

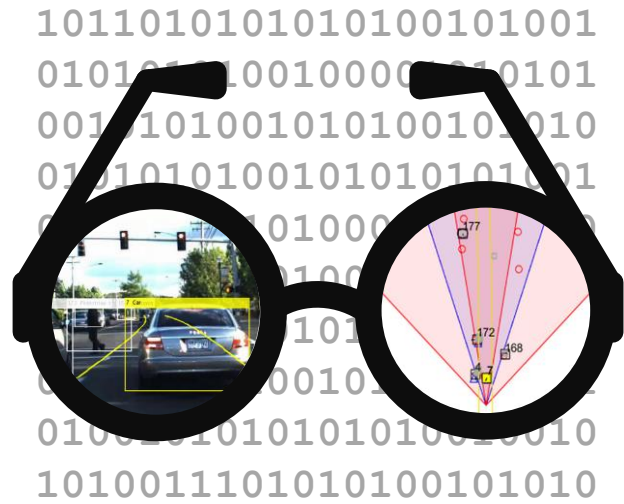
Introduction to Automated Driving System Toolbox: **Design and Verify Perception Systems**

MATLAB Expo Germany 2017

Anders Sollander
Principal Consultant



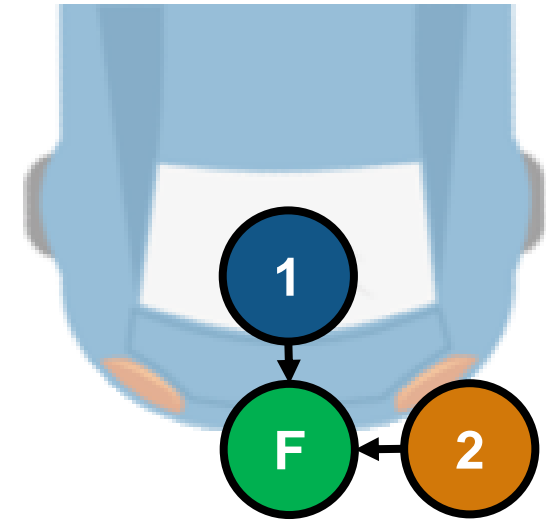
Some common questions from automated driving engineers



**How can I
visualize vehicle
data?**

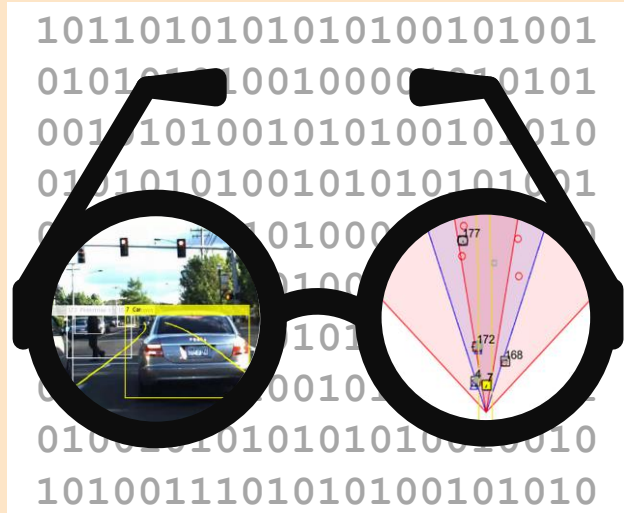


**How can I
detect objects in
images?**

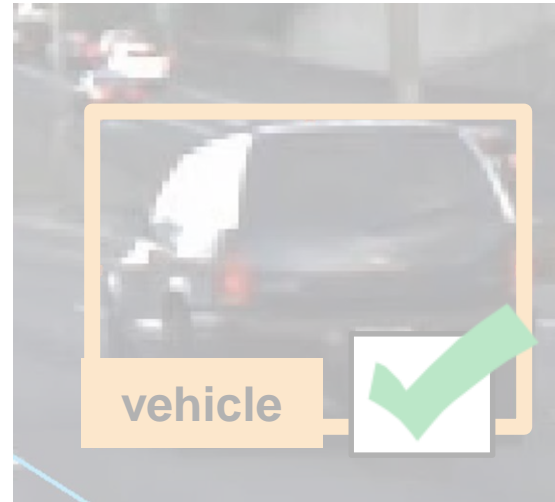


**How can I
fuse multiple
detections?**

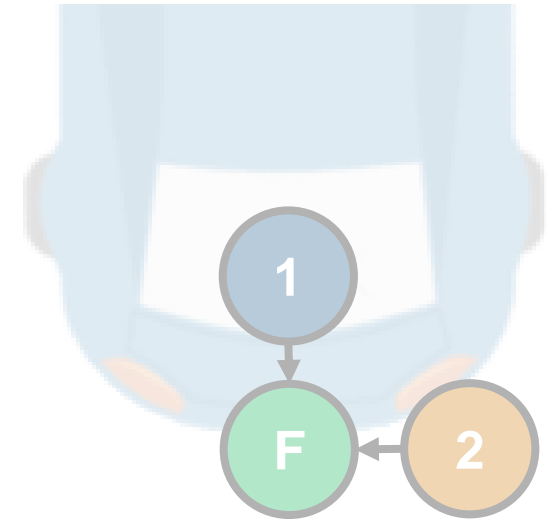
Some common questions from automated driving engineers



**How can I
visualize vehicle
data?**



**How can I
detect objects in
images?**



**How can I
fuse multiple
detections?**

Examples of automated driving sensors

Camera

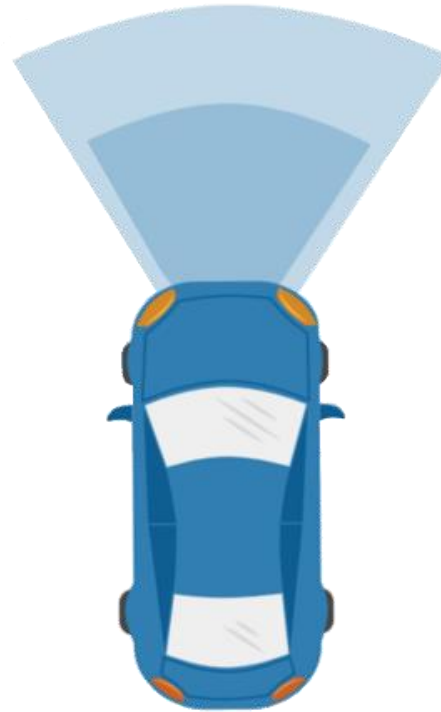
Radar-based
object detector

Vision-based
object detector

Lidar

Lane detector

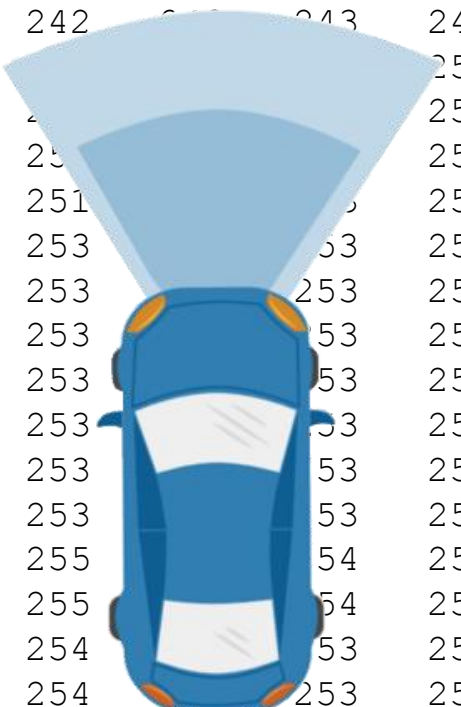
Inertial
measurement
unit



Examples of automated driving sensor data

Camera (640 x 480 x 3)

239	239	237	238	241	241	241	242	243	243
252	252	251	252	252	253	253	253	255	255
254	254	253	253	254	253	255	255	255	255
250	251	251	251	251	251	253	253	253	253
251	252	251	253	253	253	251	251	253	253
252	253	253	254	254	253	253	253	253	253
252	253	253	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
255	255	255	255	255	255	255	255	254	254
253	254	255	255	255	255	254	254	253	253
253	255	255	255	255	255	254	254	253	253
254	254	254	254	254	254	253	253	253	253
254	254	254	254	254	254	253	253	253	253
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251
253	253	253	253	253	253	251	251	251	251



Radar-based object detector

253	254	253	253	253	253	253	253	253	253
253	253	253	254	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253
253	253	253	253	253	253	253	253	253	253

Lane detector

Inertial measurement unit

Lidar

Examples of automated driving sensor data

Camera (640 x 480 x 3)

```
239 239 237 238 241 241 241 242 243 243
252 252 251 252 252 253 253
```

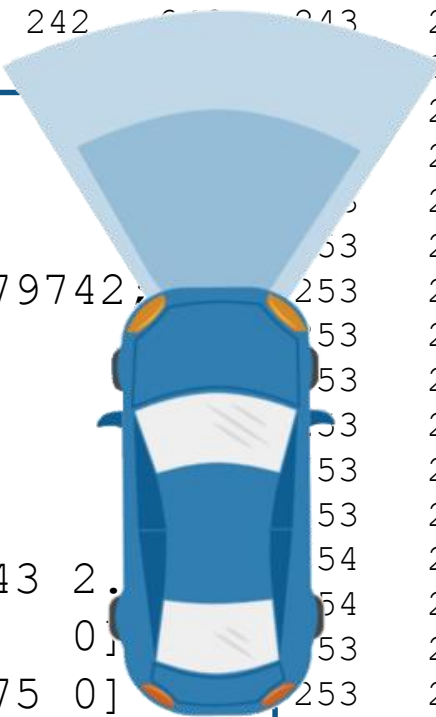
Vision Detector

```
25 SensorID = 1;
25 Timestamp = 1461634696379742;
25 NumDetections = 6;
```

```
25 Detections(1)
25 TrackID: 0
25 Classification: 5
25 Position: [22.61 -0.43 2.
25 Velocity: [-9.86 0 0]
25 Size: [0 1.75 0]
```

```
25 Detection
25 Track
25 Class
25 Position: [2 2.24]
25 Velocity: [0 0 0]
25 Size: [0 1.8 0]
```

Lane detector



Radar-based object detector

Lidar

Inertial measurement unit

Examples of automated driving sensor data

Camera (640 x 480 x 3)

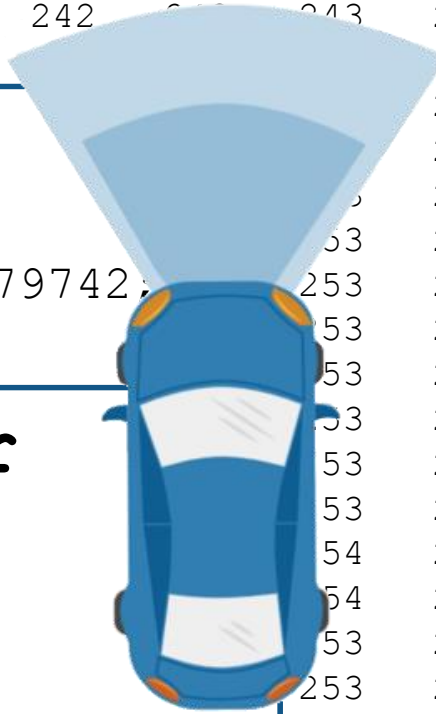
```
239 239 237 238 241 241 241 242 243 243
252 252 251 252 252 253 253
```

Vision Detector

```
25 SensorID = 1;
25 Timestamp = 1461634696379742;
25 NumDetections = 6;
```

Lane Detector

```
25 Detect Left
25 Tr Cl Po Ve Si
25 IsInvalid: 1
25 Confidence: 3
25 BoundaryType: 3
25 Offset: 1.68
25 HeadingAngle: 0.002
25 Curvature: 0.0000228
25 Right
25 Ve Si
25 IsInvalid: 1
25 Confidence: 3
```



**Radar-based
object detector**

Lidar

**Inertial
measurement
unit**

Examples of automated driving sensor data

Camera (640 x 480 x 3)

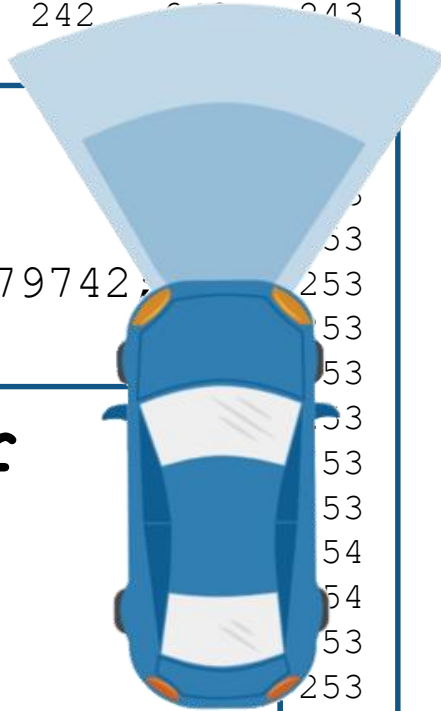
```
239 239 237 238 241 241 241 242 243
252 252 251 252 252 253 253
```

Vision Detector

```
25 SensorID = 1;
25 Timestamp = 1461634696379742;
25 NumDetections = 6;
```

Lane Detector

```
25 Tr Left
25 Cl
25 Po IsValid: 1
25 Ve Confidence: 3
25 Si BoundaryType: 3
25 Detec Offset: 1.68
25 Tr HeadingAngle: 0.002
25 Cl Curvature: 0.0000228
25 Po Right
25 Ve IsValid: 1
25 Si Confidence: 3
```



Radar Detector

```
SensorID = 2;
Timestamp = 1461634696407521;
NumDetections = 23;
Detections (1)
  TrackID: 0
  TrackStatus:
  Position:
  Velocity:
  Amplitude:
Detections (2)
  TrackID: 1
  TrackStatus: 6
  Position: [19.59 0.34]
  Velocity: [4.92 0]
  Amplitude:
Detections (3)
  TrackID: 12
  TrackStatus: 5
```

Lidar

Inertial measurement unit

Examples of automated driving sensor data

Camera (640 x 480 x 3)

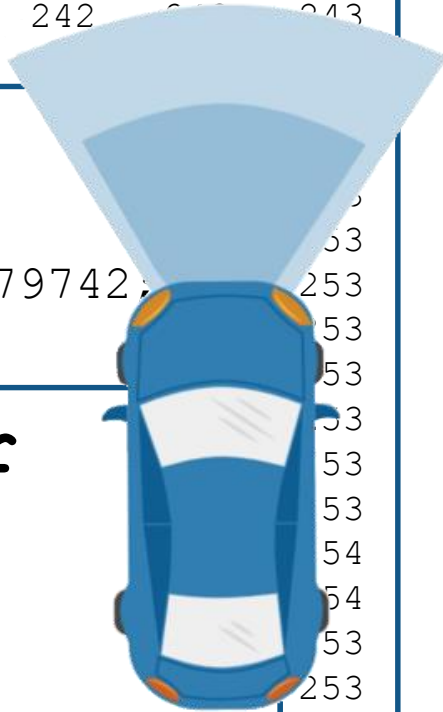
```
239 239 237 238 241 241 241 242 243
252 252 251 252 252 253 253
```

Vision Detector

```
25 SensorID = 1;
25 Timestamp = 1461634696379742;
25 NumDetections = 6;
```

