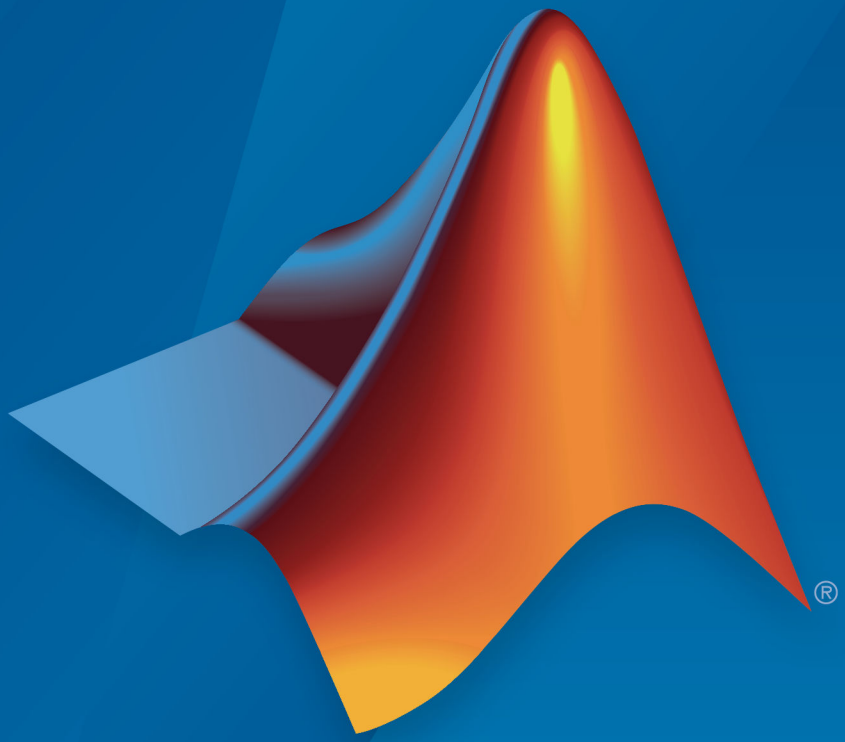
# Automated Driving System Toolbox™ Release Notes



MATLAB®

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.
3 Apple Hill Drive
Natick, MA 01760-2098

*Automated Driving System Toolbox™ Release Notes*

# Contents

## R2018b


**Bird's-Eye Scope for Simulink: Analyze sensor coverages, detections, and tracks in your model** . . . . . **1-2**

**Prebuilt Driving Scenarios: Test driving algorithms using Euro NCAP and other prebuilt scenarios** . . . . . **1-2**

**OpenDRIVE File Import Support: Load OpenDRIVE roads into a driving scenario** . . . . . **1-2**

**Improved Collision Checking in vehicleCostmap Object: Configure collision checking to plan paths through narrow passages** . . . . . **1-3**

**Kinematic Lateral Controller: Control the steering angle of an autonomous vehicle** . . . . . **1-3**

**Monocular Camera Parameter Estimation: Configure a monocular camera by estimating its extrinsic parameters** . . . . . **1-3**

**Radar Sensor Model Enhancements: Model occlusions in radar sensors** . . . . . **1-3**

**Obtain transition poses and direction changes from a planned path** . . . . . **1-4**

**Define multiple custom labels in Ground Truth Labeler connector** . . . . . **1-4**

**Ground Truth Labeler enhancements** . . . . . **1-4**

**Actors follow road elevation and banking angles in Driving Scenario Designer** . . . . . **1-4**

**Monocular camera setup with fisheye lens example** . . . . . **1-5**

**Sensor fusion and tracking examples** . . . . . **1-5**

**Functionality being removed or changed** . . . . . **1-5**

InflationRadius and VehicleDimensions properties of vehicleCostmap object are not recommended . . . . . **1-5**

connectingPoses function and driving.Path object properties KeyPoses and NumSegments are not recommended . . . . . **1-6**

Corrections to Image Width and Image Height camera parameters of Driving Scenario Designer . . . . . **1-7**

## R2018a

**Driving Scenario Designer: Interactively define actors and driving scenarios to test controllers and sensor fusion algorithms** . . . . . **2-2**

