling the `PartialDetections` name-value argument of the `detectCheckerboardPoints` function.

Visual Simultaneous Localization and Mapping (SLAM): Enhancements to the SLAM workflow

The visual SLAM and structure from motion (SfM) workflow includes these enhancements:

- The `matchFeaturesInRadius` function returns, the indices of features, most likely to correspond between two input feature matrices, within the specified spatial constraints.
- The `optimizePoses` function now supports 3-D similarity transformations within `imageviewset` objects.
- The `worldToImage` function can now return the indices of the points within a boundary in an image.
- Use the `minNumMatches` argument of the `connectedViews` function to select strongly connected views.

For more details, see Visual SLAM Overview.

The new Stereo Visual Simultaneous Localization and Mapping example shows you how to process image data from a stereo camera to build a map of an outdoor environment and estimate the trajectory of the camera. The example uses ORB-SLAM2 , which is a feature-based vSLAM algorithm supporting stereo cameras.

Point Cloud Simultaneous Localization and Mapping (SLAM): Support for point cloud SLAM with NDT map representation

Use the `pcmapndt` object to create a normal distribution transform (NDT) map from a prebuilt point cloud map of the environment. The NDT map is a compressed, memory-efficient representation, suitable for point cloud localization in SLAM. For more details, see Point Cloud SLAM Overview.

Point Cloud Processing: Set cluster density limits, get indices of bins, and registration enhancements

The point cloud segmentation and registration workflow includes these enhancements:

- New `NumClusterPoints` name-value argument for `pcsegdist` and `segmentLidarData` functions to specify the minimum and maximum number of points in each point cloud cluster.
- The `pcbin` function returns bin indices as a vector or a matrix using the `'BinOutput'` argument.

- The `pcregistercorr` function can now measure the quality of a registration as the peak correlation value of the phase difference between two occupancy grids. Use the `'Window'` name-value argument to increase the stability of the registration results.

Deep Learning Training: Pass training options to detectors, perform cutout data augmentation, and balance labels of blocked images

Computer Vision Toolbox training functions for deep learning detectors support the `'OutputFcn'` (Deep Learning Toolbox) option of the `trainingOptions` (Deep Learning Toolbox) function. Use this option to display or plot progress information, or to stop training.

The `bboxerase` function removes image data and bounding boxes that lie within a specified region of interest (ROI). Use the `bboxerase` function to update the bounding box information while you perform cutout augmentation of training data.

The `balanceBoxLabels` and `balancePixelLabels` functions now accept a collection of large images specified as `blockedImage` objects.

Semantic Segmentation Enhancements: Support for dlnetwork objects and specified classes

The `semanticseg` function now accepts networks specified as a `dlnetwork` (Deep Learning Toolbox) object. You can also specify the classes into which pixels or voxels are classified by using the '`Classes`' name-value argument of the `semanticseg` function.

Evaluate Image Segmentation: Calculate generalized Dice similarity coefficient

Use the `generalizedDice` function to compute a generalized Dice similarity coefficient for two segmented images.

Labeler Enhancements: Instance segmentation labeling, super pixel flood fill, labeling large images, and additional features

The following table describes enhancements for these labeling apps:

- **Image Labeler**
- **Video Labeler**
- **Ground Truth Labeler**
- **Lidar Labeler** (Lidar Toolbox)

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Label distinct instances of objects belonging to the same class using a polygon label. For more details, see Label Objects Using Polygons.	Yes	Yes	Yes	No

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Use superpixel automation to quickly pixel label regions of an image with similar pixel values. For more details, see Label Pixels Using Superpixel Tool.	Yes	Yes	Yes	No
Automate the labeling of multiple signals together within a single automation run. For an example, see Automate Ground Truth Labeling Across Multiple Signals.	No	No	Yes	No
Label very large images (with at least one dimension <8K) that previously could not be loaded into memory. Load these images as blocked images. For more details, see Label Large Images in Image Labeler.	Yes	No	No	No
Use a custom reader function to import any point cloud. For more details, see Use Custom Point Cloud Source Reader for Labeling (Lidar Toolbox).	No	No	No	Yes
Define and view a region of interest (ROI) in the point cloud and label objects in it. For more details, see ROI View (Lidar Toolbox).	No	No	No	Yes
Control the point dimension of the point cloud.	No	No	No	Yes

YOLO v3 Object Detector: Computer Vision Toolbox Model for YOLO v3 Object Detection (Version 21.1.0)

Perform object detection with you only look once version 3 (YOLO v3) deep learning networks by using the functions in the Computer Vision Toolbox Model for YOLO v3 Object Detection support package.

The `yolov3ObjectDetector` object creates a YOLO v3 object detector to detect objects in images. You can either create a pretrained YOLO v3 object detector or a custom YOLO v3 object detector.

Create Pretrained YOLO v3 Object Detector

- To create a pretrained YOLO v3 object detector, use the pretrained DarkNet-53 and tiny YOLO v3 deep learning networks included in the support package. These networks are trained on the COCO dataset.

Create Custom YOLO v3 Object Detector

- You can create a custom YOLO v3 object detector by using an untrained or pretrained YOLO v3 deep learning network.

- You can also create a custom YOLO v3 object detector by using a base network for feature extraction. The `yolov3ObjectDetector` object adds detection heads to the feature extraction layers of the base network and creates a YOLO v3 object detector. You must specify the source layers to which to add the detection heads.

  You must specify the anchor boxes and the class names to use to train the YOLO v3 object detector on a custom dataset.

  The `yolov3ObjectDetector` object configures the detector for transfer learning if you use a pretrained deep learning network and specify the new anchor boxes and class names.

  The Object Detection Using YOLO v3 Deep Learning example shows how to create and train a custom YOLO v3 object detector for performing object detection.

Use the `yolov3ObjectDetector` object functions for training and inference workflows:

- The `detect` function performs object detection using the pretrained YOLO v3 object detector and a test image.
- The `preprocess` function resizes the training data and the test data to the nearest network input size and rescales the intensity values to the range [0, 1]. You can use this function to preprocess the input images before training and detection.
- The `forward` function computes the network outputs during forward pass while training the network. You can use the network outputs to model the gradient losses.
- The `predict` function computes the network outputs for inference.

You can download the Computer Vision Toolbox Model for YOLO v3 Object Detection from the Add-On Explorer. For more information, see Get and Manage Add-Ons.

Extended Capability: Perform GPU and C/C++ code generation

Generate C and C++ Code Using MATLAB Coder

Computer Vision Toolbox functions now support code generation in host and nonhost target platform:

`pccat`
`scanContextDescriptor`
`scanContextDistance`
`detect` of `acfObjectDetector`

You can generate C code for your target by using any of the supported functions. The support enables you to build the application for your target using a C++ compiler. The C++ compiler links to OpenCV libraries that you provide for the particular target. You can also build a standalone application by using the `packNGo` function and using the `'packType'` name-value pair with a value of `'hierarchical'`.

Generate CUDA code for NVIDIA GPUs using GPU Coder

These Computer Vision Toolbox functions now supports code generation using the GPU Coder:

`pcbin`

pcregisterndt
segmentGroundFromLidarData

Functionality being removed or changed

GPU support for disparityBM, detectFASTFeatures, and the interface for OpenCV in MATLAB will be removed in a future release

Still runs

GPU support will be removed in a future release for:

- The `detectFASTFeatures` and `disparityBM` functions.
- The Computer Vision Toolbox OpenCV Interface in MATLAB support package. For more details see Install and Use Computer Vision Toolbox Interface for OpenCV in MATLAB.

Computer Vision Toolbox Support Package for Xilinx Zynq-Based Hardware has been moved to Vision HDL Toolbox Support Package for Xilinx Zynq-Based Hardware

Behavior change

Starting in R2021a, the Computer Vision Toolbox Support Package for Xilinx® Zynq®-Based Hardware has been renamed to Vision HDL Toolbox™ Support Package for Xilinx Zynq-Based Hardware. To use this support package in R2021a, you must have Vision HDL Toolbox installed. See Vision HDL Toolbox .

R2020b

Mask R-CNN: Train Mask R-CNN networks for instance segmentation using deep learning

Use these features to support a Mask R-CNN network:

- Use the `roiAlignLayer` object to output fixed-size feature maps for every rectangular ROI within an input feature map.
- Use the `insertObjectMask` object to insert masks in an image or video stream.

Visual SLAM Data Management: Manage 3-D world points and projection correspondences to 2-D image points

Use the `worldpointset` object to store correspondences between 3-D world points and 2-D image points across camera views.

AprilTag Pose Estimation: Detect and estimate pose for AprilTags in an image

The `readAprilTag` function detects AprilTags in an image and returns the encoded ID, tag family, and estimated pose for the tag in the scene with respect to the camera. For details on how to use this function for camera calibration, see the Camera Calibration Using AprilTag Markers example.

Point Cloud Registration: Register point clouds using phase correlation

Use the `pcregistercorr` function to register two point clouds using a phase correlation algorithm. The function performs registration on the occupancy grids of the two point clouds.

Use the `normalRotation` function to correct the ground plane such that it is parallel to the XY-plane and has a normal vector of `[0 0 1]`.

Point Cloud Loop Closure Detection: Compute Point cloud feature descriptor for loop closure detection

Use the `scanContextDescriptor` function to compute a 2-D global feature descriptor that captures the distinctiveness of a view from point cloud scans.

Use the `scanContextDistance` function to compute the distance between two scan context descriptors.

Use the `pcalign` function to align an array of point clouds and the `pccat` function to concatenate an array of point clouds.

Triangulation Accuracy Improvements: Filter triangulated 3-D world points behind camera view

Use the `validIndex` output argument of the `triangulate` and `triangulateMultiview` functions to filter and remove triangulated 3-D world points that appear behind the camera.

Geometric Transforms: Estimate 2-D and 3-D geometric transformations from matching point pairs

Use the `estimateGeometricTransform2D` function to estimate 2-D rigid, affine, similarity, and projective transformations.

Use the `estimateGeometricTransform3D` function to estimate 3-D rigid and similarity transformations.