Lane Detector

```
25 Detect Left
25 Tr
25 Cl
25 Po IsValid: 1
25 Ve Confidence: 3
25 Si BoundaryType: 3
25 Detect Offset: 1.68
25 Tr HeadingAngle: 0.002
25 Cl Curvature: 0.0000228
25 Po Right
25 Ve IsValid: 1
25 Si Confidence: 3
```



Radar Detector

```
SensorID = 2;
Timestamp = 1461634696407521;
NumDetections = 23;
```

Lidar (47197 x 3)

```
Detection
TrackID
TrackSt -12.2911 1.4790 -0.59
Positio -14.8852 1.7755 -0.64
Velocit -18.8020 2.2231 -0.73
Amplitu -25.7033 3.0119 -0.92
Detection -0.0632 0.0815 1.25
TrackID -0.0978 0.0855 1.25
TrackSt -0.2814 0.1064 1.25
Po 0.1129 1.26
Ve 0.1270 1.25
Am 0.1450 1.24
Detection 0.1699 1.23
TrackID -14.8815 1.8245 -0.64
TrackSt -18.8008 2.2849 -0.74
```

**Inertial
measurement
unit**

Examples of automated driving sensor data

Camera (640 x 480 x 3)

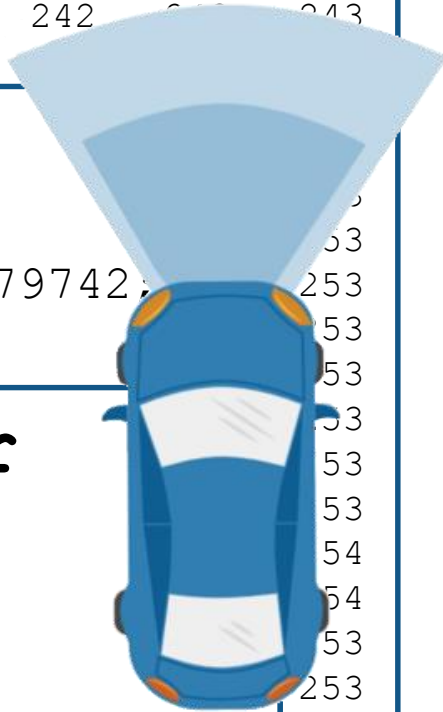
```
239 239 237 238 241 241 241 242 243
252 252 251 252 252 253 253
```

Vision Detector

```
25 SensorID = 1;
25 Timestamp = 1461634696379742;
25 NumDetections = 6;
```

Lane Detector

```
25 Detect Left
25 Tr
25 Cl
25 Po IsValid: 1
25 Ve Confidence: 3
25 Si BoundaryType: 3
25 Detect Offset: 1.68
25 Tr HeadingAngle: 0.002
25 Cl Curvature: 0.000
25 Po Right
25 Ve IsValid: 1
25 Si Confidence: 3
```



Radar Detector

```
SensorID = 2;
Timestamp = 1461634696407521;
NumDetections = 23;
```

Detection

Lidar

```
(47197 x 3)
TrackID
TrackSt -12.2911 1.4790 -0.59
Positio -14.8852 1.7755 -0.64
Velocit -18.8020 2.2231 -0.73
Amplitu -25.7033 3.0119 -0.92
Detection -0.0632 0.0815 1.25
TrackID -0.0978 0.0855 1.25
TrackSt -0.2814 0.1064 1.25
```

Inertial Measurement Unit

```
Timestamp: 1461634696379742
Velocity: 9.2795
YawRate: 0.0040
```

Visualize sensor data

Image Coordinates

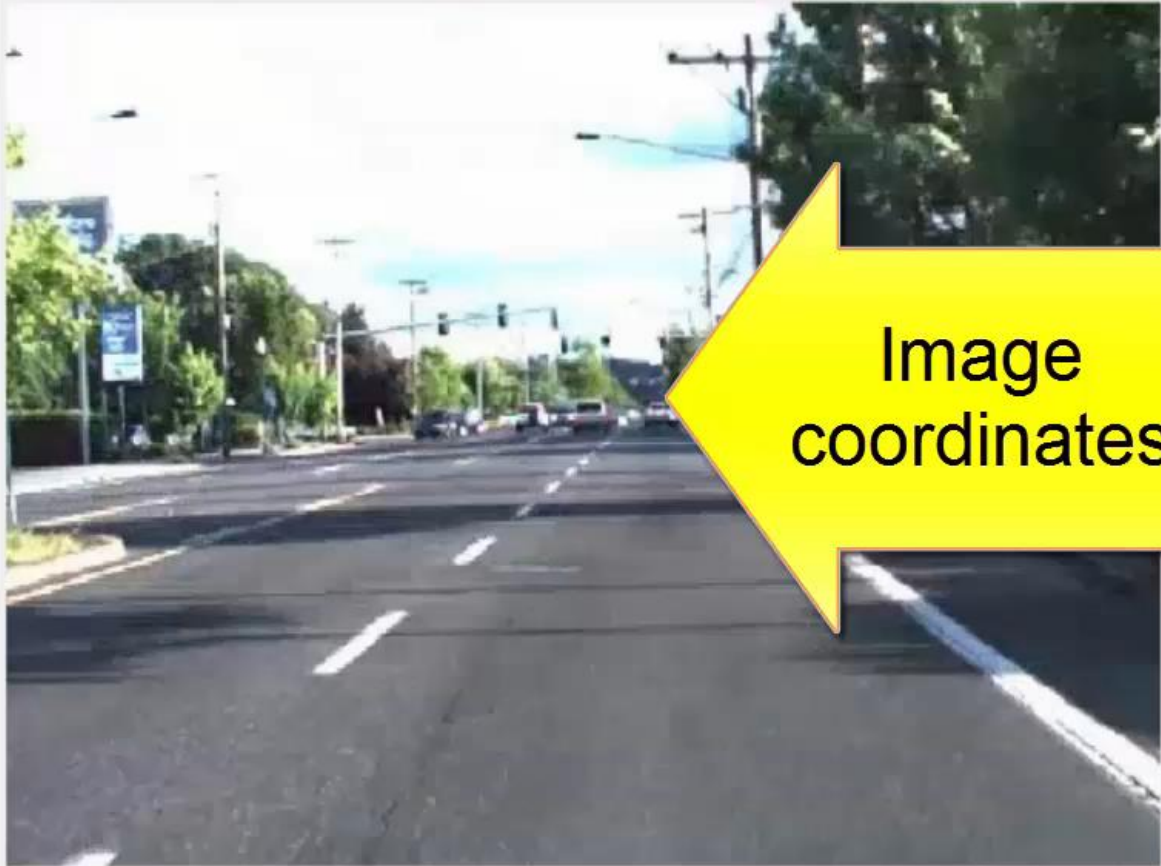
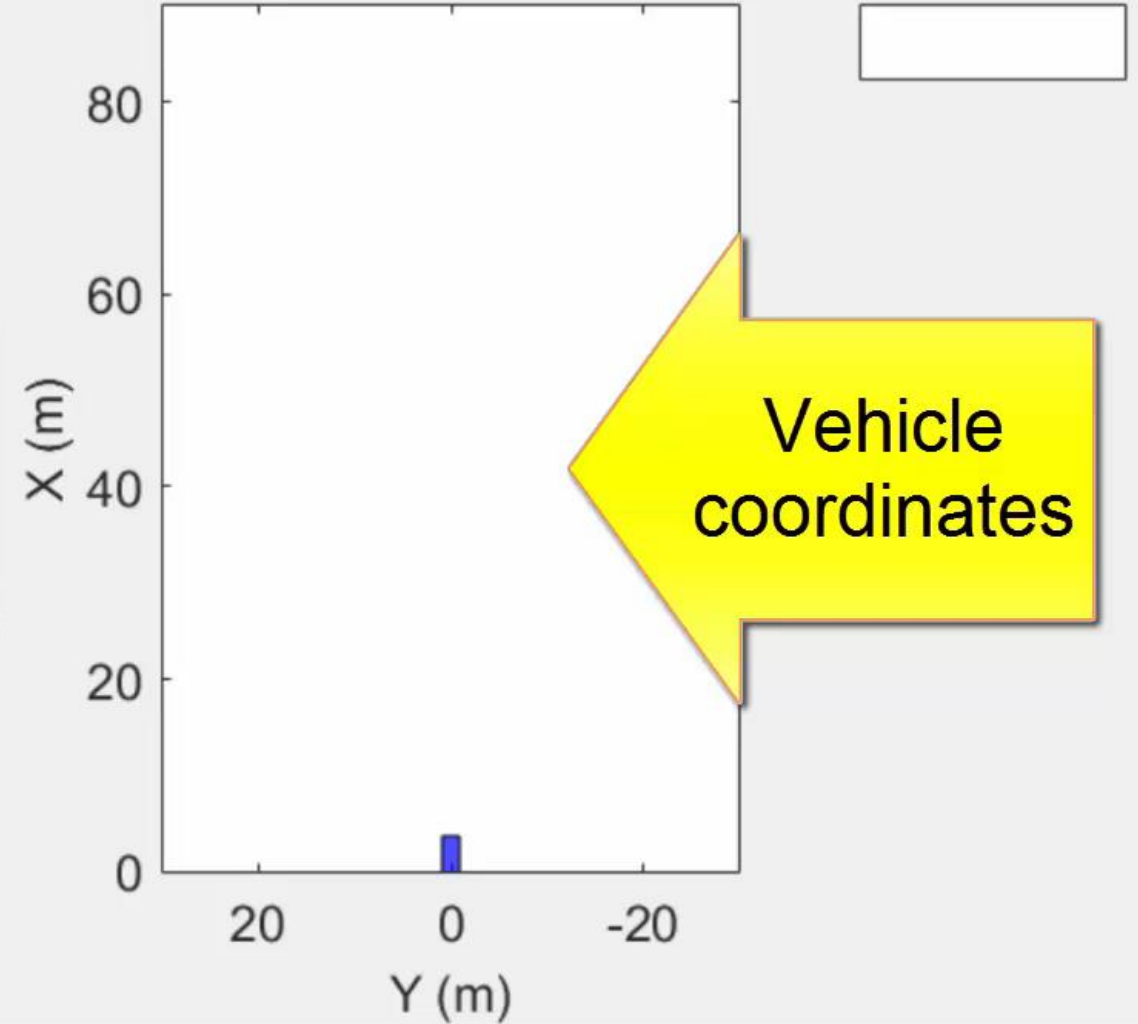


Image
coordinates

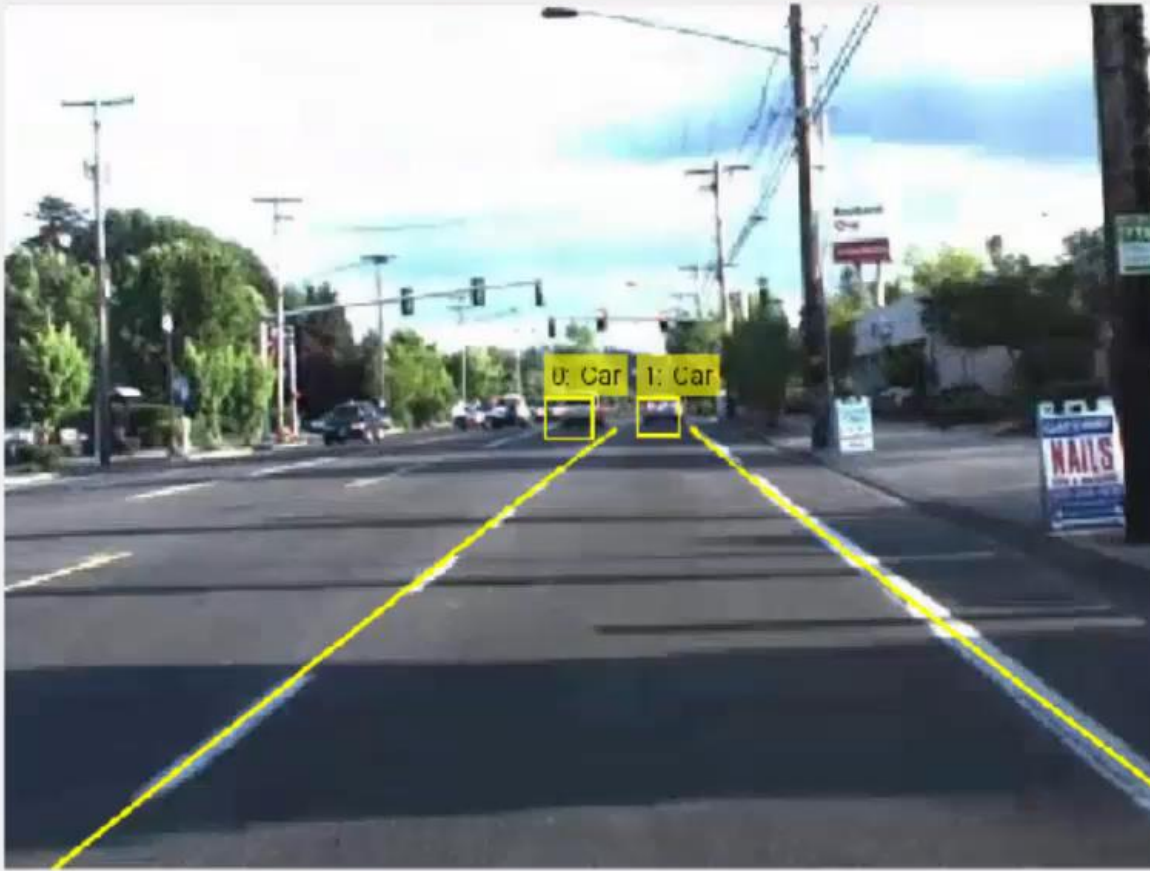
Vehicle Coordinates



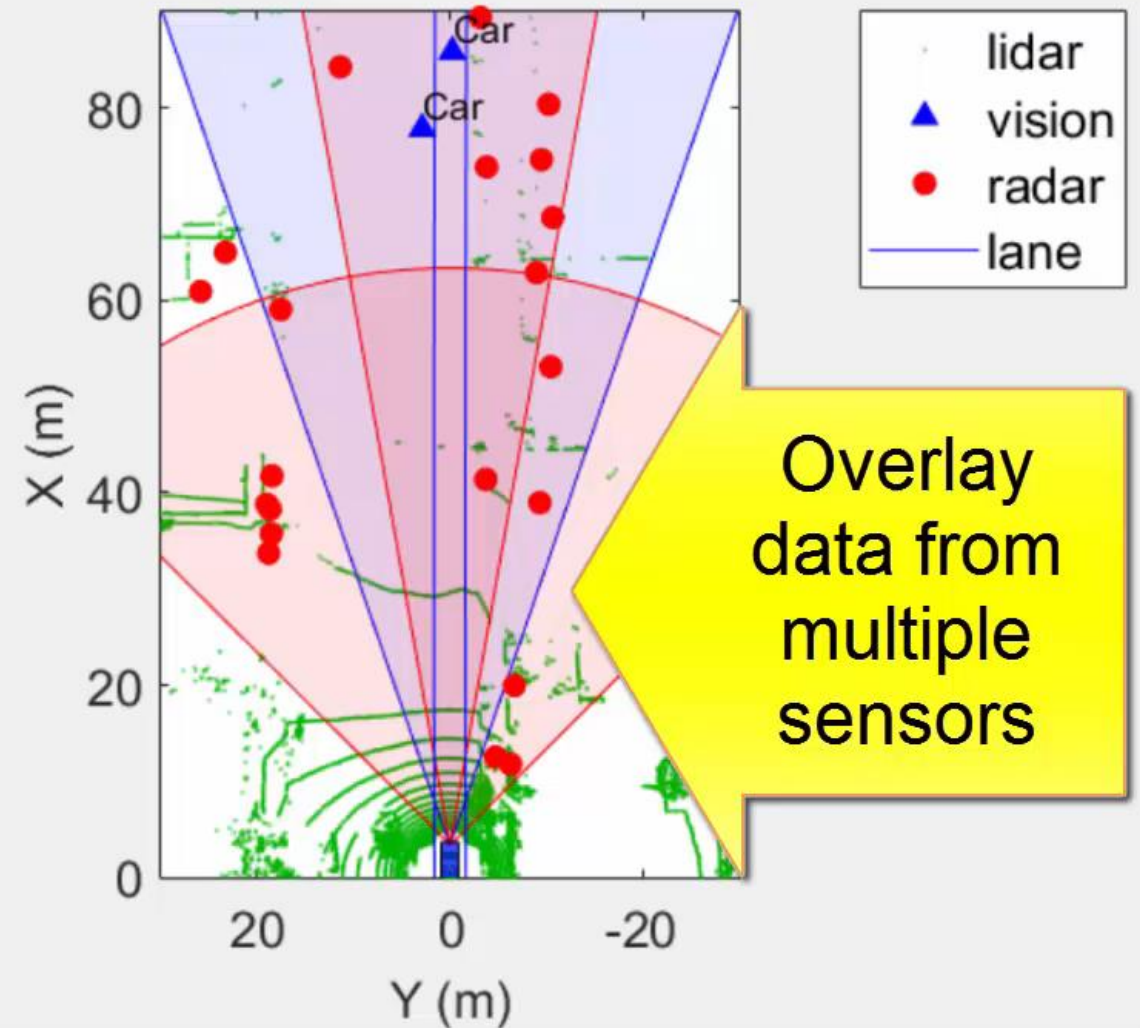
Vehicle
coordinates

Visualize differences in sensor detections

Image Coordinates



Vehicle Coordinates



Explore logged vehicle data

- Load **video data** and corresponding **mono-camera parameters**

```
>> video = VideoReader('01_city_c2s_fcw_10s.mp4')  
>> load('FCWDemoMonoCameraSensor.mat', 'sensor')
```