**Path Planning: Plan driving paths using an RRT* path planner and costmap** . . . . . **2-2**

**Streaming Geographic Map Display: Visualize a geographic route on a map** . . . . . **2-2**

**Ground Truth Pixel Labeling: Interactively label individual pixels in video data** . . . . . **2-2**

**Ground Truth Label Attributes: Organize and classify ground truth labels using attributes and sublabels** . . . . . **2-3**

**Lidar Segmentation: Quickly segment 3-D point clouds from lidar** . . . . . **2-3**

**ACC Reference Application: Use a reference model to simulate and test adaptive cruise controller (ACC) systems** . . . . . **2-3**

**Point Cloud Reader for Velodyne PCAP Files: Import Velodyne lidar data into MATLAB** . . . . . **2-3**

**Detect lanes more precisely by using third-degree polynomial lane boundary models** . . . . . **2-3**

**Add and detect lanes in Driving Scenario** . . . . . **2-4**

**Transform [x,y,z] locations in vehicle coordinates to image coordinates** . . . . . **2-4**

**Path method being removed** . . . . . **2-4**

**Direction of Yaw Angle Rotation Adjusted** . . . . . **2-5**

## R2017b

**Sensor Fusion Simulink Blocks: Track multiple objects and fuse detections from multiple sensors** . . . . . **3-2**

**Sensor Simulation Using Simulink Blocks: Generate synthetic object lists from camera and radar sensor models** . . . . . **3-2**

**Ground Truth Labeling App: Reverse playback capability while processing algorithms** . . . . . **3-2**

**Code Generation for Sensor Models: Generate C code for camera and radar sensor models** . . . . . **3-2**

**Autonomous Driving Examples** . . . . . **3-2**

## R2017a

**Ground Truth Labeling** . . . . . **4-2**

**Monocular Camera Sensor Configuration** . . . . . **4-2**

**Object and Lane Boundary Detection** . . . . . **4-2**

**Multi-object Tracking** . . . . . **4-3**

**Bird's-Eye Plot** . . . . . **4-4**

**Driving Scenario Generation and Sensor Models** . . . . . **4-4**

**Automated Driving Examples** . . . . . **4-4**

# R2018b

**Version: 1.3**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## Bird's-Eye Scope for Simulink: Analyze sensor coverages, detections, and tracks in your model

The **Bird's-Eye Scope** displays streaming detections and object tracks from your model on a bird's-eye plot. You can use the **Bird's-Eye Scope** to:

- Inspect the coverage areas of radar and vision sensors.
- Analyze the sensor detections of lanes and actors in a driving scenario.
- Analyze the tracks of moving objects.

To get started using the scope, see “Visualize Sensor Data and Tracks in Bird's-Eye Scope”.

## Prebuilt Driving Scenarios: Test driving algorithms using Euro NCAP and other prebuilt scenarios

In the **Driving Scenario Designer** app, you can now test that your algorithms comply with ADAS industry standards by using prebuilt Euro NCAP® driving scenarios. These scenarios model multiple variations of Euro NCAP test procedures for lane keeping assist, automatic emergency braking, and emergency lane keeping. For more details, see “Generate Synthetic Detections from a Euro NCAP Scenario” and the “Automatic Emergency Braking with Sensor Fusion” example.

In addition to Euro NCAP scenarios, the app includes prebuilt driving scenarios of common driving maneuvers at intersections. See “Generate Synthetic Detections from a Prebuilt Driving Scenario”

## OpenDRIVE File Import Support: Load OpenDRIVE roads into a driving scenario

In the **Driving Scenario Designer** app, you can now include roads built using the OpenDRIVE® format specification. For more details, see “Add OpenDRIVE Roads to Driving Scenario”.

You can also load these roads into a `drivingScenario` object by using the `roadNetwork` function.

## Improved Collision Checking in vehicleCostmap Object: Configure collision checking to plan paths through narrow passages

The `inflationCollisionChecker` function creates a configuration object that specifies how the `vehicleCostmap` object checks for collisions. You can use this collision-checking configuration object to reduce the amount of obstacle inflation in the costmap. By reducing this inflation amount, path planning algorithms can plan collision-free paths through narrow passages such as parking spots.

For compatibility considerations, see “InflationRadius and VehicleDimensions properties of vehicleCostmap object are not recommended” on page 1-5.