Labeler Enhancements: Label objects in images and video using projected 3-D bounding boxes, load custom image formats, use additional keyboard shortcuts, and more

This table describes enhancements for these labeling apps:

- **Image Labeler**
- **Video Labeler**
- **Ground Truth Labeler**
- **Lidar Labeler** — Introduced in R2020b

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Load images with custom image formats using an `imageDatastore` object	Supported	Not supported	Not supported	Not supported
Draw projected 3-D bounding boxes around objects in images and video using the projected cuboid label type	Supported	Supported	Supported	Not supported
Delete pixel labels	Supported	Supported	Supported	Not supported
Undo and redo drawing a pixel label an increased number of times	Supported	Supported	Supported	Not supported
Use keyboard shortcuts for selecting drawn labels and resizing bounding boxes	Supported	Supported	Supported	Not supported
Specify attributes for cuboid ROI labels	Not supported	Not supported	Supported	Supported

Enhancement	Image Labeler	Video Labeler	Ground Truth Labeler	Lidar Labeler
Visualize point cloud clusters across all frames, not just individual frames, when **Snap to Cluster** option is selected, by using a new **Cluster Settings** option	Not supported	Not supported	Supported	Supported
Use keyboard shortcuts for panning across the point cloud frame and moving multiple selected cuboids	Not supported	Not supported	Supported	Supported

Object Detection Visualizations: Visualize shapes on images and point clouds

Use the `showShape` function to highlight detected objects in an image or point cloud using shapes such as a rectangles, cuboids, polygons, and circles.

Use the `insertObjectMask` function to insert a color mask of segmented shapes into an image or video.

Evaluate Pixel-Level Segmentation: Compute a confusion matrix of multiclass pixel-level image segmentation

Use the `segmentationConfusionMatrix` function to compute a confusion matrix from the predicted pixel labels and ground truth pixel labels from labeled images.

Focal Loss Layer Improvements: Add a focal loss layer to a semantic segmentation or image classification deep learning network

You can now use the `focalLossLayer` object to add a focal loss layer to a semantic segmentation or image classification deep learning network. Use the focal loss layer to train a deep learning network on class-imbalanced data sets.

focalCrossEntropy function: Compute focal cross-entropy loss in custom training loops

Use the `focalCrossEntropy` function to compute the focal cross-entropy loss in order to train a custom deep learning network on class-imbalanced datasets.

Computer Vision Examples: Explore deep learning workflows, explore camera calibration using AprilTags, and compute segmentation metrics

Explore these Computer Vision Toolbox examples:

- The Estimate Body Pose Using Deep Learning example shows how to estimate the body pose of one or more people using the OpenPose algorithm and a pretrained network.
- The Generate Image from Segmentation Map Using Deep Learning example shows how to generate a synthetic image of a scene from a semantic segmentation map using a Pix2PixHD conditional generative adversarial network (CGAN).
- The Activity Recognition from Video and Optical Flow Data Using Deep Learning example shows how to train an Inflated-3D (I3D) two-stream convolutional neural network for activity recognition using RGB and optical flow data from videos.
- The Camera Calibration Using AprilTag Markers example shows how to detect and localize AprilTags in a calibration pattern.

Extended Capability: Perform GPU and C/C++ code generation

Generate C and C++ Code Using MATLAB Coder

These Computer Vision Toolbox functions now support code generation in host and nonhost target platform:

- `ransac`
- `pcbin`

You can generate C code for your target by using either of the supported functions. The support enables you to build the application for your target using a C++ compiler. The C++ compiler links to OpenCV libraries that you provide for the particular target. You can also build a standalone application by using the `packNGo` function and using the `'packType'` name-value pair with a value of `'hierarchical'`.

Generate CUDA code for NVIDIA GPUs using GPU Coder

The `pcsegdist` function now supports code generation using the GPU Coder.

OpenCV Interface: Integrate OpenCV version 4.2.0 projects with MATLAB

Integrate OpenCV projects with MATLAB using OpenCV version 4.2.0.

Functionality being removed or changed

focalLossLayer input arguments, alpha and gamma now have default values
Still runs

The `focalLossLayer` object's input arguments `alpha` and `gamma` are no longer required to specify the balancing and focusing parameter values, respectively. These properties now have default values:

- `Alpha`: `0.25`
- `Gamma`: `2.0`

To change the balancing value, set the `Alpha` property. To change the focusing value, set the `Gamma` property. For example, `focalLossLayer('Alpha',0.7,'Gamma',1)` sets the `Alpha` and `Gamma` properties to `0.7` and `1`, respectively, upon creation of the object.

yolov2ReorgLayer will be removed
Still runs

The `yolov2ReorgLayer` function will be removed in a future release. Use the `spaceToDepthLayer` function to add a reorganization layer to the YOLO v2 deep learning network.

estimateGeometricTransform will be removed
Still runs

The `estimateGeometricTransform` will be removed in a future release. Use the `estimateGeometricTransform2D` function instead.

R2020a

Point Cloud Deep Learning: Detect and classify objects in 3-D point clouds

Use the `boxLabelDatastore` object with cuboid bounding box support. Preprocess point cloud data using the `pcbin` function.

To build and evaluate point cloud based object detectors, use these functions, which now support rotated rectangle box formats.

- `pcbin`
- `bboxwarp`
- `bboxcrop`
- `bboxresize`
- `bboxOverlapRatio`
- `selectStrongestBbox`
- `selectStrongestBboxMulticlass`
- `bboxPrecisionRecall`
- `evaluateDetectionAOS`

The Point Cloud Classification Using PointNet Deep Learning example trains a PointNet network for point cloud classification.

Deep Learning with Big Images: Train and use deep learning object detectors and semantic segmentation networks on very large images

Balance and store big image data by using these added features.

- The `balanceBoxLabels` function balances the distribution of detector training data from a collection of very large images.
- The `balancePixelLabels` function balances the distribution of pixel labeled training data from a collection of very large images.
- The `boxLabelDatastore` and `blockLocationSet` objects load multiple blocks of data for training and evaluation.

Simultaneous Localization and Mapping (SLAM): Perform point cloud and visual SLAM

Use these objects, functions, and properties to manage SLAM and point cloud processing.

- The `imageviewset` object manages visual odometry and structure from motion (SfM) data.
- The `pcviewset` object manages point cloud odometry data.
- The `rigid3d` object stores a 3-D rigid transformation. You can use the `rigid3d` object with point cloud processing functions like `pctransform` and `pcregisterndt`.
- The `optimizePoses` function optimizes absolute poses using relative pose constraints.

  Specify an absolute pose as a `rigid3d` object in the `plotCamera` and `triangulateMultiview` functions. You can use the '`AbsolutePose`' name-value pair argument instead of the combination of the '`Location`' and '`Orientation`' name-value pairs.

- Refine camera poses using the `bundleAdjustmentMotion` function. Refine 3-D points using the `bundleAdjustmentStructure` function.

The Monocular Visual Simultaneous Localization and Mapping example processes image data from a monocular camera to build a map of an indoor environment and estimate the trajectory of the camera using ORB-SLAM, a feature-based vSLAM algorithm.

Bar Code Reader: Detect and decode 1-D and 2-D barcodes

Read linear (1-D) and matrix (2-D) barcodes using the `readBarcode` function.

The Localize and Read Multiple Barcodes in Image example demonstrates preprocessing steps that can be used to improve the detection of 1-D and 2-D barcodes in an image.

SSD Object Detection: Detect objects in images using a single shot multibox object detector (SSD)

Use these functions, objects, and layers to detect objects in images using an SSD object detector.

- The `trainSSDObjectDetector` function trains a deep learning SSD object detector.
- The `ssdObjectDetector` object detects objects using the SSD-based detector.
- The `ssdLayers` function creates an SSD object detection network.
- The `anchorBoxLayer` layer stores anchor boxes for object detection.
- The `focalLossLayer` classification layer using focal loss for object detection.
- The `ssdMergeLayer` layer merges activations from several feature maps.

The Object Detection Using SSD Deep Learning example trains a single shot object detector using a deep learning network architecture.

The Code Generation for Object Detection by Using Single Shot Multibox Detector example generates CUDA code for an SSD network.

Velodyne Point Cloud Reader: Store start time for each point cloud frame

The `velodyneFileReader` object has a new property named `Timestamps`. The `Timestamps` property stores the start time for each point cloud frame in the input sequence.

Labelers: Rename scene labels, select ROI color, and show ROI label names

The **Video Labeler** and the **Image Labeler** apps now support these features.

- Rename scene labels
- Set custom colors for ROIs
- Hover over an ROI to display its label name

Validate Deep Learning Networks: Specify training options to validate deep learning networks during training

The `trainFastRCNNObjectDetector`, `trainFasterRCNNObjectDetector`, `trainSSDObjectDetector`, and `trainYOLOv2ObjectDetector` functions now support validation during training. Specify validation options for network training by using the input argument `options`.

- The value of `options` must be a `TrainingOptionsADAM`, `TrainingOptionsRMSProp`, or `TrainingOptionsSGDM` object returned by the `trainingOptions` function.
- Use the name-value pair arguments `'ValidationData'`, `'ValidationFrequency'`, and `'ValidationPatience'` of a `trainingOptions` function to set validation options for network training.

YOLO v2 Enhancements: Import and export pretrained YOLO v2 object detectors

Create a you-only-look-once (YOLO) v2 object detector from a deep learning framework, such as ONNX™ or Keras. Import the pretrained network using the `network` input to the `yolov2ObjectDetector` object. The Import Pretrained ONNX YOLO v2 Object Detector example shows how to import a pretrained YOLO v2 network in ONNX model format to `yolov2ObjectDetector` object and perform object detection.

Additionally, you can create a YOLO v2 object detector that was trained with the `trainYOLOv2ObjectDetector` function, and then export it to ONNX using the `exportONNXNetwork` function. The Export YOLO v2 Object Detector to ONNX example shows how to:

- Export a YOLO v2 object detector to ONNX model format.
- Perform object detection using the exported YOLO v2 network in ONNX model format.

You can now specify `Classes` as input to the `yolov2OutputLayer` function. Use the name-value pair argument `'Classes'` to input the object classes in training data to the output layer.

YOLO v3 Deep Learning: Perform object detection using YOLO v3 deep learning network

The Object Detection Using YOLO v3 Deep Learning example shows how to design, train, and use a YOLO v3 network for object detection.

Computer Vision Examples: Explore object detection with deep learning workflows, structure from motion, and point cloud processing

- The Object Detection Using SSD Deep Learning example trains a single shot object detector using a deep learning network architecture.
- The Code Generation for Object Detection by Using Single Shot Multibox Detector example generates CUDA code for an SSD network.
- The Point Cloud Classification Using PointNet Deep Learning example trains a PointNet network for point cloud classification.

- The Monocular Visual Simultaneous Localization and Mapping example processes image data from a monocular camera to build a map of an indoor environment and estimate the trajectory of the camera using ORB-SLAM, a feature-based vSLAM algorithm.
- The Import Pretrained ONNX YOLO v2 Object Detector example imports a pretrained YOLO v2 object detector from an ONNX deep learning framework.
- The Export YOLO v2 Object Detector to ONNX example exports a pretrained YOLO v2 object detector to an ONNX deep learning framework.
- The Object Detection Using YOLO v3 Deep Learning example shows how to design, train, and use a YOLO v3 network for object detection.
- The Localize and Read Multiple Barcodes in Image example demonstrates preprocessing steps that can be used to improve the detection of 1-D and 2-D barcodes in an image.