- Load **detection sensor data** and corresponding **parameters**

```
>> load('01_city_c2s_fcw_10s_sensor.mat', 'vision', 'lane', 'radar')  
>> load('SensorConfigurationData.mat', 'sensorParams')
```

- Load **lidar point cloud data**

```
>> load('01_city_c2s_fcw_10s_Lidar.mat', 'LidarPointCloud')
```

Visualize in image coordinates

```
%% Specify time to inspect
currentTime = 6.55;
video.CurrentTime = currentTime;

%% Extract video frame
frame = video.readFrame;

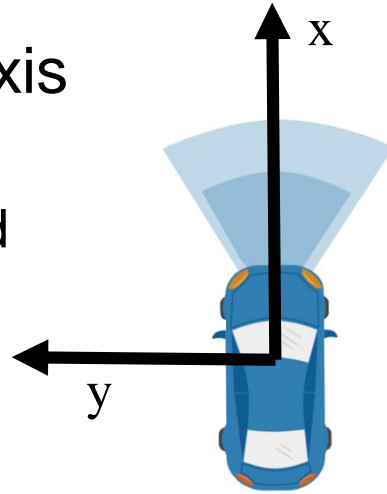
%% Plot image coordinates
ax1 = axes(...
    'Position', [0.02 0 0.55 1]);
im = imshow(frame, ...
    'Parent', ax1);
```



Plot in image coordinates using
“classic” video and image functions like
imshow

Visualize in vehicle coordinates

- ISO 8855 vehicle axis coordinate system
 - Positive x is forward
 - Positive y is left



```

%% Plot in vehicle coordinates
ax2 = axes(...
    'Position',[0.6 0.12 0.4 0.85]);
bep = birdsEyePlot(...
    'Parent',ax2,...
    'Xlimits',[0 45],...
    'Ylimits',[-10 10]);
legend('off');
  
```



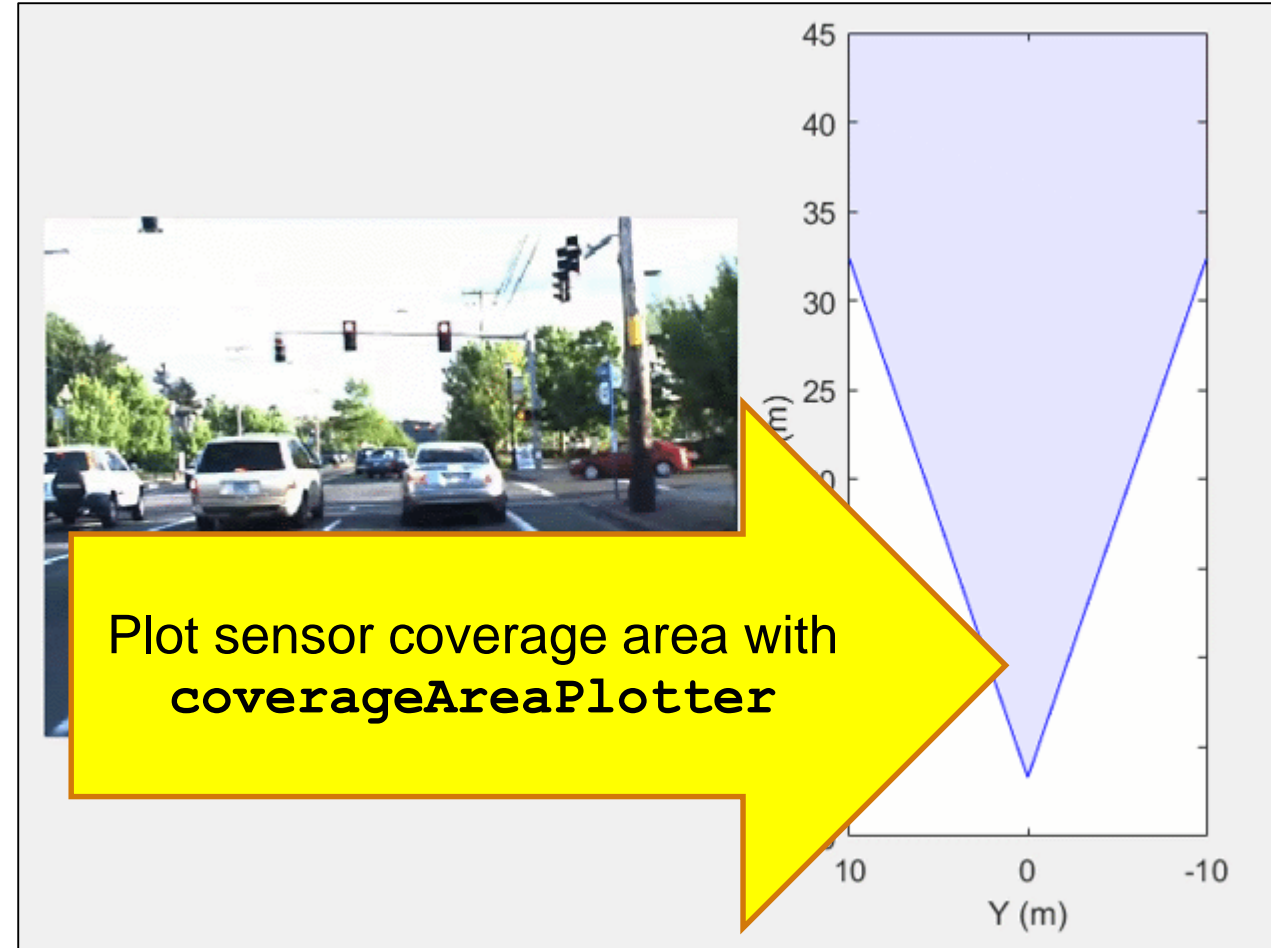
Visualize expected coverage area (vehicle coordinates)

```

%% Create coverage area plotter
covPlot = coverageAreaPlotter(bep, ...
    'FaceColor', 'blue', ...
    'EdgeColor', 'blue');

%% Update coverage area plotter
plotCoverageArea(covPlot, ...
    [sensorParams(1).X ... % Position x
     sensorParams(1).Y], ... % Position y
    sensorParams(1).Range, ...
    sensorParams(1).YawAngle, ...
    sensorParams(1).FoV(1)) % Field of view

```



Visualize detections (vehicle coordinates)

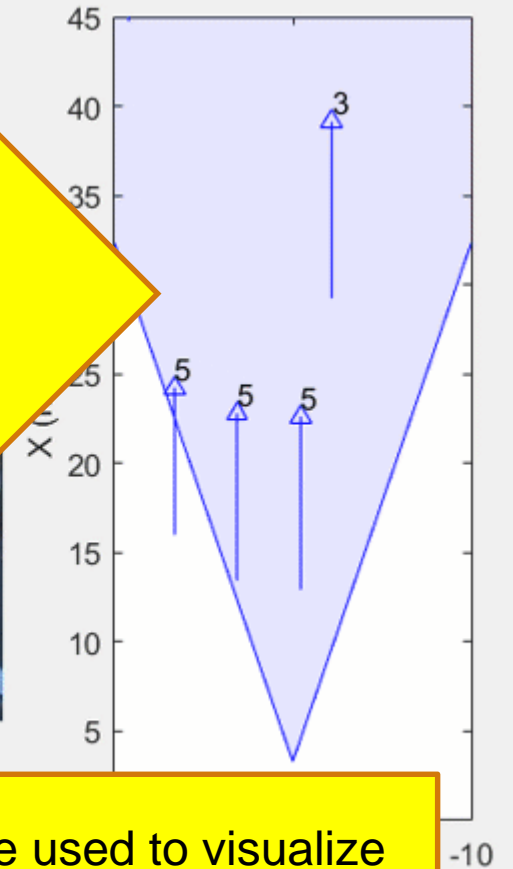
```

%% Create detection plotter
detPlot = detectionPlotter(bep, ...
    'MarkerEdgeColor','blue',...
    'Marker','^');

%% Update detection plotter
n = round(currentTime/0.05);
numDets = vision(n).numObjects;
pos = zeros(numDets,3);
vel = zeros(numDets,3);
labels = repmat({''},numDets,1);
for k = 1:numDets
    pos(k,:) = vision(n).object(k).position;
    vel(k,:) = vision(n).object(k).velocity;
    labels{k} = num2str(...
        vision(n).object(k).classification);
end
plotDetection(detPlot,pos,vel,labels);

```

Plot vision detections with
detectionPlotter



detectionPlotter can be used to visualize
vision detector, radar detector, and
lidar point cloud

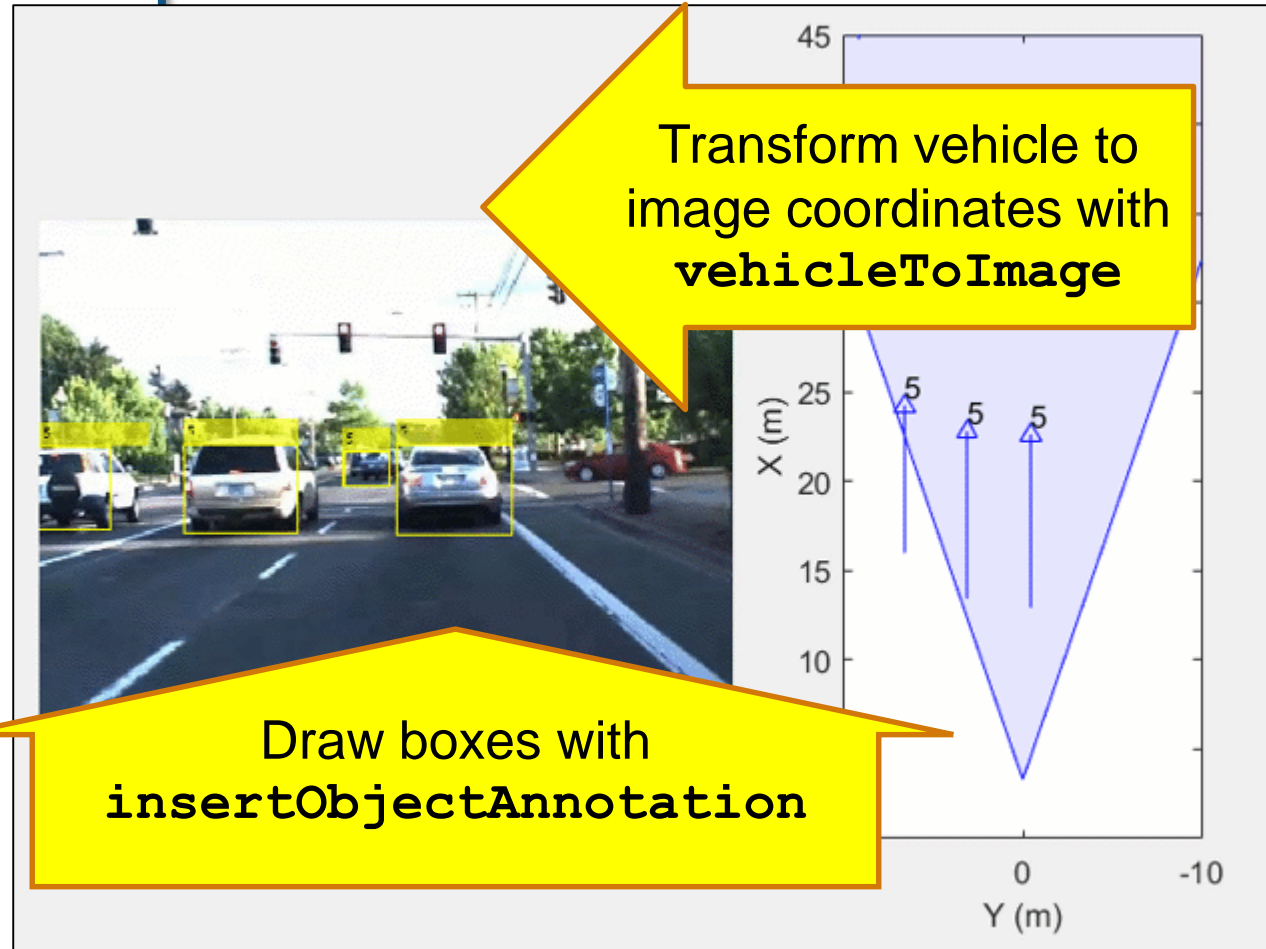
Visualize detections (image coordinates)

```

%% Bounding box positions in image coordinates
imBoxes = zeros(numDets,4);
for k = 1:numDets
    if vision(n).object(k).classification == 5
        vehPosLR = vision(n).object(k).position(1:2)';
        imPosLR = vehicleToImage(sensor, vehPosLR);
        boxHeight = 1.4 * 1333 / vehPosLR(1);
        boxWidth = 1.8 * 1333 / vehPosLR(1);
        imBoxes(k,:) = [imPosLR(1) - boxWidth/2, ...
                       imPosLR(2) - boxHeight, ...
                       boxWidth, boxHeight];
    end
end

%% Draw bounding boxes on image frame
frame = insertObjectAnnotation(frame, ...
    'Rectangle', imBoxes, labels, ...
    'Color', 'yellow', 'LineWidth', 2);
im.CData = frame;

```



Visualize lane boundaries (vehicle coordinates)