## Kinematic Lateral Controller: Control the steering angle of an autonomous vehicle

The Lateral Controller Stanley block and `lateralControllerStanley` function compute the steering angle of a vehicle using the Stanley method, a kinematic control algorithm. Use this block or function in a closed-loop simulation to adjust the steering angle of a vehicle as it follows a path. To learn more, see “Lateral Control Tutorial”.

## Monocular Camera Parameter Estimation: Configure a monocular camera by estimating its extrinsic parameters

The `estimateMonoCameraParameters` function estimates the extrinsic parameters of a monocular camera that has been calibrated using a checkerboard pattern. For more details, see “Calibrate a Monocular Camera”.

## Radar Sensor Model Enhancements: Model occlusions in radar sensors

In the `radarDetectionGenerator` System object™, use the `HasOcclusion` property to generate detections only from objects for which the radar has a direct line of sight.

## Obtain transition poses and direction changes from a planned path

The `driving.Path` object returned by `pathPlannerRRT` now contains more specific descriptions of path segments, including their motion lengths, motion directions, and motion types (Dubins or Reeds-Shepp). Use the `interpolate` function to sample poses along the path, including transition poses, and to return changes in direction. You can then use these sampled poses and direction changes to develop a path smoothing algorithm.

For compatibility considerations, see “connectingPoses function and driving.Path object properties KeyPoses and NumSegments are not recommended” on page 1-6.

## Define multiple custom labels in Ground Truth Labeler connector

You can now synchronize the **Ground Truth Labeler** app with external labeling tools containing multiple custom labels. Specify these labels and their descriptions using the `LabelName` and `LabelDescription` properties of the `driving.connector.Connector` class.

## Ground Truth Labeler enhancements

The **Ground Truth Labeler** app now includes visuals indicating the relationship between the labels and sublabels of an image. For more details on the label-sublabel relationship, see “Use Sublabels and Attributes to Label Ground Truth Data” (Computer Vision System Toolbox).

In addition, in the Label Summary window, you can now navigate between unlabeled frames. For more details on the Label Summary window, see “View Summary of Ground Truth Labels” (Computer Vision System Toolbox).

## Actors follow road elevation and banking angles in Driving Scenario Designer

In the **Driving Scenario Designer** app, when you create an actor and specify waypoints for it to follow, the actor now travels along the elevation angle and banking angle of the road.

## Monocular camera setup with fisheye lens example

The “Configure Monocular Fisheye Camera” example shows how to set up a monocular camera that has a fisheye lens.

## Sensor fusion and tracking examples

The following examples require a Sensor Fusion and Tracking Toolbox™ license.

- The “Extended Object Tracking” example shows how to track objects whose dimensions span multiple sensor resolution cells.
- The “Visual-Inertial Odometry Using Synthetic Data” example shows how to estimate the pose (position and orientation) of a vehicle by using an inertial measurement unit (IMU) and a monocular camera.

## Functionality being removed or changed

**`InflationRadius` and `VehicleDimensions` properties of `vehicleCostmap` object are not recommended**
*Still runs*

The `InflationRadius` and `VehicleDimensions` properties of `vehicleCostmap` are not recommended. Instead:

1. Use the `inflationCollisionChecker` function to create an `InflationCollisionChecker` object, which has the properties `InflationRadius` and `VehicleDimensions`.
2. Specify this object as the value of the `CollisionChecker` property of `vehicleCostmap`.

There are no current plans to remove the `InflationRadius` and `VehicleDimensions` properties of `vehicleCostmap`. If you do specify these properties, the values in the corresponding properties of `CollisionChecker` are updated to match.

When the `vehicleCostmap` object was introduced in R2018a, this object inflated obstacles based on the specified inflation radius and vehicle dimensions only. The `InflationCollisionChecker` object, which is specified in the `CollisionChecker` property of `vehicleCostmap`, provides additional configuration options for inflating obstacles. For example, you can specify the number of circles used to represent the vehicle shape, enabling more precise collision checking.