Code Generation: Generate C/C++ code using MATLAB Coder

More Computer Vision Toolbox functions and objects now support portable C code generation. The `segmentGroundFromLidarData` function now supports code generation in host and nonhost target platforms. These lidar, point cloud, and tracking functions now support code generation in nonhost platforms:

Lidar and Point Cloud Processing
`pcdenoise`
`pcdownsample`
`pcnormals`
`pcmerge`
`pcsegdist`
`segmentLidarData`
`pctransform`
`pcregistercpd`
`pcregisterndt`
`pcfitcylinder`
`pcfitsphere`
`pcfitplane`
Tracking and Motion Estimation
`insertShape`
`insertMarker`

You can generate C code for your specific target by using any of the supported functions. The support enables you to build the application for your target using a C++ compiler. The C++ compiler links to OpenCV libraries that you provide for the particular target. You can also build a standalone application by using the `packNGo` function and setting the '`packType`' name-value pair to `'hierarchical'`.

Computer Vision Toolbox Interface for OpenCV in Simulink: Import OpenCV code into Simulink

The Computer Vision Toolbox Interface for OpenCV in Simulink support package enables you to import OpenCV code into a Simulink model. To install the support package, first click **Add-Ons** on the MATLAB **Home** tab. In the **Add-On** Explorer window, find and click the support package, and then click **Install**. This support package requires Computer Vision Toolbox. After installing the support package, you can import your OpenCV code and create Simulink library by using the **OpenCV Importer** app. The importer uses two OpenCV conversion blocks ToOpenCV and FromOpenCV. You can generate C++ code from the created Simulink model and deploy the code into your target hardware. For more information, see Install and Use Computer Vision Toolbox OpenCV Interface for Simulink.

Functionality being removed or changed

pcregisterndt, pcregistericp, and the pcregistercpd functions return a rigid3d object

Behavior change

The `pcregisterndt` and `pcregistericp` functions now return a `rigid3d` object as their rigid transformation output. Previously, the functions returned an `affine3d` object.

The `pcregistercpd` function can return a `rigid3d` object, an `affine3d` object, or a displacement field as its transformation object.

New imageviewset replaces viewSet

Use the `imageviewset` object in place of the `viewSet` object for managing data for structure from motion, visual odometry, and visual SLAM workflows.

R2019b

Video and Image Labeler: Copy and paste pixel labels, improved pan and zoom, improved frame navigation, and line ROI, label attributes, and sublabels added to Image Labeler

The **Image Labeler** and **Video Labeler** apps now support these features:

Feature	Video Labeler	Image Labeler
Copy and paste pixel labels	New	New
Pan and zoom more easily within the labeling window	New	New
Create a line region of interest (ROI)	Introduced in R2018b	New
Create label attributes and sublabels	Introduced in R2018b	New
Use scrubber for video tracking	New	Not supported
Click on Visual Summary timeline to go to corresponding (timestamp) frame	New	New

Data Augmentation for Object Detectors: Transform image and bounding box

Use bounding box transformations and datastore support for deep learning workflows.

- The `bboxresize`, `bboxwarp`, and `bboxcrop` functions support bounding box transformations.
- The `boxLabelDatastore` creates a datastore for bounding box label data. The object can contain different tables for labeled bounding boxes of various classes.
- The `detect` object functions for the `trainFastRCNNObjectDetector`, `trainFasterRCNNObjectDetector`, and `trainYOLOv2ObjectDetector` object trainers now support the use of a datastore.
- The `evaluateDetectionPrecision` and `evaluateDetectionMissRate` functions now support the use of a datastore.

Semantic Segmentation: Classify individual pixels in images and 3-D volumes using DeepLab v3+ and 3-D U-Net networks.

Create convolutional neural networks with added layer and datastore support:

- The `dicePixelClassificationLayer` layer creates a pixel classification layer by using generalized dice loss for semantic segmentation.
- The `deeplabv3plusLayers` function creates a DeepLab v3+ convolutional neural network for semantic segmentation.
- The `unet3dLayers` function creates a 3-D U-Net convolutional neural network for semantic segmentation of volumetric images.
- The `unetLayers` function now supports a padding style for convolution layers in the encoder and the decoder subnetworks. Use the `'ConvolutionPadding'` name-value pair to specify the padding.

- The `semanticseg` and the `evaluateSemanticSegmentation` functions now support use of a datastore.

Deep Learning Object Detection: Perform faster R-CNN end-to-end training, anchor box estimation, and use multichannel image data

Enhancements to training deep learning object detectors functions and new faster R-CNN layer:

- Create network architecture for Faster R-CNN by using the `fasterRCNNlayers` function.
- Use the `trainFastRCNNObjectDetector` function end-to-end training method to train a Fast R-CNN or Faster R-CNN detector.
- Use datastores with the `trainFastRCNNObjectDetector` and `trainFastRCNNObjectDetector` functions.
- Use the `estimateAnchorBoxes` function to automatically estimate anchor boxes based on a training data set and a k-means clustering algorithm.
- Use a multichannel image to train an R-CNN, Fast R-CNN, or Faster R-CNN detector. You can still use grayscale or RGB images as well.

Version History

Starting in R2019b, by default, the `trainFasterRCNNObjectDetector` function uses the end-to-end method for training a detector.

In previous releases, the default training method used the four-step method. To preserve compatibility, set the `TrainingMethod` property to `'four-step'`.

Deep Learning Acceleration: Optimize YOLO v2 and semantic segmentation using MEX acceleration

Computer Vision Toolbox now supports performance optimization in both CPU and GPU execution environments for these functions.

- The `detect` function for YOLO v2 object detection. Use the '`Acceleration','mex`' name-value pair.
- The `semanticseg` function.

Multiview Geometry: Reconstruct 3-D scenes and camera poses from multiple cameras

The `triangulateMultiview` and `bundleAdjustment` functions support images from multiple (pinhole) cameras. The `Intrinsics` property of the `cameraParameters` object enables you to pass intrinsics to related functions in the structure from motion (SfM) workflow.

Velodyne Point Cloud Reader: Read lidar data from VLS- 128 device model

The `velodyneFileReader` object now supports VLS- 128 Velodyne LiDAR® device models.

Point Cloud Normal Distribution Transform (NDT): Register point clouds using NDT with improved performance

Improved performance using the `pcregisterndt` function to register two point clouds with the NDT algorithm.

Code Generation: Generate C/C++ code using MATLAB Coder

The `pcregisterndt` function supports code generation in host target platforms.

These point cloud functions now support code generation in nonhost target platforms.

- `pointCloud`
- `findNearestNeighbors`
- `findNeighborsInRadius`
- `findPointsInROI`
- `removeInvalidPoints`
- `select`

Functionality Being Removed or Changed

The NumOutputChannels argument of unetLayers function has been renamed to NumFirstEncoderFilters

Still runs

The Name-Value pair argument `NumOutputChannels` of `unetLayers` function has been renamed to `NumFirstEncoderFilters`. To update your code, replace all instances of the `NumOutputChannels` argument name with `NumFirstEncoderFilters`.

R2019a

YOLO v2 Object Detection: Train a "you-only-look-once" (YOLO) v2 deep learning object detector

Use YOLO v2, a deep convolutional neural network, for object detection.

- Construct a YOLO v2 network by using the `yolov2Layers` function. Alternatively, you can use the `yolov2TransformLayer`, `yolov2ReorgLayer`, and the `yolov2OutputLayer` functions to manually construct a YOLO v2 network.
- Use the `trainYOLOv2ObjectDetector` function to train the YOLO v2 network.
- Use the `yolov2ObjectDetector` object and the `detect` function to detect objects in a target image using the trained YOLO v2 network.

Use of these objects require Deep Learning Toolbox.

- `yolov2TransformLayer` supports GPU array inputs and GPU code generation.
- `yolov2ReorgLayer` supports GPU array inputs.

3-D Semantic Segmentation: Classify pixel regions in 3-D volumes using deep learning

These functions now support 3-D semantic segmentation:

- `semanticseg`
- `evaluateSemanticSegmentation`
- `pixelClassificationLayer`
- `pixelLabelDatastore`

Code Generation: Generate C code for point cloud processing, ORB, disparity, and ACF functionality using MATLAB Coder

The Computer Vision Toolbox objects and functions in this table now support code generation.

Objects
`pointCloud`
`ORBPoints`
`acfObjectDetector`
Lidar and Point Cloud Processing Functions
`findNearestNeighbors`
`findNeighborsInRadius`
`findPointsInROI`
`removeInvalidPoints`
`select`
`pcnormals`
`pcdownsample`

`pctransform`
`pcregistercpd`
`pcmerge`
`pcfitcylinder`
`pcfitplane`
`pcfitsphere`
`pcdenoise`
`pcsegdist`
`segmentLidarData`
Feature Detection and Extraction Functions
`detectORBFeatures`
Camera Calibration and 3-D Vision Functions
`disparityBM`
`disparitySGM`

ORB Features: Detect and extract oriented FAST and rotated BRIEF (ORB) features

For object recognition or image registration workflows use the `detectORBFeatures` function, `ORBPoints` object, and `'ORB'` name-value pair set to `'Method'` for the `extractFeatures` function.

Velodyne Point Cloud Reader: Read lidar data from Puck LITE and Puck Hi-Res device models

The `velodyneFileReader` object now supports Puck LITE and Puck Hi-Res Velodyne LiDAR device models.

GPU Acceleration for Stereo Disparity: Compute stereo disparity maps on GPUs

These functions compute a disparity map from a stereo-pair image. Both functions support GPU processing.

- `disparityBM` — Uses the block matching method to compute a disparity map
- `disparitySGM` — Uses the semiglobal matching method to compute a disparity map

Ground Truth Data: Select labels by group, type, and attribute

The `groundTruth` object now includes these object functions for selecting labels by group, label type, or attribute.

- `selectLabelsByName`
- `selectLabelsByType`

- `selectLabelsByGroup`

Projection Matrix Estimation: Use direct linear transform (DLT) to compute projection matrix

Use the `estimateCameraMatrix` function to compute a projection matrix. The function uses the DLT algorithm to compute a 2-D projection matrix from 3-D correspondences for depth sensors and cameras. The use of a camera projection matrix speeds up the nearest neighbors search in a point cloud that is generated by an RGBD sensor, such as Microsoft® Kinect®. You can use the `estimateCameraMatrix` function with the `findNearestNeighbors` function to speed up the search.