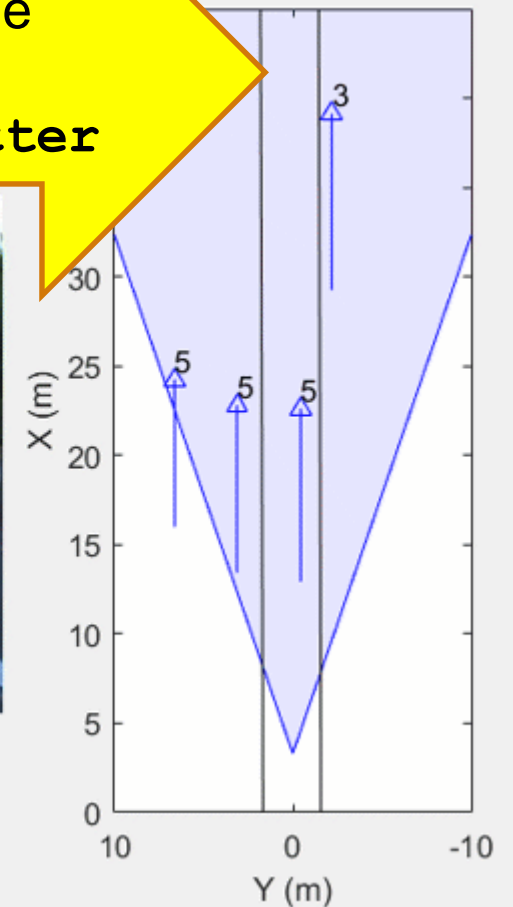
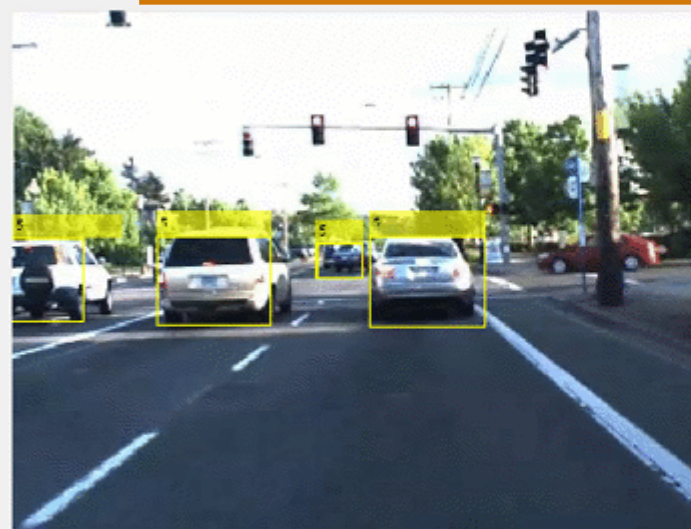
```

%% Create lane detection plotter
lanePlot = laneBoundaryPlotter(bep, ...
    'Color', 'black');

%% Update lane detection plotter
lb = parabolicLaneBoundary([...
    lane(n).left.curvature, ...
    lane(n).left.headingAngle, ...
    lane(n).left.offset]);
rb = parabolicLaneBoundary([...
    lane(n).right.curvature, ...
    lane(n).right.headingAngle, ...
    lane(n).right.offset]);
plotLaneBoundary(lanePlot, [lb rb])

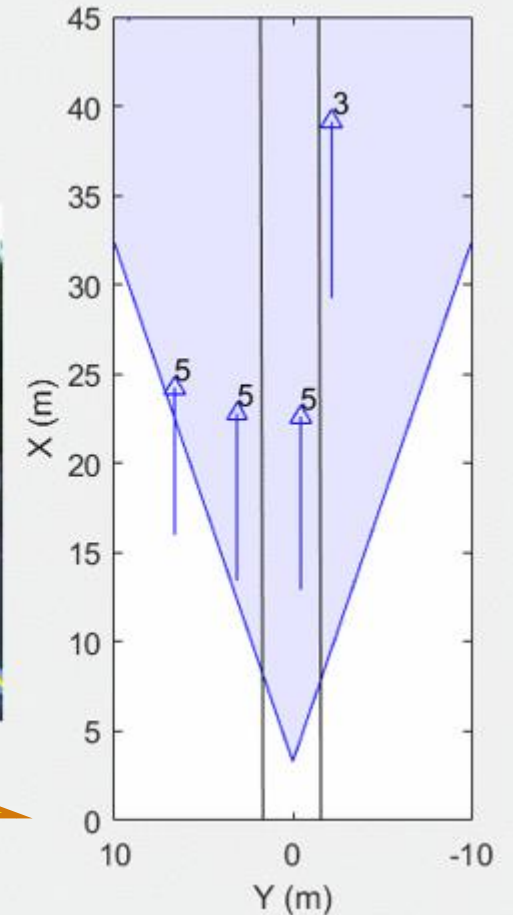
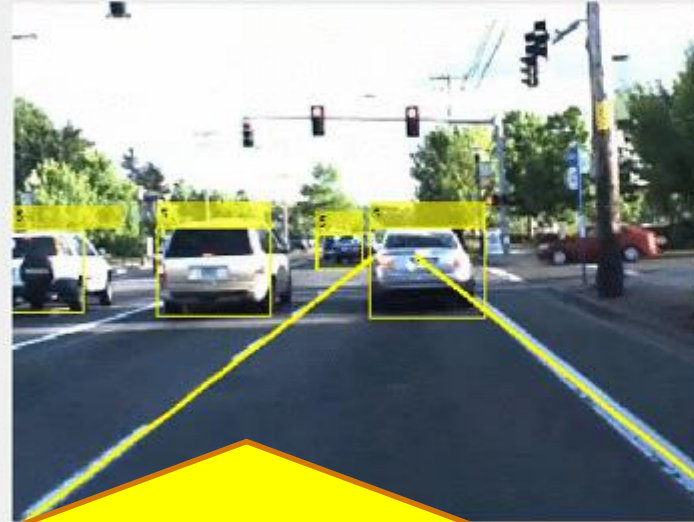
```

Plot lanes in vehicle coordinates with `laneBoundaryPlotter`



Visualize lane boundaries (image coordinates)

```
%% Draw in image coordinates
frame = insertLaneBoundary(frame, ...
    [lb rb], sensor, (1:100), ...
    'LineWidth',5);
im.CData = frame;
```



Plot lanes in image coordinates with `insertLaneBoundary`

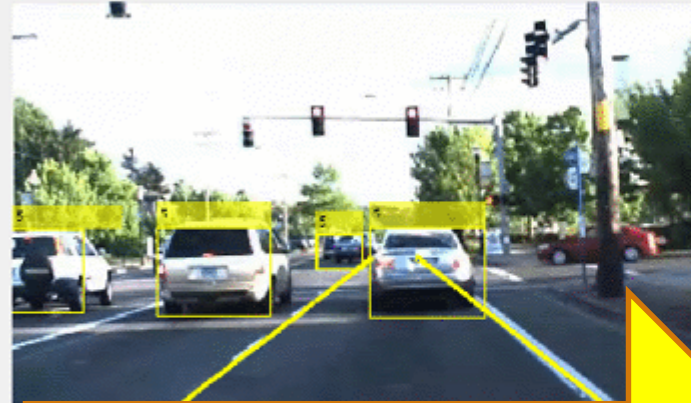
Visualize radar detections (vehicle coordinates)

```

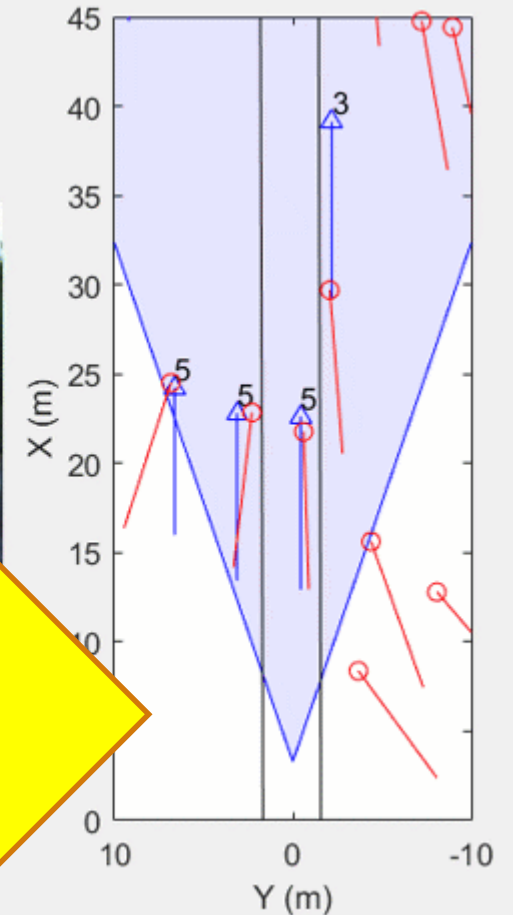
%% Create radar detection plotter
radarPlot = detectionPlotter(bep, ...
    'MarkerEdgeColor','red',...
    'Marker','o');

%% Update radar detection plotter
numDets = radar(n).numObjects;
pos = zeros(numDets,3);
vel = zeros(numDets,3);
for k = 1:numDets
    pos(k,:) = radar(n).object(k).position;
    vel(k,:) = radar(n).object(k).velocity;
end
plotDetection(radarPlot,pos,vel);

```



Plot radar detections just like vision detections with **detectionPlotter**

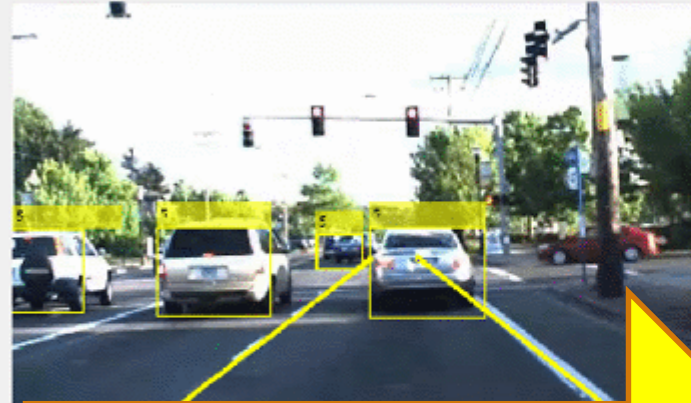


Visualize lidar point cloud (vehicle coordinates)

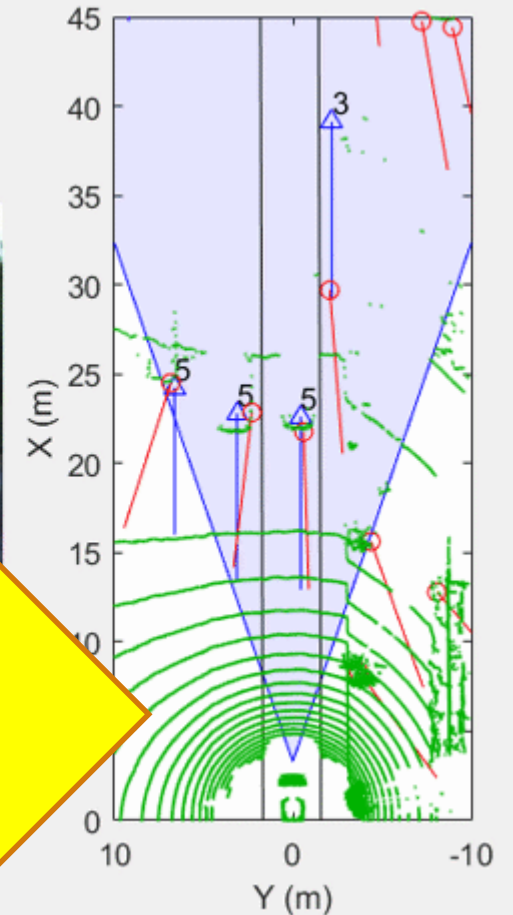
```

%% Create lidar detection plotter
lidarPlot = detectionPlotter(bep, ...
    'Marker','.',...
    'MarkerSize',1.5,...
    'MarkerEdgeColor',[0 0.7 0]); % Green

%% Update lidar detection plotter
n = round(video.CurrentTime/0.1);
pos = ...
    LidarPointCloud(n).ptCloud.Location(:,1:2);
plotDetection(lidarPlot,pos);
  
```

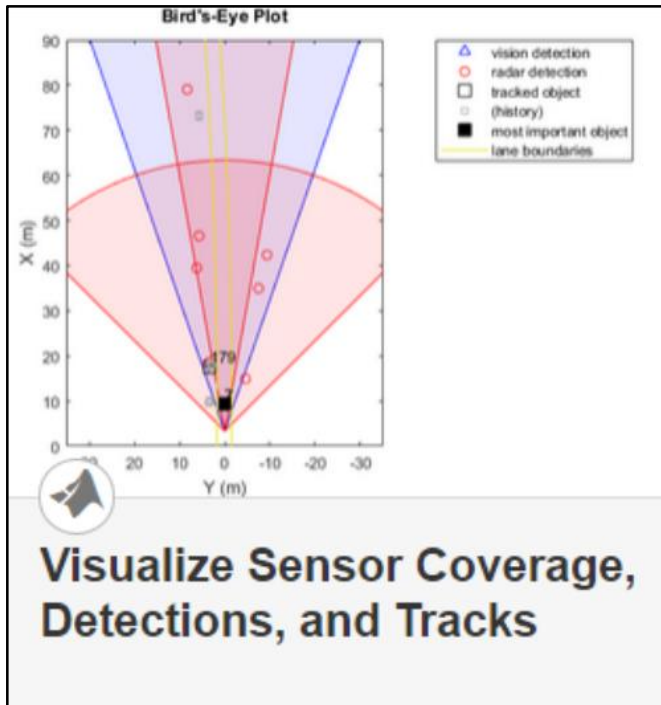


Plot lidar points just like
vision detections with
detectionPlotter

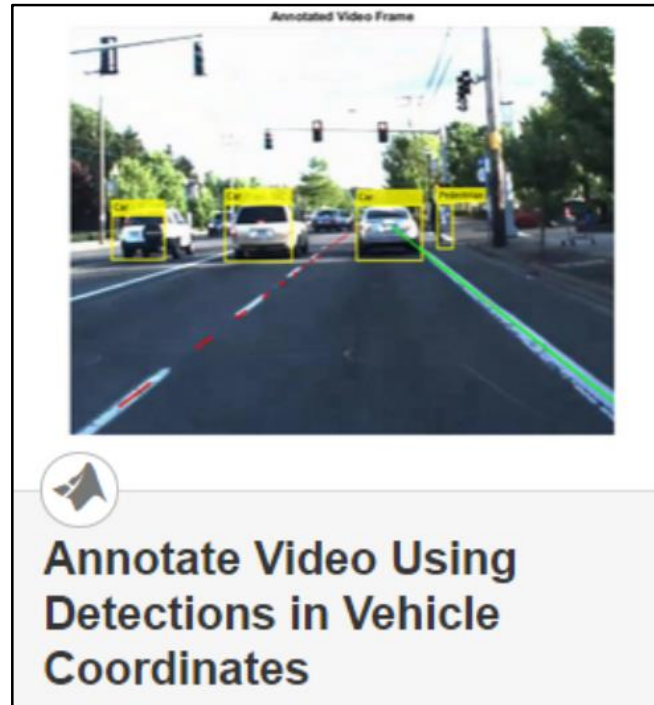


Learn more about visualizing vehicle data

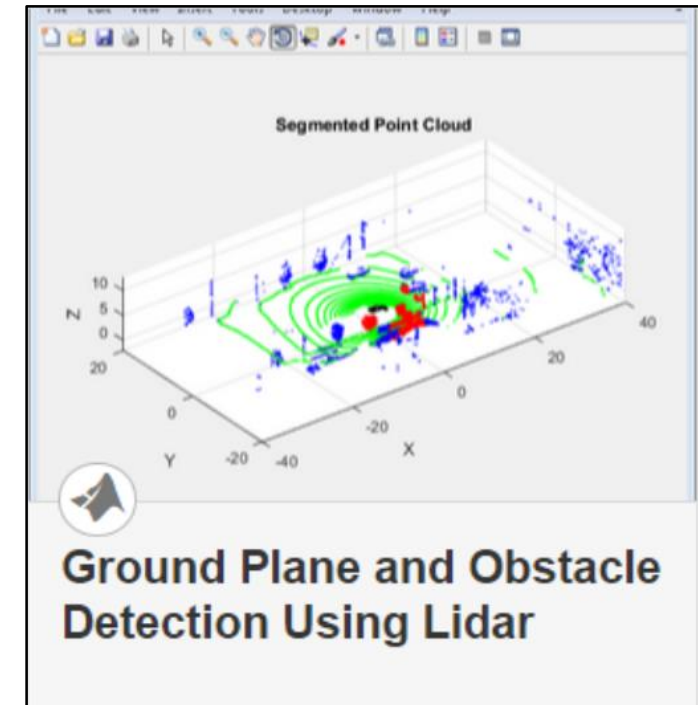
by exploring examples in the Automated Driving System Toolbox