#### Update Code

The table shows a typical usage of the `InflationRadius` and `VehicleDimensions` properties of `vehicleCostmap`. It also shows how to update your code using the corresponding properties of an `InflationCollisionChecker` object.

| Discouraged Usage | Recommended Replacement |
|---|---|
| `vehicleDims = vehicleDimensions(5,2);`<br>`inflationRadius = 1.2;`<br>`costmap = vehicleCostmap(C, ...`<br>`    'VehicleDimensions',vehicleDims, ...`<br>`    'InflationRadius',inflationRadius);` | `vehicleDims = vehicleDimensions(5,2);`<br>`inflationRadius = 1.2;`<br>`ccConfig = inflationCollisionChecker(vehicleDims, ...`<br>`    'InflationRadius',inflationRadius);`<br>`costmap = vehicleCostmap(C, ...`<br>`    'CollisionChecker',ccConfig);` |

### connectingPoses function and driving.Path object properties KeyPoses and NumSegments are not recommended

*Still runs*

The `connectingPoses` function and the `KeyPoses` and `NumSegments` properties of the `driving.Path` object are not recommended. Instead, use the `interpolate` function, which returns key poses, connecting poses, transition poses, and direction changes. The `KeyPoses` and `NumSegments` properties are no longer relevant. `KeyPoses`, `NumSegments`, and `connectingPoses` will be removed in a future release.

In R2018a, `connectingPoses` enabled you to obtain intermediate poses either along the entire path or along the path segments that are between key poses (as specified by `KeyPoses`). Using the `interpolate` function, you can now obtain intermediate poses at any specified point along the path. The `interpolate` function also provides transition poses at which changes in direction occur.

#### Update Code

Remove all instances of `KeyPoses` and `NumSegments` and replace all instances of `connectingPoses` with `interpolate`. The table shows typical usages of `connectingPoses` and how to update your code to use `interpolate` instead. Here, `path` is a `driving.Path` object returned by `pathPlannerRRT`.

| Discouraged Usage | Recommended Replacement |
|---|---|
| `poses = connectingPoses(path);` | `poses = interpolate(path);` |

| Discouraged Usage | Recommended Replacement |
| --- | --- |
| `segID = 1;`<br>`posesSegment = connectingPoses(path,segID);` | `interpolate` does not have a direct syntax for obtaining segment poses. However, you can sample poses of a segment using a specified step time. For example:<br><br>`step = 0.1;`<br>`samples = 0 : step : path.PathSegments(1).Length`<br>`segmentPoses = interpolate(path,samples);` |

**Corrections to Image Width and Image Height camera parameters of Driving Scenario Designer**
*Behavior change*

Starting in R2018b, in the **Camera Settings** group of the **Driving Scenario Designer** app, the **Image Width** and **Image Height** parameters set their expected values. Previously, **Image Width** set the height of images produced by the camera, and **Image Height** set the width of images produced by the camera.

If you are using R2018a, to produce the expected image sizes, transpose the values set in the **Image Width** and **Image Height** parameters.

# R2018a

**Version: 1.2**

**New Features**

**Compatibility Considerations**

## Driving Scenario Designer: Interactively define actors and driving scenarios to test controllers and sensor fusion algorithms

Use the **Driving Scenario Designer** app to design a synthetic driving scenario composed of roads and actors (vehicles, pedestrians, and so on). You can generate visual and radar detections of actors in the scenario to test your sensor fusion and control algorithms. To learn how to generate detections, see Generate Synthetic Detections from an Interactive Driving Scenario.

## Path Planning: Plan driving paths using an RRT* path planner and costmap

Use the `pathPlannerRRT`, `vehicleCostmap`, and `checkPathValidity` functions to plan a driving path by using an optimal rapidly exploring random tree (RRT*) motion-planning algorithm. To learn how to use these functions to plan a path, see the Automated Parking Valet example.

## Streaming Geographic Map Display: Visualize a geographic route on a map

Use the `geoplayer` function to create an interactive map that displays the streaming geographic coordinates of a driving route.

## Ground Truth Pixel Labeling: Interactively label individual pixels in video data

In the **Ground Truth Labeler** app, you can now interactively label individual pixels in video data for training semantic segmentation algorithms. You can also automate the labeling. See Automate Ground Truth Labeling for Semantic Segmentation.

## Ground Truth Label Attributes: Organize and classify ground truth labels using attributes and sublabels

In the **Ground Truth Labeler** app, you can now attach attributes to labels and create hierarchical sublabels. For more details, see Define Ground Truth Data for Video or Image Sequences.

## Lidar Segmentation: Quickly segment 3-D point clouds from lidar

Use the `segmentLidarData` function to segment organized point clouds into clusters.