Organized Point Clouds: Perform faster approximate search using camera projection matrix

You can now pass a camera matrix as an input argument to the `findNearestNeighbors`, `findNeighborsInRadius`, and `findPointsInROI` functions. The functions use the camera projection matrix to identify the relationship between adjacent points,speeding up the nearest neighbor search in organized point clouds.

Point Cloud Viewers: Modify color display and view data tips

The `pcshow` function, `pcplayer` object, and `pcshowpair` function provide figure options for viewing data and changing colormaps.

- Data tips — Select any point to view (*x*, *y*, *z*) point data and additional data value properties. For example, this table shows the data value properties available for a depth image and lidar point cloud.

Point Cloud Data	Data Value Properties
Depth image (RGB-D sensor)	Color, row, column
Lidar	Intensity, range, azimuth angle, elevation angle, row, column

- Background color — Change background color.
- Colormap value — Map a color in the current colormap to a data point.
- View angle — Change the viewing axis angle to an *xz*-, *zx*-, *yz*-, *zy*-, *xy*-, or *yx*-plane.

Image and Video Labeling: Organize labels by logical groups, use assisted freehand for pixel labeling, and other label management enhancements

With the **Image Labeler** and **Video Labeler** apps, you can now:

- Create groups for organizing label definitions. You can also move labels between groups by dragging them.
- Use the assisted freehand feature to create pixel regions of interest (ROIs) for semantic segmentation. This tool automatically finds edges between selected points in an image.

- Move multiple selected ROIs in an image.
- Edit previously created label definitions.
- Add additional list items to a previously created attribute (Video Labeler only).

DeepLab v3+, deep learning, and lidar tracking examples

- Semantic Segmentation Using Deep Learning updated to use DeepLab v3+.
- Object Detection Using YOLO v2 Deep Learning
- Track Vehicles Using Lidar: From Point Cloud to Track List
- Code Generation for Object Detection Using YOLO v2

Relative camera pose computed from homography matrix

The `relativeCameraPose` function can now compute the relative camera pose based on a homography matrix, specified as a `projective2d` object. The relative camera pose can now be computed from a homography matrix in addition to an essential or a fundamental matrix.

Functionality being removed or changed

disparity function will be removed
Still runs

The `disparity` function will be removed in a future release. Use the `disparityBM` or `disparitySGM` functions instead. Use `disparityBM` to compute a disparity map by using block matching method. Use `disparitySGM` to compute disparity map using the semi-global matching method.

selectLabels object function will be removed
Still runs

The `selectLabels` object function will be removed in a future release. Use the `selectLabelsByGroup`, `selectLabelsByType` , and `selectLabelsByName` functions instead.

R2018b

Video Labeler App: Interactive and semi-automatic labeling of ground truth data in a video, image sequence, or custom data source

The **Video Labeler** app enables you to label ground truth in a video, image sequence, or a data custom source. Use the app to interactively specify regions of interest. You can export marked labels from the app and use them to train an object detector or to compare against ground truth data. The app includes computer vision algorithms to automate the labeling of ground truth by using detection and tracking algorithms.

For more details, see Get Started with the Video Labeler.

Lidar Segmentation: Segment ground points from organized 3-D lidar data and organize point clouds into clusters

Use the `segmentGroundFromLidarData` function to segment ground points from organized lidar data. Use the `segmentLidarData` function to organize 3-D point cloud range data into clusters.

Point Cloud Registration: Align 3-D point clouds using coherent point drift (CPD) registration

Use the `pcregistercpd` function for point cloud registration based on the coherent point drift algorithm.

MSAC Fitting: Find a polynomial that best fits noisy data using the M-estimator sample consensus (MSAC)

Use the `ransac` and `fitPolynomialRANSAC` functions to find the coefficients of a polynomial that best fits input noisy data.

Faster R-CNN Enhancements: Train Faster R-CNN object detectors using DAG networks such as ResNet-50 and Inception-v3

You can use the `trainFasterRCNNObjectDetector`, `trainFastRCNNObjectDetector`, or `trainRCNNObjectDetector` functions with deep learning pretrained network to create a CNN model

Use the new layers to train directed acyclic graph (DAG) networks using the faster R-CNN model:

- `roiInputLayer`
- `roiMaxPooling2dLayer`
- `rpnSoftmaxLayer`
- `rpnClassificationLayer`
- `rcnnBoxRegressionLayer`
- `regionProposalLayer`

Semantic Segmentation Using Deep Learning: Create U-Net network

Use the `unetLayers` function to create U-Net semantic segmentation networks.

Velodyne Point Cloud Reader: Support for VLP-32 device

The `velodyneFileReader` now supports the VLP-32C Velodyne LiDAR device.

Labeler Apps: Create a definition table, change file path, and assign data attributes

Use the `labelDefinitionCreator` to create a label definitions table to use with the **Ground Truth Labeler** (requires Automated Driving System Toolbox™), **Image Labeler** and **Video Labeler** apps.

Use `changeFilePaths` to change file paths in the data source and pixel label data of a `groundTruth` object.

Use `attributeType` to specify the type of attributes in the `labelDefinitionCreator`.

OpenCV Interface: Integrate OpenCV version 3.4.0 projects with MATLAB

Integrate OpenCV projects with MATLAB using OpenCV version 3.4.0.

Functionality Being Removed or Changed

Functionality	Change	Use Instead	Compatibility Considerations
`trainFasterRCNNObjectDetector`, `trainFastRCNNObjectDetector`	You must set the training option `MiniBatchSize` property to `1` or these functions produce an error.	Use `NumRegionsToSample` property to specify the number of regions to sample per training iteration.	Change the `MiniBatchSize` property in your code. Set the `NumRegionsToSample` property value in these functions to the value you had the `MiniBatchSize` property (in the `trainingOptions` function) set to before R2018b.
`fastRCNNObjectDetector`	`Network` property changed to a `DAGNetwork`.	No change	None

Functionality	Change	Use Instead	Compatibility Considerations
`fasterRCNNObjectDetector`	• `Network` property changed to a `DAGNetwork`. • The `RegionProposalNetwork` property was removed. • The `MinBoxSizes`, `BoxPyramidScale`, and `NumBoxPyramidLevels` properties were removed.	The `Network` property now contains the complete Faster R-CNN network. Use the `AnchorBoxes` property instead of the `MinBoxSizes`, `BoxPyramidScale`, `NumBoxPyramidLevels`.	None

ClassNames property of PixelClassificationLayer will be removed
Still runs

`ClassNames` property of `PixelClassificationLayer` will be removed. Use `Classes` instead. To update your code, replace all instances of the `ClassNames` property with `Classes`. There are some differences between the functions that require additional updates to your code.

The `ClassNames` property contains a cell array of character vectors. The `Classes` contains a categorical array. To use the `Classes` property with functions that require cell array input, then convert the classes using the `cellstr` function.

R2018a

Lidar Segmentation: Segment lidar point clouds using Euclidean distance

Use the `pcsegdist` function to segment a point cloud into clusters based on the Euclidean distance between individual points.

Lidar Registration: Register multiple lidar point clouds using normal distributions transform (NDT)

Use the `pcregisterndt` function to register multiple lidar point clouds using NDT.

Image Labeler App: Mark foreground and background for pixel labeling

In the **Image Labeler** app, the **Smart Polygon** tool now enables you to refine the segmentation within a polygonal region of interest by marking pixels as foreground or background.

Fisheye Calibration: Interactively calibrate fisheye lenses using the Camera Calibrator app

The **Camera Calibrator** app now includes fisheye lens calibration.

Stereo Baseline Estimation: Estimate baseline of a stereo camera with known intrinsic parameters

Use the `estimateStereoBaseline` function or the **Stereo Camera Calibrator** app to estimate the baseline of a stereo camera when intrinsics of the individual cameras are known.

Interactively rotate point cloud around any point

In the point cloud viewer functions, `pcplayer`, `pcshow`, and `pcshowpair`, you can now rotate a point cloud around any point.

Mutlticlass nonmaxima suppression (NMS)

Use the `selectStrongestBboxMulticlass` function to select the strongest multiclass bounding boxes from overlapping clusters. The function uses greedy nonmaximal suppression (NMS) to eliminate overlapping bounding boxes.

pcregrigid name changed to pcregistericp

The `pcregrigid` function has been renamed to `pcregistericp`. The `pcregistericp` function supports angular difference in degrees. The prior version, `pcregrigid`, used radians. You can still use `pcregrigid`.

Efficiently read and preprocess pixel-labeled images for deep learning training and prediction

The `pixelLabelImageDatastore` object preprocesses pixel-labeled data for training semantic segmentation networks. Preprocessing operations include resizing, rotation, reflection, and cropping.

Version History

In the previous release, you could preprocess pixel-labeled training images by using a `pixelLabelImageSource` object. The `pixelLabelImageSource` function now creates a `pixelLabelImageDatastore` object instead. This new object has similar behavior as a `pixelLabelImageSource`, with additional properties and methods to assist with data preprocessing.

You can use `pixelLabelImageDatastore` for both training and prediction. In the previous release, you could use `pixelLabelImageSource` for training but not prediction.

Code Generation Support for KAZE Detection

The `detectKAZEFeatures` function and the `KAZEPoints` object that it returns now supports code generation.

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
`pixelLabelImageSource`	Still runs	`pixelLabelImageDatastore`	In R2018a, you cannot create a `pixelLabelImageSource` object. The `pixelLabelImageSource` function now creates a `pixelLabelImageDatastore` object.
`vision.VideoFileWriter`, To Multimedia File	removed `wmv` and `.wma` file format support.	Use `..avi` file format for writing audio and video or `.mp4` for highly compressed video.	`.wmv` and `.wma` formats are not supported.

Functionality	Result	Use Instead	Compatibility Considerations
`trainFastRCNNObjectDetector`, `trainFasterRCNNObjectDetector`	Same	Use same functions	The `InitialLearnRate` specified by `trainingOptions` should be increased to achieve similar training results. For example, if the `InitialLearnRate` was between 1e-5 and 1e-6, this should be set to a value between 1e-3 and 1e-4 in R2018a.
`pcshow`, `pcshowpair`, `pcplayer`	Same	Use same functions	For single-color point clouds, the `MarkerSize` property now approximates the marker diameter with a finer scale. To achieve the same size as in previous releases, you must use the square-root of the value used. For example, if you used `100` for `MarkerSize` in previous versions, you must now use `10`.