- **Plot object detectors in vehicle coordinates**
 - Vision & radar detector
 - Lane detectors
 - Detector coverage areas



- **Transform between vehicle and image coordinates**

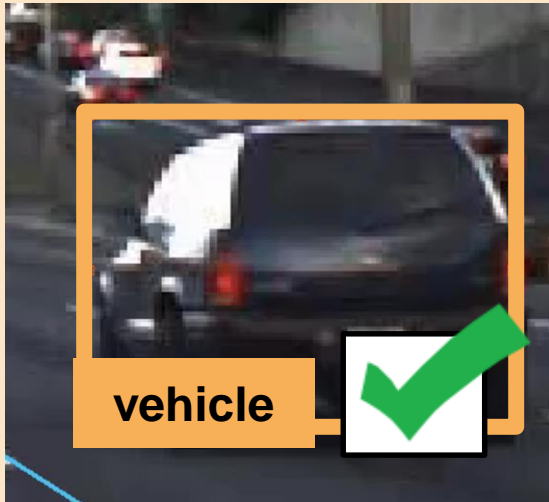


- **Plot lidar point cloud**

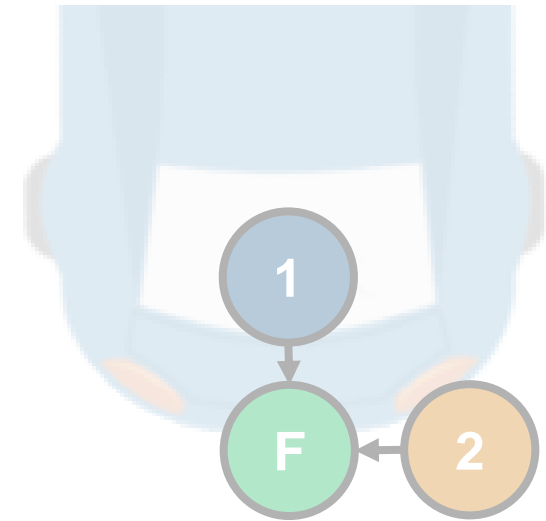
Some common questions from automated driving engineers



How can I
visualize vehicle
data?

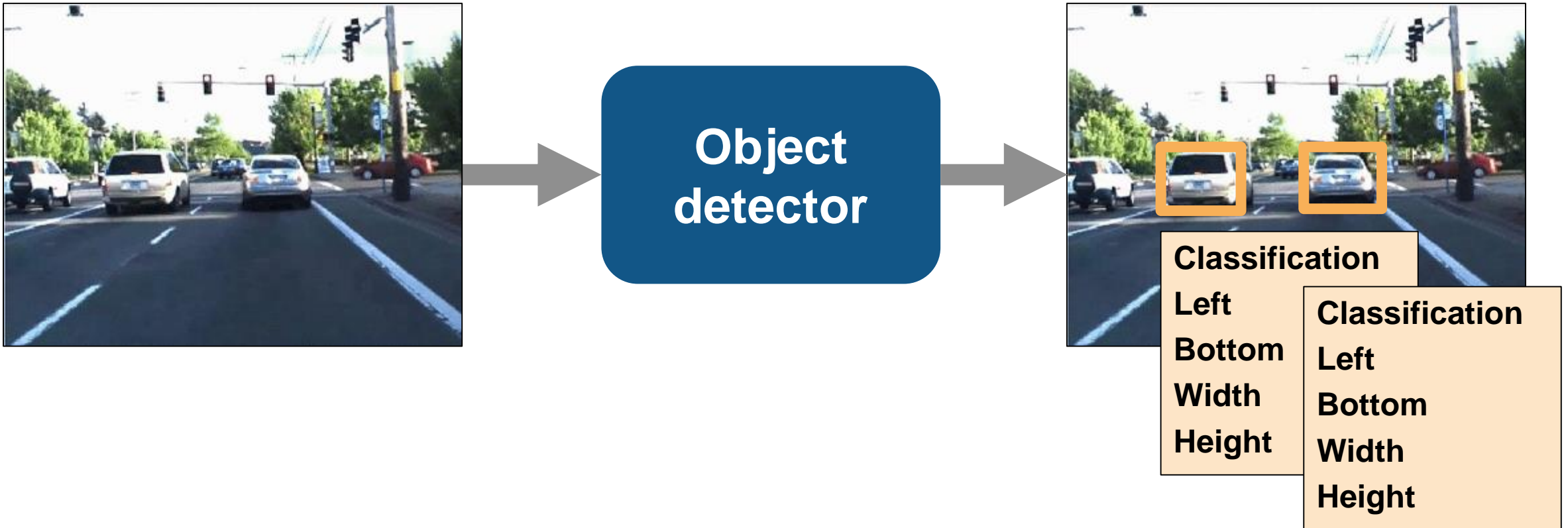


How can I
detect objects in
images?

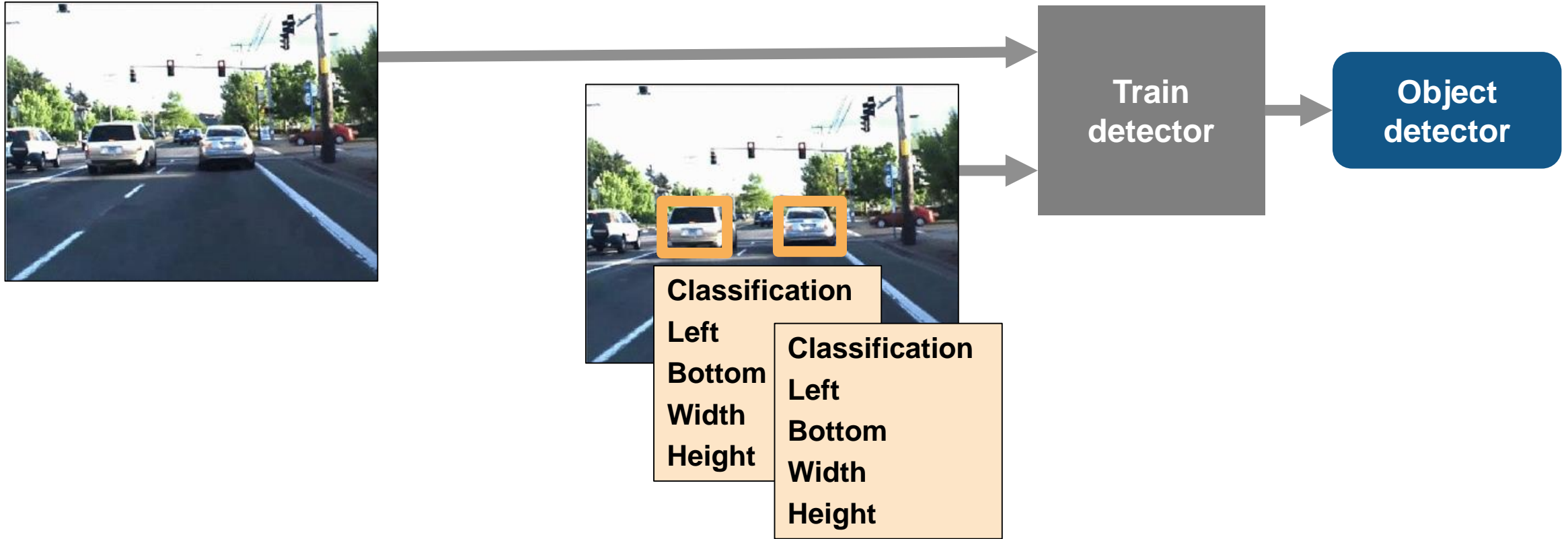


How can I
fuse multiple
detections?

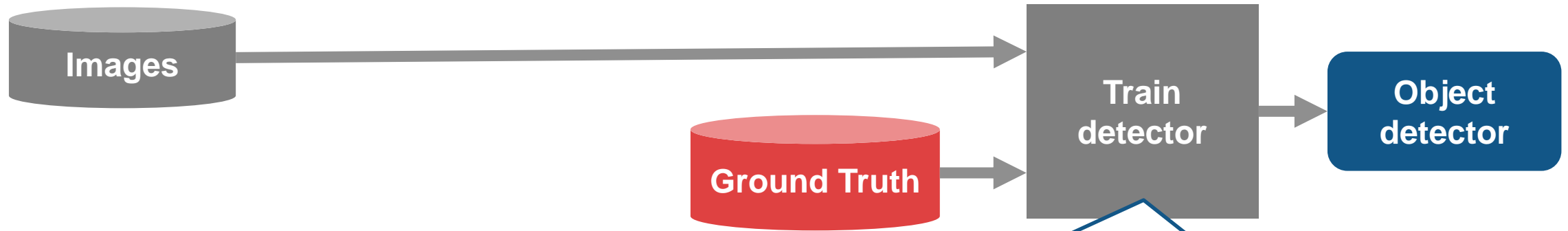
How can I detect objects in images?



Train object detectors based on ground truth



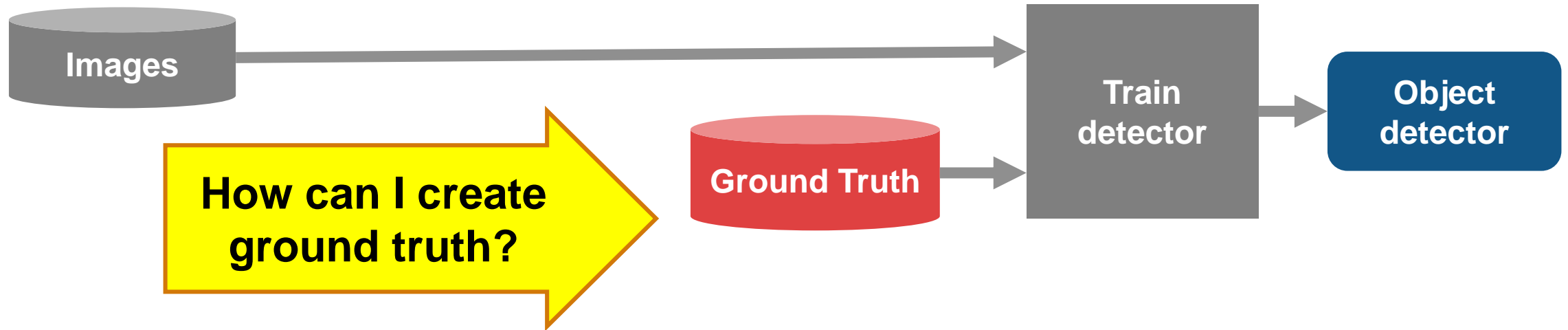
Train object detectors based on ground truth



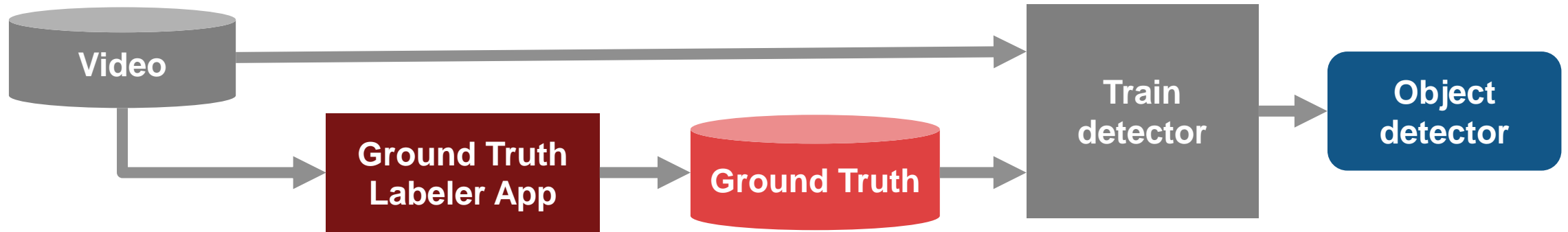
Design object detectors with the Computer Vision System Toolbox

Machine Learning	Aggregate Channel Feature	<code>trainACFObjectDetector</code>
	Cascade	<code>trainCascadeObjectDetector</code>
Deep Learning	R-CNN (Regions with Convolutional Neural Networks)	<code>trainRCNNObjectDetector</code>
	Fast R-CNN	<code>trainFastRCNNObjectDetector</code>
	Faster R-CNN	<code>trainFasterRCNNObjectDetector</code>