## ACC Reference Application: Use a reference model to simulate and test adaptive cruise controller (ACC) systems

The ACC reference application is a model of an ACC system implemented using sensor fusion. Use this model to design your own ACC system, test it in Simulink® using synthetic radar and vision data generated by Automated Driving System Toolbox blocks, and automatically generate C code. To learn more, see Adaptive Cruise Control with Sensor Fusion.

## Point Cloud Reader for Velodyne PCAP Files: Import Velodyne lidar data into MATLAB

Use a `velodyneFileReader` object to read point cloud data from Velodyne® packet capture (PCAP) files.

## Detect lanes more precisely by using third-degree polynomial lane boundary models

Use the `cubicLaneBoundary` and `findCubicLaneBoundaries` functions to create and find lane boundaries using third-degree polynomial models. You can display detected lanes on a bird's-eye-view plot, and overlay the lane markings onto images, by using the `insertLaneBoundary` function.

## Add and detect lanes in Driving Scenario

You can add lane markings to roads in a driving scenario simulation using the new lane marking function, `laneMarking`, and lane specification function, `lanespec`. The driving scenario `road` method accepts a lane specification as an input. To plot lane markings in `birdsEyePlot`, use `laneMarkingPlotter` and `plotLaneMarking`.

In addition, the vision detection generator System object, `visionDetectionGenerator`, can now detect lanes in a driving scenario simulation. The corresponding Simulink block, Vision Detection Generator, can also detect lanes.

## Transform [x,y,z] locations in vehicle coordinates to image coordinates

The `vehicleToImage` method of `monoCamera` now accepts three-dimensional [*x*,*y*,*z*] point coordinates. Previously, `vehicleToImage` accepted only [*x*,*y*] coordinates. By transforming [*x*,*y*,*z*] locations in vehicle coordinates, you can display point locations above the road surface.

## Path method being removed

The `path` method of the `actor` and `vehicle` classes is being removed. Use the `trajectory` method instead.

## Compatibility Considerations

| Functionality | Result | Use Instead | Compatibility Considerations |
|---|---|---|---|
| `path` method | Still runs | `trajectory` method | Replace all instances of `path` with `trajectory`. The `path` syntax which assumes a default speed does not exist in `trajectory`. You must specify a speed input argument. |

## Direction of Yaw Angle Rotation Adjusted

The `monoCamera` function was updated to correct the direction of rotation for the yaw angle.

## Compatibility Considerations

| Functionality | Compatibility Considerations |
| --- | --- |
| `monoCamera` function | If you are using R2017b version of this function, you must multiply the yaw angle by `-1`. |

# R2017b

**Version: 1.1**

**New Features**

## Sensor Fusion Simulink Blocks: Track multiple objects and fuse detections from multiple sensors

Use the Detection Concatenation block and the Multi Object Tracker block to fuse and track objects detected by multiple sensors.

## Sensor Simulation Using Simulink Blocks: Generate synthetic object lists from camera and radar sensor models

Use the Radar Detection Generator and the Vision Detection Generator blocks to generate synthetic detections for testing and design of your sensor fusion and tracking algorithms

## Ground Truth Labeling App: Reverse playback capability while processing algorithms

In the **Ground Truth Labeler** app, you can now process the video in reverse using the automation algorithm. You can also now dock and undock the Visual Summary display.

## Code Generation for Sensor Models: Generate C code for camera and radar sensor models

Use the `radarDetectionGenerator` and `visionDetectionGenerator` System objects to generate C code to generate synthetic sensor detection object lists.

## Autonomous Driving Examples

- Sensor Fusion Using Synthetic Radar and Vision Data
- Adaptive Cruise Control with Sensor Fusion
- Evaluate and Visualize Lane Boundary Detections Against Ground Truth
- Radar Signal Simulation and Processing for Automated Driving

# R2017a

**Version: 1.0**

**New Features**

## Ground Truth Labeling

The **Ground Truth Labeler** app enables you to label ground truth data in a video or in a sequence of images. Use the app to interactively specify rectangular and polyline regions of interest (ROIs), and scene labels. 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. It also provides an API and workflow that enables you to import your own algorithms to automate the labeling of ground truth. You can also use the `driving.connector.Connector` API to display additional time-synchronized signals, such as lidar or CAN bus data.