R2017b

Semantic Segmentation Using Deep Learning: Classify pixel regions in images, evaluate, and visualize segmentation results

Several new features to support semantic segmentation using deep learning techniques:

- `semanticseg`: Perform semantic image segmentation on images and image collections.
- `segnetLayers` and `fcnLayers`: Create SegNet and a fully convolutional network (FCN) segmentation networks.
- `pixelLabelImageSource`: Provides training data for semantic segmentation networks. Additionally supports several on-the-fly data augmentation techniques during training.
- `pixelLabelDatastore`: Provides a data store object that can be used to read pixel label data.
- `evaluateSemanticSegmentation`: Evaluate semantic segmentation results using intersection-over-union (IoU) and other common metrics.
- `crop2dLayer`: Provides a layer for center cropping an input feature map.
- `pixelClassificationLayer`: Creates a pixel classification layer.

Image Labeling App: Interactively label individual pixels for semantic segmentation and label regions using bounding boxes for object detection

Use the **Image Labeler** app for interactive image labeling. You can label rectangular regions of interest (ROI) for object detection, pixels for semantic segmentation, and scenes for image classification. You can also define and execute custom label automation algorithms with the app.

Fisheye Camera Calibration: Calibrate fisheye cameras to estimate intrinsic camera parameters

The following functions support calibration of a wide-angle fisheye camera:

- `fisheyeCalibrationErrors`
- `fisheyeParameters`
- `fisheyeIntrinsics`
- `fisheyeIntrinsicsEstimationErrors`
- `undistortFisheyeImage` and `undistortFisheyePoints`

In addition, the `showExtrinsics` and `showReprojectionErrors` now accommodate fisheye data.

KAZE Features: Detect and extract KAZE features for object recognition or image registration workflows

Detect KAZE points from an image with the `detectKAZEFeatures` function, which returns a `KAZEPoints` object. Extract the KAZE features with the new KAZE method selection of the `extractFeatures` function.

Code generation for camera intrinsics

The `cameraIntrinsics` object now supports code generation.

Image Labeler app replaces Training Image Labeler app

Use the **Image Labeler** app in place of the Training Image Labeler app. The `trainingImageLabeler` function now opens the Image Labeler app.

Ground Truth Labeling Utilities

The `groundTruth`, `groundTruthDataSource`, and `objectDetectorTrainingData` functions are added to support the image labeling.

Ground Truth Labeling Utilities	Description
`groundTruth`	Object for storing ground truth labels
`groundTruthDataSource`	Create a ground truth data source
`objectDetectorTrainingData`	Create training data from ground truth data for an object detector

Computer Vision Example

- Semantic Segmentation Using Deep Learning

R2017a

Deep Learning for Object Detection: Detect objects using Fast R-CNN and Faster R-CNN object detectors

Use `trainFastRCNNObjectDetector` to train a Fast R-CNN deep learning object detector. Use `trainFasterRCNNObjectDetector` to train a Faster R-CNN deep learning object detector. You can train a Faster R-CNN detector to detect multiple object classes. Also new this release are the `fastRCNNObjectDetector` and `fasterRCNNObjectDetector` RCNN support functions.

Object Detection Using ACF: Train object detectors using aggregate channel features

Use `trainACFObjectDetector` to train a classifier to recognize rigid objects. Use `acfObjectDetector` to detect objects in images.

Object Detector Evaluation: Evaluate object detector performance, including precision and miss-rate metrics

Use the `evaluateDetectionPrecision` to return the average precision to measure detection performance. Use the `evaluateDetectionMissRate` to evaluate the miss rate metric for object detection.

OpenCV Interface: Integrate OpenCV version 3.1.0 projects with MATLAB

Integrate OpenCV projects with MATLAB using OpenCV version 3.1.0.

Object for storing intrinsic camera parameters

Use the `cameraIntrinsics` object to store information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

Disparity function updated to fix inconsistent results between multiple invocations

In prior releases, the disparity function sporadically returned different results for depth estimation using `SemiGlobal` method.

Version History

When you use the `disparity` function's `SemiGlobal` method, the results will be different. Examine the results from your code carefully to see if you need to make any adjustments.

Improved algorithm to calculate intrinsics in Camera Calibration apps

This release improves the stability of estimating the principle point in Camera Calibration apps.

Version History

To reproduce prior results, you must use `estimateCameraParameters` function for camera calibration and do not specify the `ImageSize` property.

R2016b

Deep Learning for Object Detection: Detect objects using region-based convolution neural networks (R-CNN)

Use the `trainRCNNObjectDetector` function and the `rcnnObjectDetector` object to train an R-CNN deep learning object detector.

Structure from Motion: Estimate the essential matrix and compute camera pose from 3-D to 2-D point correspondences

Use the `estimateEssentialMatrix`, `estimateWorldCameraPose`, `extrinsicsToCameraPose`, `cameraPoseToExtrinsics` functions to estimate a 3-D structure of a scene from a set of 2-D images. It also adds a `worldToImage` method to the `cameraParameters` class to project world points into an image.

Point Cloud File I/O: Read and write PCD files using Point Cloud File I/O Functions

The `pcread` and `pcwrite` functions now support PCD (point cloud data) format files.

Code Generation for ARM Example: Detect and track faces on a Raspberry Pi 2 target

This release adds two examples that detail the steps for generating code for detecting and tracking faces on the Raspberry Pi 2 hardware.

- Detect Face (Raspberry Pi2)
- Track Face (Raspberry Pi2)

Visual Odometry Example: Estimate camera locations and trajectory from an ordered sequence of images

This release adds a visual odometry example, Monocular Visual Odometry, that details the steps for estimating camera locations and camera trajectory.

cameraPose function renamed to relativeCameraPose

The `cameraPose` function has been renamed to the more descriptive `relativeCameraPose`. Additionally, the function can now accept an essential matrix from the new `estimateEssentialMatrix` function.

New capabilities for Training Image Labeler app

You can now use the Training Image Labeler app to:

- Create a full-image region of interest (ROI).
- Add multiple ROI labels (categories).
- Import ROIs from a `MAT` file or from the workspace.

- Output a table if there are multiple ROI labels.

Train cascade object detector function takes tables and uses imageDatastore

The `trainCascadeObjectDetector` function can now take positive instances as a table or as a `struct` array. It can also take negative images using `imageDatastore`.

Project 3-D world points into image

The `cameraParameters` object now provides a `worldToImage` method that projects 3-D world points into an image.

Code generation support

The following functions and methods:

- `cameraPoseToExtrinsics`
- `extrinsicsToCameraPose`
- `worldToImage` method of the `cameraParameters` object
- `estimateEssentialMatrix`
- `estimateWorldCameraPose`
- `relativeCameraPose`

Plot camera function accepts a table of camera poses

The `plotCamera` function can now accept and plot a table of camera poses.

Eliminate 3-D points input from extrinsics function

The `extrinsics` function no longer accepts 3-D *x,y,z* points as an input. Instead, use the `estimateWorldCameraPose` function.

Version History

When you try to input *x,y,z* points, the `extrinsics` issues a warning.

Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object™, you can call the object with arguments, as if it were a function. The `step` method will continue to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `vision.Pyramid` System object named `gaussPyramid`, then you call the System object as a function with that name.

```
gaussPyramid = vision.Pyramid('PyramidLevel',2);
gaussPyramid(x);
```

The equivalent operation using the `step` method is:

```
gaussPyramid = vision.Pyramid('PyramidLevel',2);
step(gaussPyramid,x);
```

When the `step` method has the System object as its only argument, the function equivalent has no arguments. This function must be called with empty parentheses. For example, `step(sysobj)` and `sysobj()` perform equivalent operations.

R2016a

OCR Trainer App: Train an optical character recognition (OCR) model to recognize a specific set of characters

This release adds the OCR Trainer app.

Structure from Motion: Estimate the camera poses and 3-D structure of a scene from multiple images

This release adds a collection of functions and objects to support structure from motion.

- `bundleAdjustment`
- `pointTrack`
- `viewSet` with several supporting methods for finding tracks and storing camera poses.
- `triangulateMultiview`

Pedestrian Detection: Locate pedestrians in images and video using aggregate channel features (ACF)

This release adds the `detectPeopleACF` function to detect people in a scene.

Bundle Adjustment: Refine estimated locations of 3-D points and camera poses for the structure from motion (SFM) framework

This release adds the `bundleAdjustment` function to estimate camera poses and 3-D points simultaneously.

Multiview Triangulation: Triangulate 3-D locations of points matched across multiple images

This release adds the `triangulateMultiview` function to recover the location of a 3-D world point from its projections into 2-D images.

Rotate matrix to vector and vector to matrix

This release adds the `rotationMatrixToVector` and the `rotationVectorToMatrix` functions. These function implement the Rodrigues transform.

Select spatially uniform distribution of feature points

This release adds the `selectUniform` method to the `SURFPoints`, `cornerPoints`, and `BRISKPoints` objects.

Single camera and stereo camera calibration app enhancements

This release continues to enhance the single camera and stereo camera calibrator apps. The enhancements include:

- Ability to select multiple outlier images that correspond to a high mean reprojection error.
- Minimized analysis charts.
- Removed Viewing tab and placed viewing controls in the toolbar.
- Speeded up calibration for the Single Calibration App.

Point cloud from Kinect V2

This release adds support for extracting point clouds from Kinect V2 using the `pcfromkinect` function.

Point cloud viewer enhancements

This release adds enhancements to the `pcshow`, `pcplayer`, and `pcshowpair` viewers. The functions now rotates the point cloud around the center of the axis and shows the rotation axis. The display of point clouds are downsampled for a large range of data. The downsampling is for display only, it does not modify the data. This change makes display time faster. The functions also now support `subplot`.

Support package for Xilinx Zynq-based hardware

The Computer Vision System Toolbox™ Support Package for Xilinx Zynq-Based Hardware supports verification and prototyping of vision algorithms on Zynq boards. HDL Coder™ is required for customizing the algorithms running on the FPGA fabric of the Zynq device. Embedded Coder® is required for customizing the algorithms running on the ARM® processor of the Zynq device.

- Target your video processing algorithms to Zynq hardware from Simulink
- Stream HDMI signals into Simulink to explore designs with real data
- Generate HDL vision IP cores using HDL Coder
- Deploy algorithms and visualize using HDMI output on a screen

For additional information, see Computer Vision System Toolbox Support Package for Xilinx Zynq-Based Hardware .

C code generation support

This release continues to add C code generation support to new and existing functions and objects.