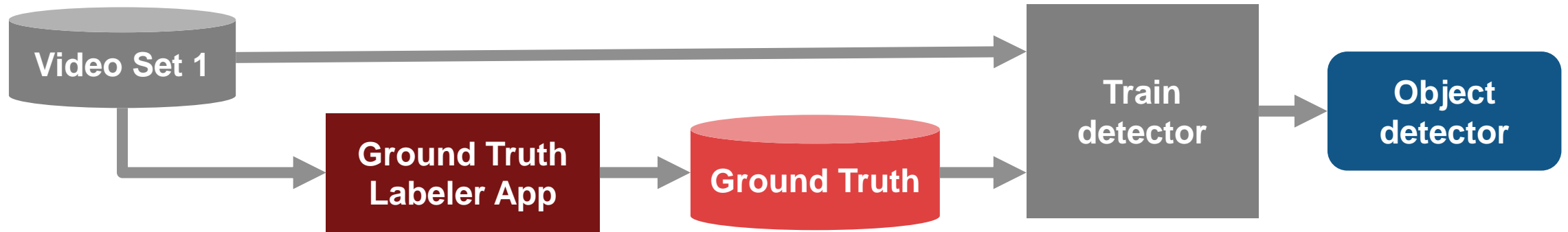
Specify ground truth to train detector



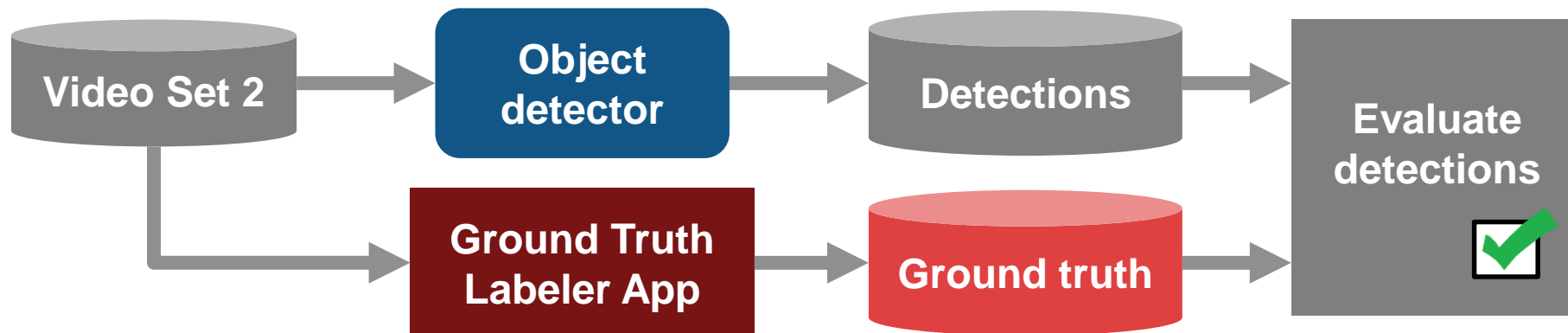
Specify ground truth to train detector



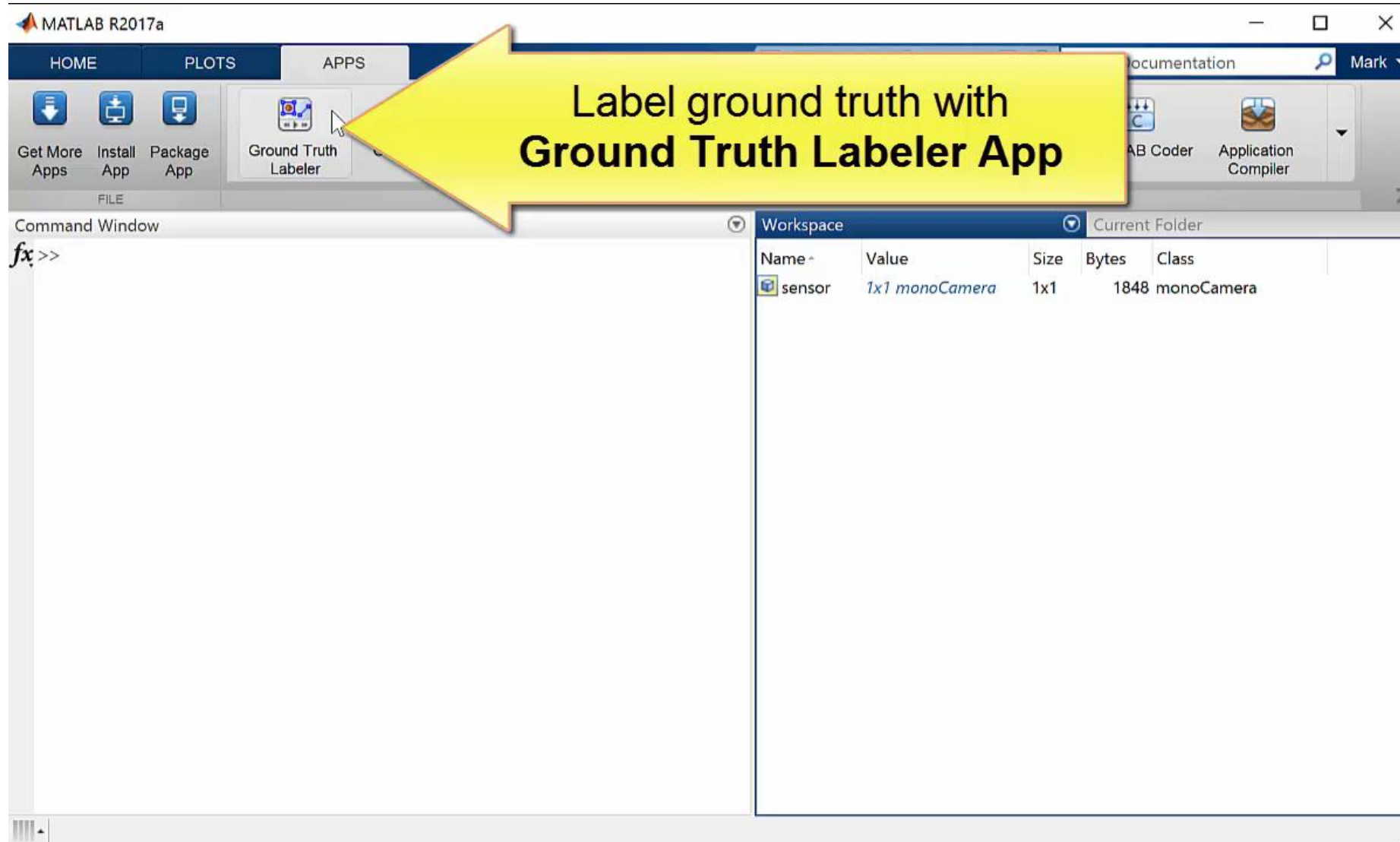
Specify ground truth to train detectors



Specify ground truth to evaluate detectors



Manually label ground truth objects with Ground Truth Labeling App



The screenshot shows the MATLAB R2017a interface. The 'APPS' tab is selected, and the 'Ground Truth Labeler' app icon is highlighted. A yellow arrow points to this icon with the text 'Label ground truth with Ground Truth Labeler App'. The 'Command Window' shows the prompt `fx >>`. The 'Workspace' window displays a table with the following data:

Name	Value	Size	Bytes	Class
sensor	1x1 monoCamera	1x1	1848	monoCamera

Automate labeling between manually labeled frames with temporal interpolator

The screenshot displays the 'Ground Truth Labeler' software interface. The main window is titled 'LABEL' and contains a toolbar with icons for Load, Save, Import Labels, ROI, Zoom In, Zoom Out, Pan, Default Layout, Show ROI Labels, and Show Scene Labels. Below the toolbar are three tabs: 'ROI Label Definition', 'Scene Label Definition', and '05_high'. The 'ROI Label Definition' tab is active, showing a list of labels: 'Car' (highlighted in yellow), 'SunnyDay', and 'LaneChange'. The 'Scene Label Definition' tab is also visible, showing options for 'Current Frame' and 'Time Interval'. The 'Algorithm' dropdown menu is open, showing a list of algorithms: 'ACF Vehicle Detector', 'Point Tracker', and 'Temporal Interpolator'. The 'Temporal Interpolator' algorithm is selected, and its description is displayed: 'Estimate ROIs in intermediate frames using interpolation of rectangle ROIs in key frames.' A yellow callout box with a black border points to the 'Temporal Interpolator' algorithm, containing the text: 'Interpolate regions between frames with Temporal Interpolator'. Below the algorithm list is a video player showing a car on a road, with a red bounding box labeled 'Car' around it. At the bottom of the interface is a timeline with a playhead and several time values: 00.00000 (Start Time), 00.05000 (Current), 16.48607 (End Time), and 25.00000 (Max Time). There are also navigation buttons and a 'Zoom In Time Interv' button.

Automate labeling based on a manually labeled frame with point tracker

The screenshot displays the Ground Truth Labeler software interface. The main window shows a video frame with a car labeled "Car". A yellow arrow points to the "Point Tracker" algorithm in the "Algorithm:" dropdown menu, which is highlighted. The interface includes a toolbar with options like "Load", "Save", "Import Labels", "ROI", "Zoom In", "Zoom Out", and "Pan". The "Algorithm:" dropdown menu lists several options: "Temporal Interpolator", "ACF Vehicle Detector", "Point Tracker", and "Temporal Interpolator". The "Point Tracker" option is selected, and its description is visible: "Track one or more rectangle ROIs over short intervals using Kanade-Lucas-Tomasi (KLT) algorithm." The interface also shows a "Scene Label Definition" section with a "SunnyDay" label and a "LaneChange" label. A timeline at the bottom indicates the current frame is at 06.75310, with a maximum time of 25.00000.

Track region with Point Tracker

Start Time	Current	End Time	Max Time
06.75310	06.75310	14.54546	25.00000

Automate initial ground truth of vehicles with ACF ground truth detector

The screenshot displays the Ground Truth Labeler software interface. The main window shows a video frame of a highway with a car labeled "Car". The interface includes a menu bar with "FILE", "MODE", and "VIEW". The "MODE" menu is open, showing options like "Zoom In", "Zoom Out", and "Pan". The "VIEW" menu is also open, showing options like "Default Layout", "Show ROI Labels", and "Show Scene Labels".

The "ROI Label Definition" section on the left shows a list of labels: "Car" (selected), "SunnyDay", and "LaneChange". The "Scene Label Definition" section shows a list of scene labels: "SunnyDay" and "LaneChange".

The "Algorithm:" dropdown menu is open, showing the following options:

- ACF Vehicle Detector**: Detect vehicles using Aggregate Channel Features (ACF).
- Point Tracker**: Track one or more rectangle ROIs over short intervals using Kanade-Lucas-Tomasi (KLT) algorithm.
- Temporal Interpolator**: Estimate ROIs in intermediate frames using interpolation of rectangle ROIs in key frames.

The "Add Algorithm" and "Refresh list" buttons are visible at the bottom of the algorithm list.

A yellow callout box on the right side of the interface contains the text: "Detect initial regions with Vehicle Detector".

The bottom of the interface shows a timeline with a play button and a "Zoom In Time Interval" button. The timeline displays the following values:

Start Time	Current	End Time	Max Time
14.54546	14.54546	21.12013	25.00000

Export labeled regions as MATLAB time table

The screenshot displays the Ground Truth Labeler application window. The title bar reads "Ground Truth Labeler". The interface is divided into several sections:

- Top Bar:** Contains a "LABEL" tab, a "FILE" menu with "Load", "Save", and "Import Labels" options, and buttons for "View Label Summary" and "Export Labels".
- Left Panel:** Includes "ROI Label Definition" with a "Define new ROI label" button and a list containing "Car". Below it is "Scene Label Definition" with a "Define New Scene Label" button, radio buttons for "Current Frame" and "Time Interval", and "Add Label" and "Remove Label" buttons. A list at the bottom shows "SunnyDay" and "LaneChange" with checkboxes.
- Center:** A video player showing a highway scene with three cars labeled "Car". The video title is "05_highway_lanechange_25s.mp4".
- Right Panel:** A "Scene Labels" legend with "SunnyDay" (teal) and "LaneChange" (red).
- Bottom:** A timeline with a playhead and numerical values: "00.00000" (Start Time), "14.55000" (Current), "25.00000" (End Time), and "25.00000" (Max Time). Playback controls and a "Zoom In Time Interval" button are also present.

A large yellow arrow points from the text "Explore labels by viewing label summary" to the "View Label Summary" button in the top bar.

Customize Ground Truth Labeler App

Ground Truth Labeler - gtlCustomizations

LABEL

Load Save Import Labels ROI Zoom In Zoom Out Pan Default Layout Show ROI Labels Show Scene Labels

Algorithm: Select Algorithm Automate View Label Summary Export Labels

FILE MODE VIEW AUTOMATE LABELING SUMMARY EXPORT

ROI Label Definition

01_city_c2s_fcw_10s.mp4

Define new ROI label

- Car
- Pedestrian
- StopLight
- Lane**

Scene Label Definition

Define New Scene Label

Current Frame Add Label
 Time Interval Remove Label

Before you can label a scene, begin by defining a Scene Label.

00.00000 09.00000 10.20000 10.20000
 Start Time Current End Time Max Time

The screenshot displays the Ground Truth Labeler application window. The main workspace shows a video frame from '01_city_c2s_fcw_10s.mp4' with several objects labeled: two cars (orange), a pedestrian (yellow), three stoplights (green), and two lanes (blue). The interface includes a top toolbar with file, mode, view, and automation options. On the left, there are panels for 'ROI Label Definition' and 'Scene Label Definition'. The 'ROI Label Definition' panel lists 'Car', 'Pedestrian', 'StopLight', and 'Lane' with corresponding color swatches and icons. The 'Scene Label Definition' panel has radio buttons for 'Current Frame' and 'Time Interval', along with 'Add Label' and 'Remove Label' buttons. At the bottom, a timeline shows the current frame at 09.00000, with start and end times set to 00.00000 and 10.20000 respectively. Playback controls and a 'Zoom' button are also visible.

Customize Ground Truth Labeler App

The screenshot displays the Ground Truth Labeler application window titled "Ground Truth Labeler - gtlCustomizations". The interface includes a top toolbar with icons for Load, Save, Import Labels, ROI, Zoom In, Zoom Out, Pan, Default Layout, Show ROI Labels, Show Scene Labels, Algorithm selection, Automate, View Label Summary, and Export Labels. Below the toolbar are tabs for DATA SOURCE, LABEL DEFINITIONS, and SESSION. The DATA SOURCE tab is active, and the "Custom Reader" option is highlighted with a red box. A large yellow arrow points from the text "Add custom image reader with `groundTruthDataSource`" to the "Custom Reader" option. The main workspace shows a video frame with ground truth labels for "Car", "Pedestrian", and "Lane". A timeline at the bottom indicates the current frame is at 09.00000, with start and end times at 00.00000 and 10.20000 respectively.

DATA SOURCE

- Video
- Image Sequence
- Custom Reader**

LABEL DEFINITIONS

- Label Definitions

SESSION

- Session

Scene Label Definition

- Define New Scene Label

Labels in Video Frame: Car, Pedestrian, Lane

Timeline: Start Time: 00.00000, Current: 09.00000, End Time: 10.20000, Max Time: 10.20000

Customize Ground Truth Labeler App

The screenshot shows the 'Ground Truth Labeler - gtlCustomizations' window. The interface is divided into several sections:

- Top Bar:** Contains icons for Load, Save, Import Labels, ROI, Zoom In, Zoom Out, and Pan.
- Algorithm List:** A dropdown menu is open, showing a list of algorithms:
 - Point Tracker:** Track one or more rectangle ROIs over short intervals using Kanade-Lucas-Tomasi (KLT) algorithm.
 - Temporal Interpolator:** Estimate ROIs in intermediate frames using interpolation of rectangle ROIs in key frames.
 - ACF Vehicle Detector:** Detect vehicles using Aggregate Channel Features (ACF).
- Customization Buttons:** A red box highlights two buttons: 'Create New Algorithm' and 'Import Algorithm'.
- ROI Label Definition:** A list of predefined labels: Car, Pedestrian, StopLight, and Lane. The 'Lane' label is currently selected.
- Scene Label Definition:** Options for 'Current Frame' and 'Time Interval' with 'Add Label' and 'Remove Label' buttons.
- Video Player:** Shows a video frame with bounding boxes for 'Car' and 'Pedestrian'.
- Timeline:** Displays 'Start Time' (00.00000), 'Current' (09.00000), 'End Time', and 'Max Time'.

Add custom automation algorithm
`driving.automation.AutomationAlgorithm`

Customize Ground Truth Labeler App

The screenshot displays the 'Ground Truth Labeler - gtlCustomizations' application window. The interface includes a top toolbar with icons for Load, Save, Import Labels, ROI, Zoom In, Zoom Out, Pan, Default Layout, Show ROI Labels, Show Scene Labels, Automate, View Label, and Export. Below the toolbar are three tabs: FILE, MODE, and VIEW. The left sidebar contains two main sections: 'ROI Label Definition' and 'Scene Label Definition'. Under 'ROI Label Definition', there is a 'Define new ROI label' button and a list of labels: Car (orange), Pedestrian (yellow), StopLight (green), and Lane (blue). Under 'Scene Label Definition', there is a 'Define New Scene Label' button, radio buttons for 'Current Frame' and 'Time Interval', and 'Add Label' and 'Remove Label' buttons. The main workspace shows a video frame with bounding boxes around a white car, a pedestrian, and a blue car. Labels 'Car', 'Pedestrian', and 'Lane' are placed near their respective bounding boxes. A yellow arrow points from the text 'Add connection to other tools with driving.connector.Connector' to the 'Automate' button in the toolbar. To the right, a separate window titled 'Figure 1: Point Cloud Pla...' shows a 3D point cloud visualization of the scene with blue lines representing lane markings.