| Ground Truth Labeling Utilities | Description |
|---|---|
| **Ground Truth Labeler** | App for labeling ground truth data in a video or sequence of images |
| `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 |
| `driving.automation.AutomationAlgorithm` | Define automated labeling algorithm in the Ground Truth Labeler app |
| `driving.connector.Connector` | Interface to connect an external tool to the Ground Truth Labeler app |
| `evaluateLaneBoundaries` | Evaluate lane boundary models against ground truth |

## Monocular Camera Sensor Configuration

Use the `monoCamera` object to define your monocular camera configuration. You can use this object to convert locations between vehicle and image coordinate systems. You can also use `birdsEyeView` with the `monoCamera` object to create a bird’s-eye-view image.

## Object and Lane Boundary Detection

Detect objects using machine learning techniques, including deep learning. You can also segment, detect, and model parabolic lane boundaries using RANSAC. Configure object

detectors to detect objects of a known physical size using the `configureDetectorMonoCamera` function.

**Object Detection**

- `vehicleDetectorACF`
- `vehicleDetectorFasterRCNN`
- `peopleDetectorACF`
- `configureDetectorMonoCamera`
- `acfObjectDetectorMonoCamera`
- `objectDetectorTrainingData`
- `fastRCNNObjectDetectorMonoCamera`
- `fasterRCNNObjectDetectorMonoCamera`

**Lane Boundary Detection**

- `segmentLaneMarkerRidge`
- `findParabolicLaneBoundaries`
- `parabolicLaneBoundary`
- `insertLaneBoundary`
- `evaluateLaneBoundaries`
- `fitPolynomialRANSAC`
- `ransac`

## Multi-object Tracking

You can create a multi-object tracker for sensor fusion. The tracker uses Kalman filters for estimating the state of motion of an object. Measurements made on the object let you continuously solve for the object's position and velocity. You can use constant-velocity or constant-acceleration motion models, or define your own models.

- `multiObjectTracker`
- `objectDetection`
- `getTrackPositions`
- `getTrackVelocities`

- `trackingKF`
- `trackingEKF`
- `trackingUKF`

## Bird’s-Eye Plot

Use `birdsEyePlot` to display a bird's-eye plot of a 2-D scene in the immediate vicinity of a vehicle. You can use bird's-eye plots with sensors capable of detecting objects and lanes.

## Driving Scenario Generation and Sensor Models

The `drivingScenario` class defines road networks, vehicles, and traffic scenarios. A driving scenario is a 3-D arena containing roads and actors. Actors can represent anything that moves, such as cars, pedestrians, and bicycles. Actors can also include stationary obstacles that can influence the motion of other actors. You can use `radarDetectionGenerator` and the `visionDetectionGenerator` to create statistical models for generating synthetic radar and camera sensor detections.

## Automated Driving Examples

The release of Automated Driving System Toolbox includes the following examples.

| **Reference Applications** |
| --- |
| “Visual Perception Using Monocular Camera” |
| “Forward Collision Warning Using Sensor Fusion” |
| “Sensor Fusion Using Synthetic Radar and Vision Data” |

| **Tracking and Sensor Fusion** |
| --- |
| “Forward Collision Warning Using Sensor Fusion” |
| “Track Multiple Vehicles Using a Camera” |
| “Track Pedestrians from a Moving Car” |
| “Multiple Object Tracking Tutorial” |
| “Code Generation for Tracking and Sensor Fusion” |

| Perception with Computer Vision |
| --- |
| “Visual Perception Using Monocular Camera” |
| “Ground Plane and Obstacle Detection Using Lidar” |
| “Train a Deep Learning Vehicle Detector” |

| Algorithm Validation and Visualization |
| --- |
| “Automate Ground Truth Labeling of Lane Boundaries” |
| “Annotate Video Using Detections in Vehicle Coordinates” |
| “Visualize Sensor Coverage, Detections, and Tracks” |
| “Evaluate Lane Boundary Detections Against Ground Truth Data” |

| Scenario Generation |
| --- |
| “Sensor Fusion Using Synthetic Radar and Vision Data” |
| “Driving Scenario Tutorial” |
| “Define Road Layouts” |
| “Create Actor and Vehicle Trajectories” |
| “Model Radar Sensor Detections” |
| “Model Vision Sensor Detections” |