- `rotationVectorToMatrix`
- `rotationMatrixToVector`
- `insertObjectAnnotation`

This release also adds new support for portable C code generation. You can generate C code for your specific target using any of the newly supported functions. The new support allows you to build the application for your target using a C++ compiler. The C++ compiler links to OpenCV (Version 2.4.9) libraries that you provide for the particular target. To build a standalone application, use `packNGo` with `'Hierarchical' packType`.

The newly supported functions for portable C code generation:

- `vision.CascadeObjectDetector`

- detectBRISKFeatures
- detectFASTFeatures
- detectMSERFeatures
- disparity
- extractFeatures for BRISK, FREAK, and SURF methods.
- detectSURFFeatures
- vision.PeopleDetector
- vision.PointTracker
- matchFeatures
- opticalFlowFarneback

Future removal warning of several System objects

The Computer Vision System Toolbox begins removal of overlapping functionality with equivalent functions.

Version History

Starting in this release, when you use any of the System objects listed in the table, MATLAB issues a warning. Replace the use of the System object with the corresponding function.

Computer Vision System Toolbox System object	Equivalent function
Analysis and Enhancement	
vision.ContrasterAdjuster	imadjust, stretchlim
vision.CornerDetector	detectHarrisFeatures , detectMinEigenFeatures, cornerPoints
vision.EdgeDetector	edge
vision.HistogramEqualizer	histeq
vision.OpticalFlow	opticalFlowLKDoG,opticalFlowLK, opticalFlowFarneback, opticalFlowHS
vision.BoundaryTracer	bwtraceboundary,bwboundaries
Conversions	
vision.Autothresholder	graythresh, multithresh
vision.ColorSpaceConverter	rgb2gray, rgb2ycbcr, makecform, applycform
vision.DemosaicInterpolator	demosaic
vision.GammaCorrector	imadjust
vision.ImageComplementer	imcomplement
vision.ImageDataTypeConverter	im2double,im2single, im2uint8, im2int16, im2uint16

Computer Vision System Toolbox System object	Equivalent function
Filtering	
`vision.ImageFilter`	`imfilter`
`vision.MedianFilter2D`	`medfilt2`
Geometric Transformations	
`vision.GeometricTransformer`	`imwarp`
`vision.GeometricTransformEstimator`	`fitgeotrans`
`vision.GeometricScaler`	`imresize`
`vision.GeometricRotator`	`imrotate`
`vision.GeometricTranslator`	`imtranslate`
Sinks and Sources	
`vision.BinaryFileWriter`	No support
`vision.BinaryFileReader`	No support
Statistics	
`vision.Histogram`	`imhist`
`vision.PSNR`	`psnr`
Text & Graphics	
`vision.MarkerInserter`	`insertMarker`
`vision.ShapeInserter`	`insertShape`
`vision.TextInserter`	`insertText`
Transforms	
`vision.HoughTransform`	`hough`
Utilities	
`vision.ImagePadder`	`padarray`

R2015b

3-D Shape Fitting: Fit spheres, cylinders, and planes into 3-D point clouds using RANSAC

This release adds 3-D point cloud processing functions and classes to fit a sphere, cylinder, or plane to a point cloud.

- `pcfitsphere`
- `pcfitplane`
- `pcfitcylinder`
- `planeModel`
- `sphereModel`
- `cylinderModel`

Streaming Point Cloud Viewer: Visualize streaming 3-D point cloud data from sensors such as the Microsoft Kinect

This release adds the `pcplayer` function for visualizing streaming 3-D point cloud data.

Point Cloud Normal Estimation: Estimate normal vectors of a 3-D point cloud

This release adds the `pcnormals` function to estimate normal vectors for point clouds.

Farneback Optical Flow: Estimate optical flow vectors using the Farneback method

This release adds the `opticalFlowFarneback` object that allows you to compute optical flow. This object supports C code generation.

LBP Feature Extraction: Extract local binary pattern features from a grayscale image

This release adds the `extractLBPFeatures` function that extracts local binary patterns (LBP) from a grayscale image.

Multilanguage Text Insertion: Insert text into image data, with support for multiple languages

This release adds the ability to insert a TrueType font with the `insertText` function. Also new, is the `listTrueTypeFonts` function to list available TrueType fonts on your system.

3-D point cloud extraction from Microsoft Kinect

This release adds the `pcfromkinect` function to convert a Microsoft Kinect depth and RGB image to a 3-*D* point cloud. To use this function you must have the Image Acquisition Toolbox™ installed.

3-D point cloud displays

This release adds the `pcshow` function for plotting a 3-D point cloud. The release also adds the `pcshowpair` function to visualize the differences between two point clouds.

Downsample point cloud using nonuniform box grid filter

This release adds a new syntax to the `pcdownsample` function which returns a downsampled point cloud using a nonuniform box grid filter. The box grid filter can be used as a preprocessing step for point cloud registration.

Compute relative rotation and translation between camera poses

This release adds the `cameraPose` function to compute the relative pose of a calibrated camera based on two views.

Warp block

This release adds the Warp block. Use the Warp block to apply projective or affine transforms to an image. You can use this block to transform the entire image or portions of the image with either a polygon or rectangular region of interest (ROI). This block replaces the Apply Geometric Transformation block.

Version History

Apply Geometric Transformation block will be removed in a future release. Use the Warp block instead.

GPU support for FAST feature detection

This release adds GPU acceleration for the `detectFASTFeatures` function. GPU acceleration for this function requires Parallel Computing Toolbox™.

Camera calibration optimization options

This release adds new properties to the `estimateCameraParameters` function that provide the ability to supply an initial guess for calibration parameters prior to optimization.

C code generation support

This release continues to add C code generation support to new and existing functions and objects.

- `cameraPose`
- `detectCheckerboardPoints`
- `extractLBPFeatures`
- `generateCheckerboardPoints`
- `insertText`

- `opticalFlowFarneback`

Examples for face detection, tracking, 3-D reconstruction, and point cloud registration and display

This release adds the following new featured examples:

- The Face Detection and Tracking Using Live Video Acquisition example shows how to automatically detect and track a face in a live video stream, using the KLT algorithm.
- The Tracking Pedestrians from a Moving Car example shows how to perform automatic detection and tracking of people in a video from a moving camera.
- The Structure From Motion From Two Views example shows you how to generate a point cloud from features matched between two images taken with a calibrated camera.
- The 3-D Point Cloud Registration and Stitching example shows how to combine multiple point clouds to reconstruct a 3-D scene using iterative closest point (ICP) algorithm.

Example using Vision HDL Toolbox for noise removal and image sharpening

This release, the Vision HDL Toolbox product adds the Noise Removal and Image Sharpening featured example. This example shows how to implement a front-end module of an image processing design. This front-end module removes noise and sharpens the image to provide a better initial condition for the subsequent processing.

Removed video package from Computer Vision System Toolbox

This release removes the use of the `video` package. It was replaced with the use of the `vision` package name.

Version History

Replace the use of the `video` name with `vision`.

Morphological System objects future removal warning

The Computer Vision System Toolbox begins removal of overlapping functionality with equivalent functions in the Image Processing Toolbox™.

Version History

Starting in this release, when you use any of the morphological System objects, MATLAB issues a warning. Replace the use of the System object with the corresponding Image Processing Toolbox function.

Computer Vision System Toolbox System object	Equivalent Image Processing Toolbox function
`vision.MorphologicalDilate`	`imdilate`

Computer Vision System Toolbox System object	**Equivalent Image Processing Toolbox function**
`vision.MorphologicalOpen`	`imopen`
`vision.MorphologicalClose`	`imclose`
`vision.MorphologicalErode`	`imerode`
`vision.MorphologicalBottomHat`	`imbothat`
`vision.MorphologicalTopHat`	`imtophat`
`vision.ConnectedComponentLabeler`	`bwlabel` or `bwlabeln` (n-D version)

No edge smoothing in outputs of undistortImage and rectifyStereoImages

The `undistortImage` and `rectifyStereoImages` functions were modified in version 7.0 of the Computer Vision System Toolbox (MATLAB R2015b).

Version History

Previous versions of these functions smoothed the edges of the output image. The current versions do not smooth the image borders in order to increase speed.

VideoFileReader play count

The `vision.VideoFileReader` `PlayCount` default value was changed from `inf` to `1`.

Version History

Any `vision.VideoFileReader` objects saved in previous versions and loaded into R2015b (or later) will loop continuously.

R2015a

3-D point cloud functions for registration, denoising, downsampling, geometric transformation, and PLY file reading and writing

This release adds 3-D point cloud processing functions. It also adds an object for storing a point cloud, and functions to read and write point cloud files in a PLY format.

- `pctransform`
- `pcregrigid`
- `pcmerge`
- `pcdownsample`
- `pcdenoise`
- `pointCloud`
- `pcread`
- `pcwrite`

Image search and retrieval using bag of visual words

This release adds functionality for searching large image collections. It adds the `retrieveImages`, `evaluateImageRetrieval`, and `indexImages` functions, and the `invertedImageIndex` object.

User-defined feature extractor for bag-of-visual-words framework

This release adds the ability to define a custom feature extractor to use with the `bagOfFeatures` framework.

C code generation for eight functions, including rectifyStereoImages and vision.DeployableVideoPlayer on Mac

This release adds C code generation support to eight functions, one System object, and one block. Additionally, the new optical flow functions support C code generation.

- `rectifyStereoImages`
- `reconstructScene`
- `undistortImage`
- `triangulate`
- `extrinsics`
- `cameraMatrix`
- `cameraParameters`
- `stereoParameters`
- `vision.DeployableVideoPlayer`
- To Video Display block on the Mac
- `opticalFlow`
- `opticalFlowHS`
- `opticalFlowLK`

- `opticalFlowLKDoG`

Mac support for vision.DeployableVideoPlayer and To Video Display block

This release enables the `vision.DeployableVideoPlayer` System object and the To Video Display block on the Mac. These players are capable of displaying high-definition video at high frame rates.

Version History

The player displays frames at the rate of the video input. Setting the `FrameRate` property has no effect on changing the rate of the display. The `FrameRate` property will be removed in a future release.

Plot camera figure in 3-D coordinate system

This release adds the `plotCamera` function. The function returns a camera visualization object to plot a camera figure in the current axes.

Line width for insertObjectAnnotation

This release introduces a `LineWidth` property for the `insertObjectAnnotation` function. Setting this property enables you to modify the line width of inserted annotations.

Upright option for extractFeatures

This release introduces the `Upright` property for the `extractFeatures` function. This property enables you to restrict extracted features to an upright orientation only.

Rotate integral images in integralImage, integralKernel, and integralFilter functions

This release introduces rotated summed area table (RSAT) support to the `integralImage`, `integralKernel`, and `integralFilter` functions.