Add connection to other tools with **driving.connector.Connector**

Learn more about detecting objects in images

by exploring examples in the Automated Driving System Toolbox

The screenshot shows the Ground Truth Labeler App interface. On the left, there are panels for 'DATA SOURCE' (Video, Image Sequence, Custom Reader), 'LABEL DEFINITIONS' (Label Definitions), and 'SESSION'. The main area is divided into 'ROI Label Definition' and 'Scene Label Definition' sections. The 'ROI Label Definition' section includes a 'Define New ROI Label' button and a list of labels like 'cars' and 'streetLights'. The 'Scene Label Definition' section includes a 'Define New Scene Label' button and a list of scene labels like 'Sunny', 'Overcast', and 'Tunnel'. A video preview window shows a street scene with bounding boxes around objects. Below the interface, a blue bar contains four steps: 'LOAD Video, Image Sequence, or Custom Reader', 'DEFINE ROIs and Scene Label Definitions', 'SET Interval and Controls', and 'LABEL Rectangles & Lines'.

Define Ground Truth Data for Video or Image Sequences

- Label detections with Ground Truth Labeler App

The screenshot shows the Automate Ground Truth Labeling of Lane Boundaries interface. It features a central video frame of a road with lane boundaries highlighted in cyan and yellow. The interface includes a toolbar with options like 'Zoom In', 'Zoom Out', 'Default Layout', 'Settings', 'Run', 'Stop', 'Undo Run', 'Accept', and 'Cancel'. Below the video frame, there are controls for 'ROI Label Definition' and 'Scene Label Definition'. A timeline at the bottom shows 'Start Time', 'Current', 'End Time', and 'Max Time' values. A circular icon with a cursor is positioned below the video frame.

Automate Ground Truth Labeling of Lane Boundaries

- Add automation algorithm for lane tracking

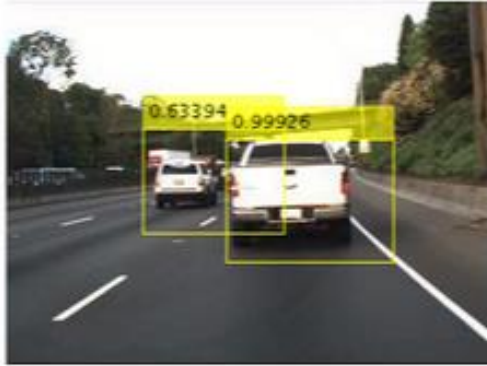
The screenshot shows the driving.connector.Connector class interface. It features a 3D Lidar display on the left and a video frame on the right. The 3D display shows a grid of points representing the Lidar data. The video frame shows a street scene with lane boundaries. The interface includes a toolbar with options like 'Zoom In', 'Zoom Out', 'Default Layout', 'Settings', 'Run', 'Stop', 'Undo Run', 'Accept', and 'Cancel'. Below the video frame, there are controls for 'ROI Label Definition' and 'Scene Label Definition'. A timeline at the bottom shows 'Start Time', 'Current', 'End Time', and 'Max Time' values.

driving.connector.Connector class
Connect Lidar Display to Ground Truth Labeler

- Extend connectivity of Ground Truth Labeler App

Learn more about detecting objects in images

by exploring examples in the Automated Driving System Toolbox



Train a Deep Learning Vehicle Detector

- **Train object detector** using deep learning and machine learning techniques



Track Pedestrians from a Moving Car

- **Explore pre-trained pedestrian detector**



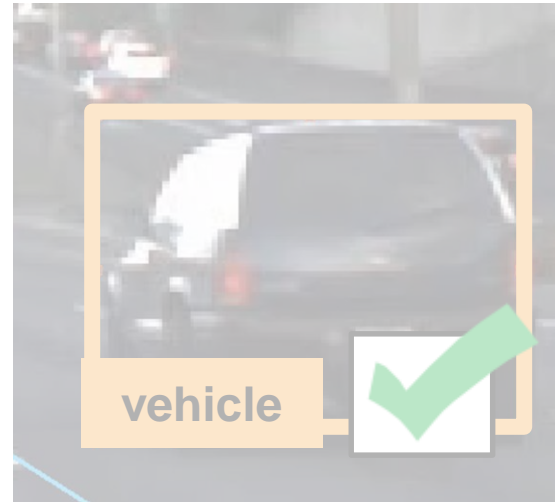
Visual Perception Using Monocular Camera

- **Explore lane detector** using coordinate transforms for mono-camera sensor model

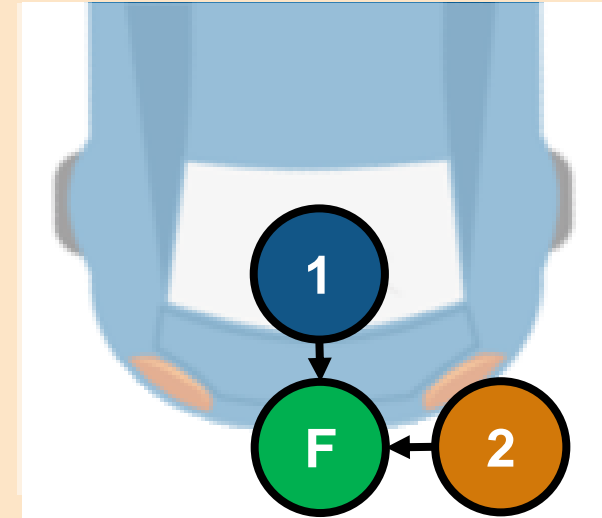
Some common questions from automated driving engineers



How can I
visualize vehicle
data?

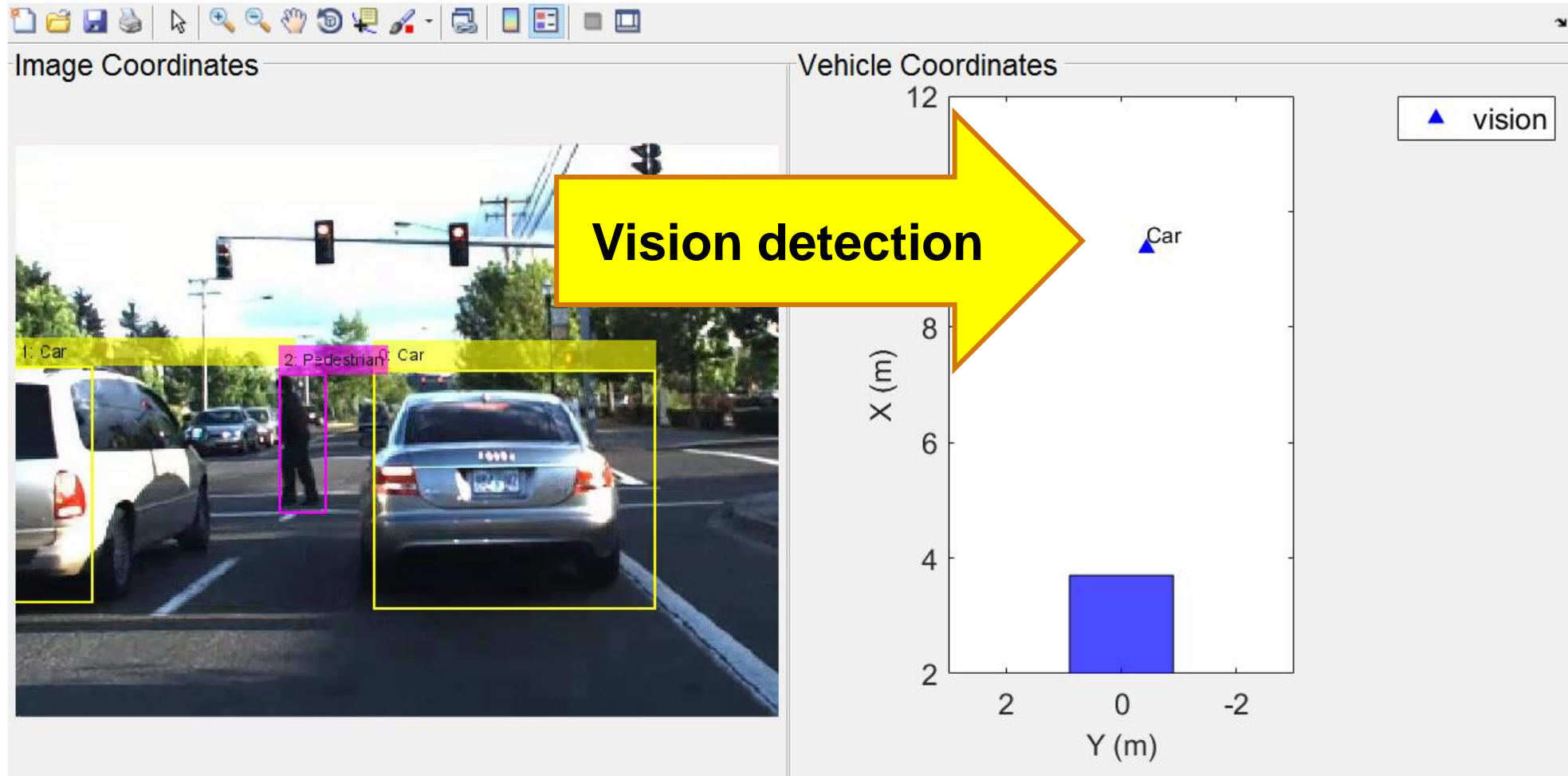


How can I
detect objects in
images?

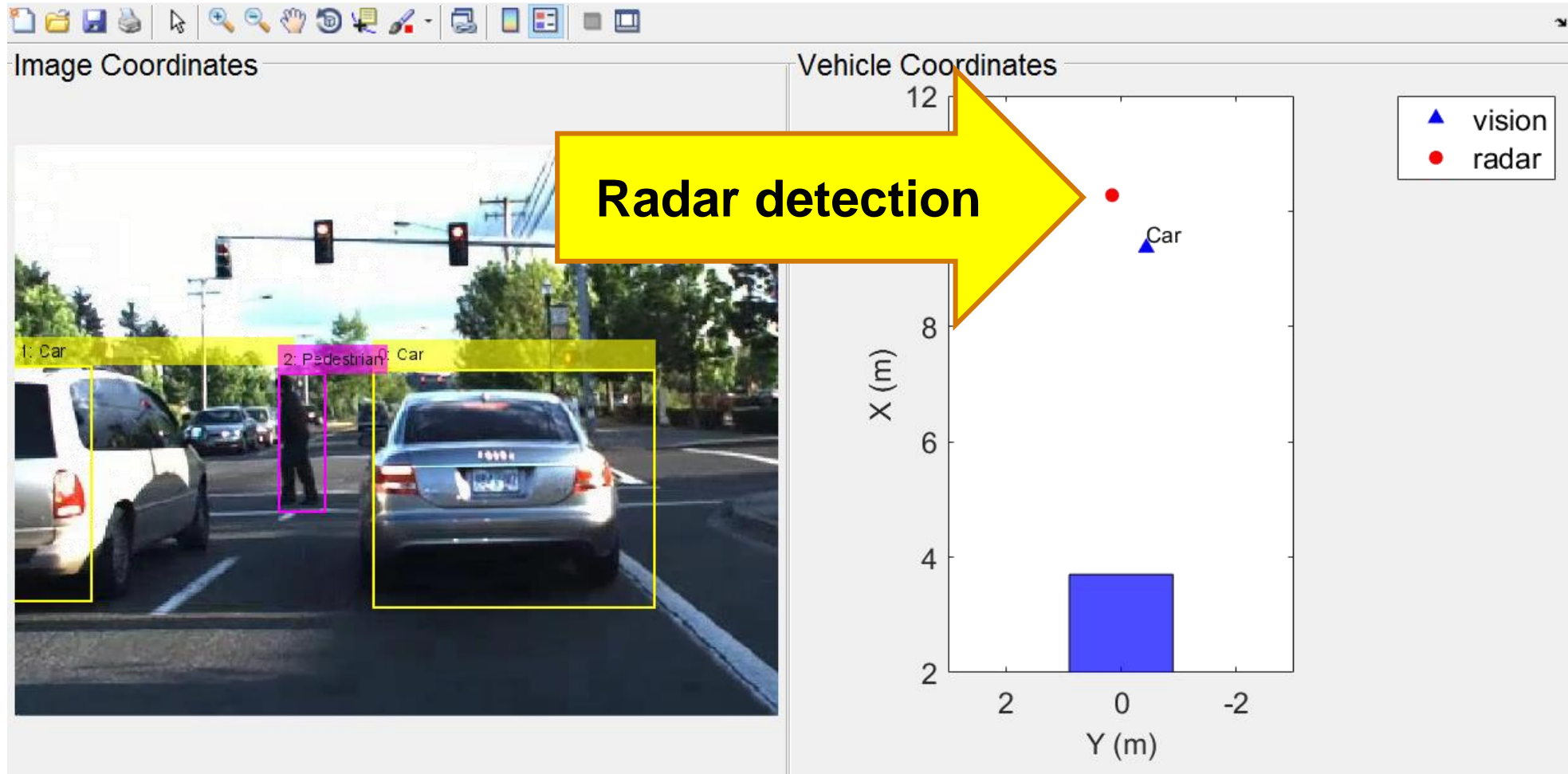


How can I
fuse multiple
detections?