Performance improvements

- `detectCheckerboardPoints` and `estimateCameraParameters` functions
- `integralFilter` function
- `vision.DeployableVideoPlayer` System object
- To Video Display block

Optical flow functions and object

This release adds the `opticalFlowHS`, `opticalFlowLK`, and `opticalFlowLKDoG` functions. These functions enable you to compute optical flow using the Horn-Schunck and Lucas-Kanade methods.

This release also adds the `opticalFlow` object for storing optical flow velocity matrices. The functions and the object support C code generation.

Examples for image retrieval, 3-D point cloud registration and stitching, and code generation for depth estimation from stereo video

- Image Retrieval Using Customized Bag-of-Features
- 3-D Point Cloud Registration and Stitching
- Code Generation for Depth Estimation From Stereo Video

R2014b

Stereo camera calibration app

This release adds a stereo calibration app. You can use the app to calibrate a stereo camera. Use the `stereoCameraCalibrator` function to invoke the app. See the Stereo Calibration Using the Stereo Camera Calibrator App tutorial.

imageSet class for handling large collections of image files

This release adds the `imageSet` class for managing large collections of image files. Use the `imageSet` `partition` method to create subsets of image sets. You can use these subsets to provide training and validation images for classification.

Bag-of-visual-words suite of functions for image category classification

This release adds a suite of functions to support the bag-of-features workflow. The workflow allows you to manage your image collections and partition them into training and validation sets. It constructs a bag of visual words for use in image category classification. The training and classification includes support for Parallel Computing Toolbox.

- `imageSet`
- `bagOfFeatures`
- `trainImageCategoryClassifier`
- `imageCategoryClassifier`

Approximate nearest neighbor search method for fast feature matching

This release provides updates to the `matchFeatures` function. The update replaces previous matching methods with `'Exhaustive'` and `'Approximate'` Nearest Neighbor methods. It also adds the `Unique` match logical property to only return unique matches from the input feature set.

As a result of this update, the following methods and properties were removed:

- `'NearestNeighborRatio'`, `'NearestNeighborSymmetric'`, and `'Threshold'` matching methods
- `'normxcorr'` normalized cross-correlation metric and the `'Prenormalized'` properties

Use the following new methods to match the behavior of the removed properties.

Previous Match Method	Set New Match Method
`'NearestNeighborRatio'`	Set the `Method` property to `'Exhaustive'` and the `Unique` property to `false`.
`'NearestNeighborSymmetric'`	Set the `Method` property to `'Exhaustive'`, the `Unique` property to `true`, and the `MaxRatio` to `1`.

3-D point cloud visualization function

This release adds the `showPointCloud` function for plotting point clouds.

3-D point cloud extraction from Kinect

This release adds the `depthToPointCloud` function to convert a Kinect depth image to a 3-*D* point cloud. This function requires the Image Acquisition Toolbox.

Kinect color image to depth image alignment

This release adds the `alignColorToDepth` function for registering a Kinect color image to a depth image. This function requires the Image Acquisition Toolbox.

Point locations from stereo images using triangulation

This release adds the `triangulate` function. You can use this function to find 3-D locations of matching points in stereo images.

Red-cyan anaglyph from stereo images

This release adds the `stereoAnaglyph` function. Use this function to combine stereo images to form an anaglyph, which can be viewed with red-blue stereo glasses.

Point coordinates correction for lens distortion

This release adds the `undistortPoints` function. Use this function to remove the effects of lens distortion from individual point locations.

Camera projection matrix

This release adds the `cameraMatrix` function. You can use the matrix returned by this function to project 3-D world points in homogeneous coordinates into an image.

Calculation of calibration standard errors

This release adds the ability to return the standard errors incurred during the camera calibration process. The `estimateCameraParameters` function returns the errors. You can use the errors to evaluate the quality of the camera calibration. You can return errors for both single and stereo camera calibration.

Live image capture in Camera Calibrator app

You can now do live camera calibration using the Camera Calibrator app. The new **Image Capture** tab allows you to bring live images from USB Webcams into the app. Previously, you had to save your images to disk and manually add them into the app.

The image capture functionality in the Camera Calibrator app allows you to:

- Capture live images from USB Webcams
- Browse the captured images
- Save acquired images
- Integrate between image acquisition and calibration
- Control camera properties, such as brightness and contrast.

Use the `cameraCalibrator` function to open the app. Then select **Add Images > From camera** to open the **Image Capture** tab. Select your device, set any properties, and define the capture settings. You can then capture images and calibrate the camera.

Region of interest (ROI) copy and paste support for Training Image Labeler app

This release adds the ability to copy-and-paste regions of interest within the Training Image Labeler app.

Non-maximal suppression of bounding boxes for object detection

This release adds the `bboxOverlapRatio` and the `selectStrongestBbox` functions. Use `bboxOverlapRatio` to compute the overlap ratio between pairs of bounding boxes. Use `selectStrongestBbox` to select the strongest bounding boxes from overlapping clusters.

Linux support for deployable video player

This release adds Linux® support for the To Video Display block and the `vision.DeployableVideoPlayer` System object. This added support includes the ability to generate code.

GPU support for Harris feature detection

This release adds GPU acceleration for the `detectHarrisFeatures` function. GPU acceleration for this function requires Parallel Computing Toolbox.

Extended language support package for optical character recognition (OCR)

This release adds the ability to download additional language support for the `ocr` function. You can use the `visionSupportPackages` function to download the language support package.

Support package for OpenCV Interface

This release adds a support package to help you integrate your OpenCV C++ code into MATLAB. It lets you build MEX files that calls OpenCV functions. You can use the `visionSupportPackages` function to download the OpenCV Interface support package.

Convert format of rectangle to a list of points

This release adds the `bbox2points` function. You can use this function to convert a rectangle, specified as [*x*, *y*, *width*, *height*], into a list of [*x*, *y*] points.

Bag-of-visual-words, stereo vision, image stitching, and tracking examples

This release adds several new examples.

- Pedestrian tracking from a moving car
- Image classification using bag-of-visual-words workflow
- Face tracking from a web cam
- Evaluate camera calibration results
- Image stitching
- Depth estimation from a stereo video
- Code generation with PackNGo

R2014a

Stereo vision functions for rectification, disparity calculation, scene reconstruction, and stereo camera calibration

This release adds a suite of stereo vision algorithms to the Computer Vision System Toolbox.

- `rectifyStereoImages` for stereo rectification.
- `reconstructScene` for computing dense 3-*D* reconstruction based on a disparity map.
- `extrinsics` for computing location of a calibrated camera.
- `stereoParameters` object for storing stereo system parameters.
- `detectCheckerboardPoints` extended to support stereo calibration
- `disparity` adds new method for semi-global block matching.
- `estimateCameraParameters` extended to calibrate stereo cameras.
- `cameraParameters` object for storing camera parameters.
- `showExtrinsics` extended to support stereo cameras.
- `showReprojectionErrors` extended to support stereo cameras.

Version History

This release modifies the `disparity` function's default method for block matching. The new `SemiGobal` default method may produce different results in code created that used the previous `BlockMatching` default method. To obtain the same results, set the `'Method'` property to `'BlockMatching'`.

Optical character recognition (OCR)

This release adds the `ocr` function and `ocrText` object. You can use the `ocr` function to recognize text using optical character recognition. The `ocrText` object stores optical character recognition results.

Binary Robust Invariant Scalable Keypoints (BRISK) feature detection and extraction

This release adds the `detectBRISKFeatures` function. You can use the Binary Robust Invariant Scalable Keypoints (BRISK) algorithm to detect multi-scale corner features. This release also adds the `BRISKPoints` object to store the BRISK detection results. This release adds BRISK descriptor to the `extractFeatures`.

App for labeling images for training cascade object detectors

This release adds a Training Image Labeler app. The app can be used to select regions of interest in images for the purpose of training a classifier. You can invoke the app by using the `trainingImageLabeler` function. See the Label Images for Classification Model Training tutorial.

C code generation for Harris and minimum eigenvalue corner detectors using MATLAB Coder

This release adds C code generation support for the `detectHarrisFeatures` and `detectMinEigenFeatures` functions. This release also adds C code generation to the `estimateGeometricTransform` function.

Line width control for insertShape function and Draw Shapes block

This release adds line thickness control to the `insertShape` function and the Draw Shapes.

Replacing vision.CameraParameters with cameraParameters

This release replaces the `vision.CameraParameters` object with the `cameraParameters` object. The new object contains identical functionality.

Version History

You must replace the `vision.CameraParameters` with `cameraParameters` object in your code. If you attempt to create a `vision.CameraParameters` object, MATLAB returns an error.

Output view modes and fill value selection added to undistortImage function

This release adds new output view modes and fill value selection to the `undistortImage` function. You can control the output view size by setting the `OutputView` property. You can also set the fill value with the `FillValues` property.

Generated code optimized for the matchFeatures function and vision.ForegroundDetector System object

This release provides generated code optimization for the `matchFeatures` function and the `vision.ForegroundDetector` System object on a Windows®, Linux, Mac OS platform.

Merging mplay viewer into implay viewer

This release merges the `mplay` viewer function from the Computer Vision System Toolbox into the `implay` function in Image Processing Toolbox.

Version History

Use the `implay` function with functionality identical to `mplay`. The `mplay` function will be removed in a future release.

MPEG-4 and JPEG2000 file formats added to vision.VideoFileWriter System object and To Multimedia File block

This release adds support for writing MPEG-4 and JPEG 2000 file formats with the `vision.VideoFileWriter` object and the To Multimedia File block.

Region of interest (ROI) support added to detectMSERFeatures and detectSURFFeatures functions

This release adds region of interest (ROI) support to the `detectMSERFeatures` and `detectSURFFeatures` functions.

MATLAB code script generation added to Camera Calibrator app

This release adds MATLAB code script generation to the Camera Calibrator app.

Featured examples for text detection, OCR, 3-D reconstruction, 3-D dense reconstruction, code generation, and image search

This release the Computer Vision System Toolbox adds several new featured examples:

- Automatically Detect and Recognize Text in Natural Images
- Image Search using Point Features
- Recognize Text Using Optical Character Recognition (OCR)
- Code Generation for Feature Matching and Registration (updated)
- Stereo Calibration and Scene Reconstruction
- Sparse 3-D Reconstruction From Multiple Views

Play count default value updated for video file reader

This release the Computer Vision System Toolbox modifies the default value from `inf` to `1` for the `PlayCount` property of the VideoFileReader System object. This change allows proper functionality while using the `isDone` method.

R2013b

Camera intrinsic, extrinsic, and lens distortion parameter estimation using camera calibration app

This release adds a camera calibration app. The app can be used to estimate camera intrinsic and extrinsic parameters, and to compute parameters needed to remove the effects of lens distortion from an image. You can invoke the calibrator using the `cameraCalibrator` function. See the Find Camera Parameters with the Camera Calibrator tutorial.

Camera calibration functions for checkerboard pattern detection, camera parameter estimation, correct lens distortion, and visualization of results

This release adds a suite of functions that, when used together, provide a workflow to calibrate a camera:

- `detectCheckerboardPoints`
- `estimateCameraParameters`
- `generateCheckerboardPoints`
- `showExtrinsics`
- `showReprojectionErrors`
- `undistortImage`
- `vision.CameraParameters`

Histogram of Oriented Gradients (HOG) feature extractor

This release adds the `extractHOGFeatures` descriptor function. The extracted features encode local shape information from regions within an image. You can use this function for many tasks including classification, detection, and tracking.

C code generation support for 12 additional functions

This release adds C code generation support for several Computer Vision System Toolbox functions, classes, and System objects.

- `extractHOGFeatures`
- `extractFeatures`
- `detectSURFFeatures`
- `disparity`
- `detectMSERFeatures`
- `detectFASTFeatures`
- `vision.CascadeObjectDetector`
- `vision.PointTracker`
- `vision.PeopleDetector`
- `MSERRegions`
- `cornerPoints`

- `SURFPoints`

System objects matlab.system.System warnings

The System object base class, `matlab.system.System` has been replaced by `matlab.System`. If you use `matlab.system.System` when defining a new System object, a warning message results.

Version History

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

Restrictions on modifying properties in System object Impl methods

When defining a new System object, certain restrictions affect your ability to modify a property.

You cannot use any of the following methods to modify the properties of an object:

- `cloneImpl`
- `getDiscreteStateImpl`
- `getDiscreteStateSpecificationImpl`
- `getNumInputsImpl`
- `getNumOutputsImpl`
- `getOutputDataTypeImpl`
- `getOutputSizeImpl`
- `isInputDirectFeedthroughImpl`
- `isOutputComplexImpl`
- `isOutputFixedSizeImpl`
- `validateInputsImpl`
- `validatePropertiesImpl`

This restriction is required by code generation, which assumes that these methods do not change any property values. These methods are validation and querying methods that are expected to be constant and should not impact the algorithm behavior.

Also, if either of the following conditions exist:

- You plan to generate code for the object
- The object will be used in the MATLAB System block

you cannot modify tunable properties for any of the following runtime methods:

- `outputImpl`
- `processTunedPropertiesImpl`
- `resetImpl`
- `setupImpl`
- `stepImpl`

- `updateImpl`

This restriction prevents tunable parameter updates within the object from interfering with updates from outside the generated code. Tunable parameters can only be changed from outside the generated code.

Version History

If any of your class definition files contain code that changes a property in one of the above `Impl` methods, move that property code into an allowable `Impl` method. Refer to the System object `Impl` method reference pages for more information.

R2013a

Cascade object detector training using Haar, Histogram of Oriented Gradients (HOG), and Local Binary Pattern (LBP) features

This release adds the `trainCascadeObjectDetector` function for Haar, Histogram of Oriented Gradients (HOG), and Local Binary Pattern (LBP) features. The function creates a custom classification model to use with the `vision.CascadeObjectDetector` cascade object detector.

Fast Retina Keypoint (FREAK) algorithm for feature extraction

This release adds the Fast Retina Keypoint (FREAK) descriptor algorithm to the `extractFeatures` function. This function now supports the FREAK method for descriptor extraction.

Hamming distance method for matching features

This release adds the Hamming distance method to the `matchFeatures` function in support of binary features produced by descriptors such as the FREAK method for extraction. It also adds the new `binaryFeatures` object, which is an output of the `extractFeatures` function and serves as an input to the `matchFeatures` function.

Multicore support in matchFeatures function and ForegroundDetector System object

This release brings multicore performance improvements for the `matchFeatures` function and the `vision.ForegroundDetector` detector.

Functions for corner detection, geometric transformation estimation, and text and graphics overlay, augmenting similar System objects

This release adds several new functions. For corner detection, the new `detectHarrisFeatures`, `detectMinEigenFeatures`, and `detectFASTFeatures` functions. The `insertText`, `insertMarker`, and `insertShape` functions for inserting text, markers, and shapes into images and video. Lastly, the `estimateGeometricTransform` function for estimating a geometric transform from putatively matched point pairs.

Error-out condition for old coordinate system

This release ends support for the row-column coordinate system for the Computer Vision System Toolbox algorithms. All blocks are replaced with blocks using [x y] coordinates, and all functions and System objects are updated to use the one-based [x y] convention. Using any MATLAB or Simulink related algorithms will error out when using RC-based functions or blocks.

Version History

Conventions for indexing, spatial coordinates, and representation of geometric transforms were changed in R2011b to provide improved interoperability with the Image Processing Toolbox product. Beginning in this release, all Computer Vision System Toolbox blocks, functions, classes, and System objects will only operate in the [x y] coordinate system. Blocks affected by the [x y] coordinate system should be replaced with blocks of the same name from the Vision library. Adjust your models, code, and data as necessary.

For extended details on the coordinate system change, see “Conventions Changed for Indexing, Spatial Coordinates, and Representation of Geometric Transforms” on page 22-2 R2011b Release Notes.

Support for nonpersistent System objects

You can now generate code for local variables that contain references to System objects. In previous releases, you could not generate code for these objects unless they were assigned to persistent variables.

New method for action when System object input size changes

The new `processInputSizeChangeImpl` method allows you to specify actions to take when an input to a System object you defined changes size. If an input changes size after the first call to `step`, the actions defined in `processInputSizeChangeImpl` occur when `step` is next called on that object.

Scaled double data type support for System objects

System objects now support scaled double data types.

Scope Snapshot display of additional scopes in Simulink Report Generator

Using Simulink Report Generator™ software, you can include snapshots of the display produced by a Scope block in a generated report. The Scope Snapshot component, which inserts images of the Simulink Scope block and XY Graph block, now supports the Video Viewer block in Computer Vision System Toolbox software.

Note This feature requires that you have a license for the Simulink Report Generator product.

For more information, see the Simulink Report Generator product documentation.

R2012b

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 `assignDetectionsToTracks` 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

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.

Version History

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

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.

Version History

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

Dependency on DSP System Toolbox and Signal Processing Toolbox Software Removed

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.

Version History

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.

Version History

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.

Version History

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 Using CAMShift`. 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

This release changes error and warning message identifiers.

Version History

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

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`.

Version History

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

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

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

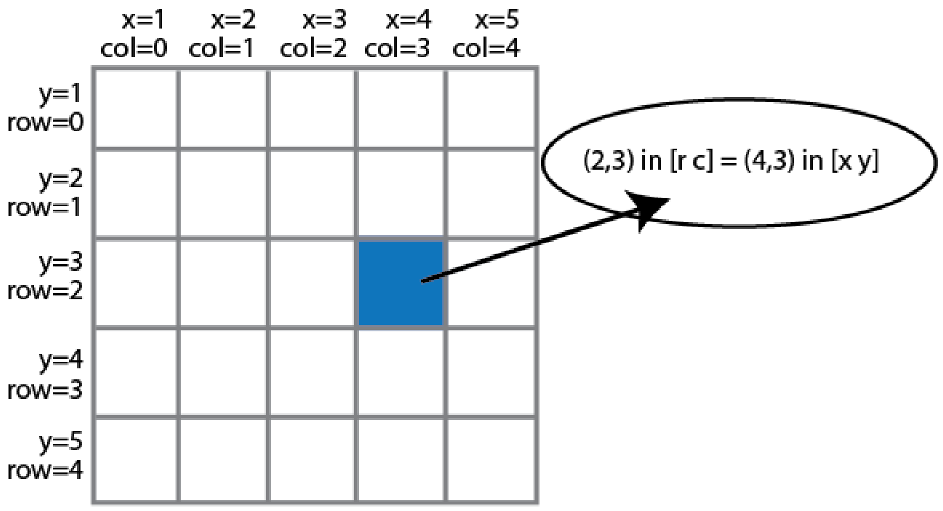
The change from zero-based to one-based indexing simplifies the ability to blend Image Processing Toolbox functionality with Computer Vision System Toolbox algorithms and visualization functions.

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,:)]
```

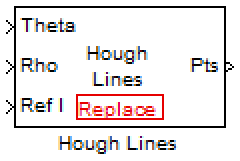
Note: You cannot use this code to remap a projective transformation matrix. You must derive the tformXY matrix from your data.

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.
`estimateUncalibratedRectification`	Fundamental matrix, matched points, and output projective transformation matrices provided in new format.	Fundamental matrix formatted only for zero-based [r;c] coordinate system	Fundamental matrix formatted for one-based [x y] coordinate system.
		[r;c] 2-by-*N* zero-based points.	[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.

Functions	Description of Update	Prior to R2011b	R2011b
	Output intersection points converted to [x y] one-based coordinate system.	The function returned the intersection points in an 4-by-*M* matrix. format of [r1;c1;r2;c2] zero-based coordinate system.	The function returns the intersection points in an *M*-by-4 matrix of format of [x1, y1, x2, y2] one-based coordinate system.
`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.
`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.GeometricTransformEstimator`	Converted formatting for input points.	Input points: [r1 r2;c1 c2].	Input points: [x1 y1; x2 y2].

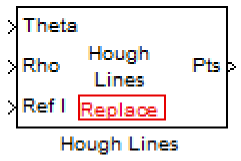
System Objects	Description of Update	Prior to R2011b	R2011b
	Converted transformation matrix to [x y] one-based coordinate system.	Transformation matrix formatted only for zero-based [r;c] coordinate system.	Transformation matrix format matches Image Processing Toolbox format.
`vision.HoughLines`	Converted 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.
`vision.Maximum` `vision.Mean` `vision.Minimum` `vision.StandardDeviation` `vision.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].
`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.
`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-*N* 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.
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.
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.

Blocks	Description of Update	Prior to R2011b	R2011b
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.
nsert 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].
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.

Version History

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, `Find Image Rotation and Scale 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

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.

Version History

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

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.

Version History

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

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

Version History

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

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.

Version History

Remove the property from your code, and update accordingly.

vision.GeometricTransformEstimator Default Properties Changed

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`

Version History

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

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.

Version History

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 for more information.

System Object DataType and CustomDataType Properties Changes

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.

Version History

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

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

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

Version History

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

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.

Version History

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

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.

Version History

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

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

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.

Version History

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

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

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

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.