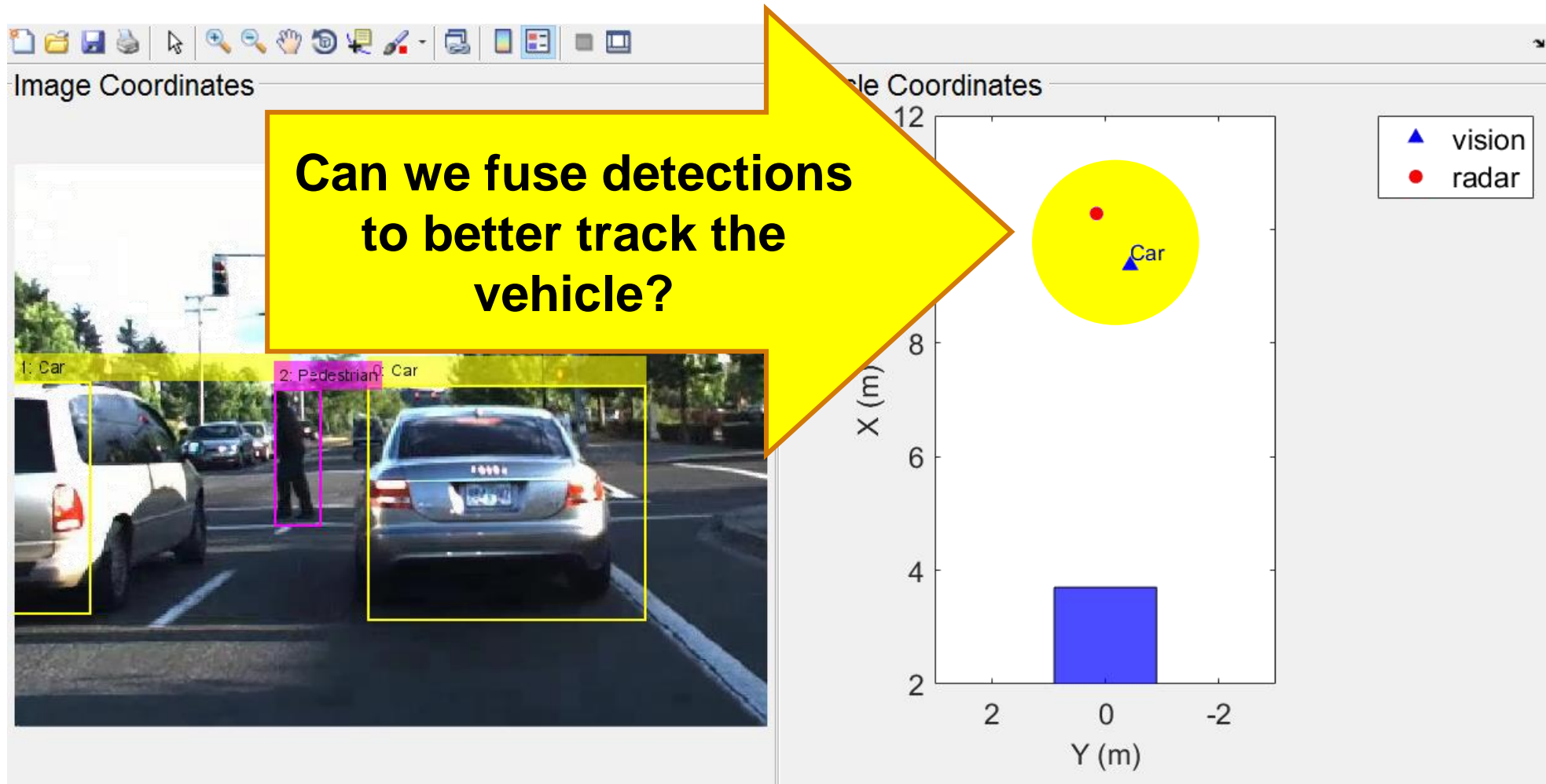
Example of radar and vision detections of a vehicle



Example of radar and vision detections of a vehicle

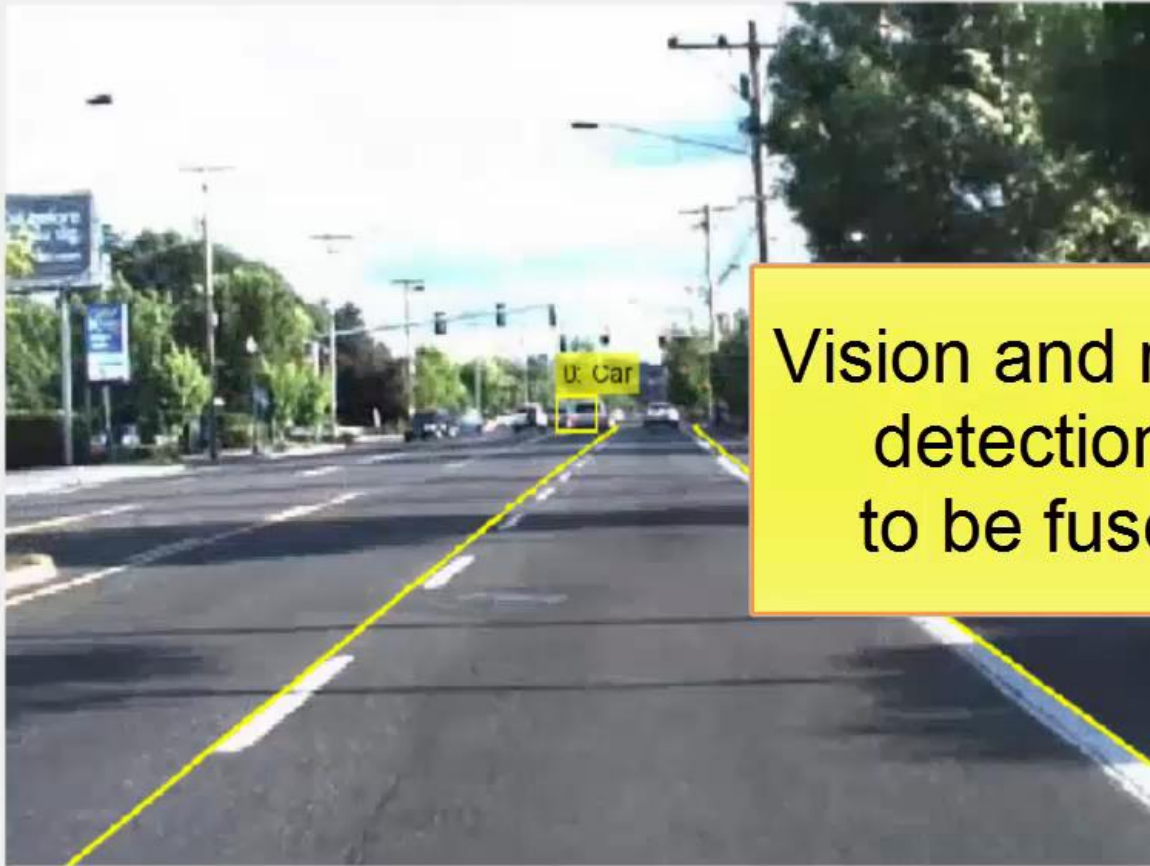


Example of radar and vision detections of a vehicle



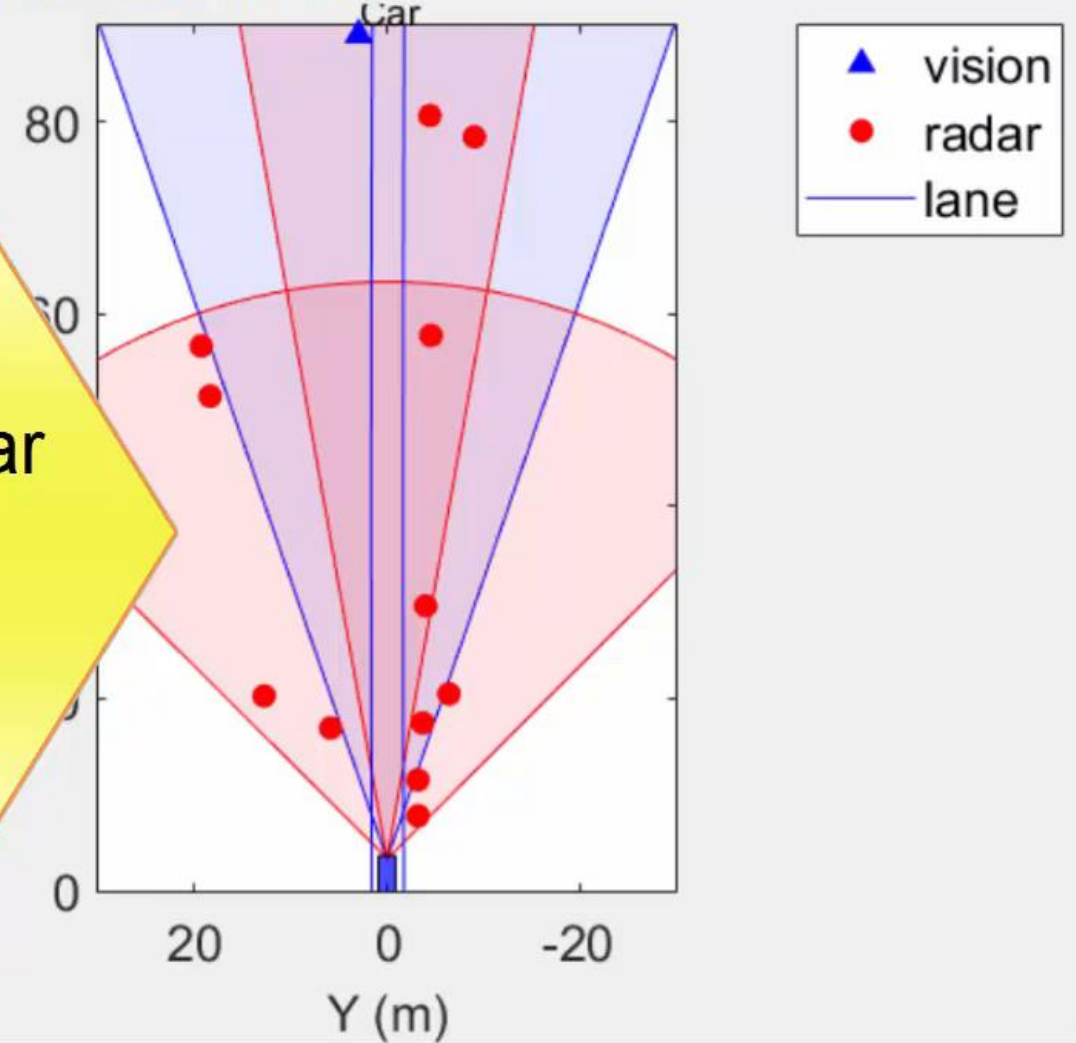
Fuse detections with multi-object tracker

Image Coordinates



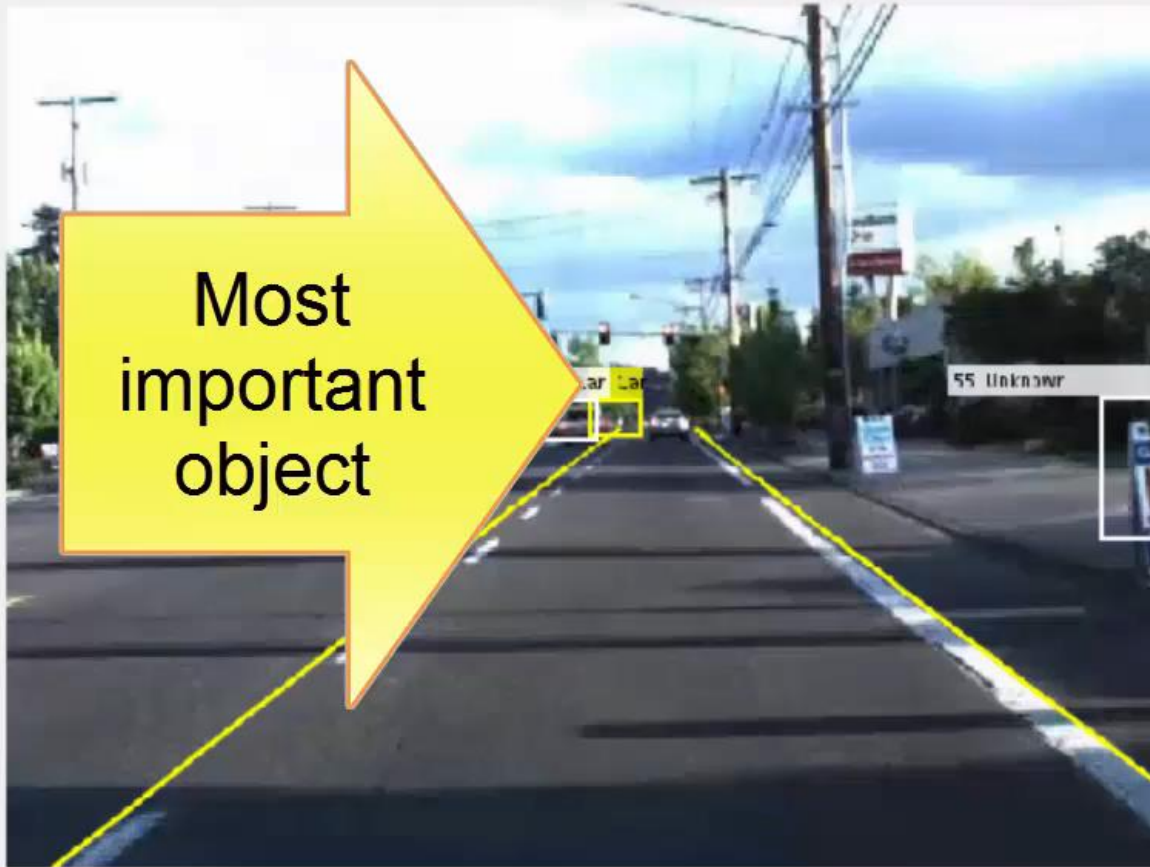
Vision and radar
detections
to be fused

Vehicle Coordinates

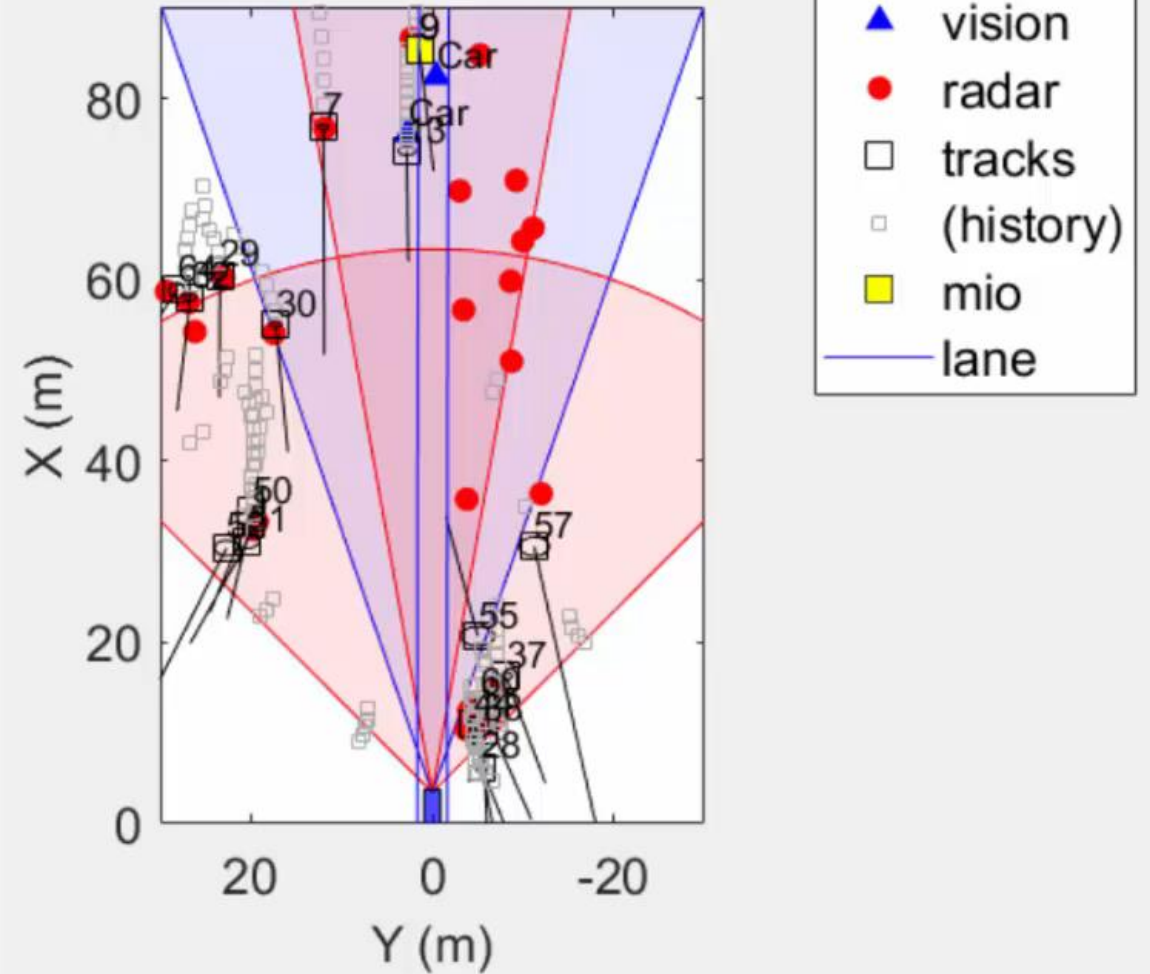


Integrate tracker into higher level algorithm

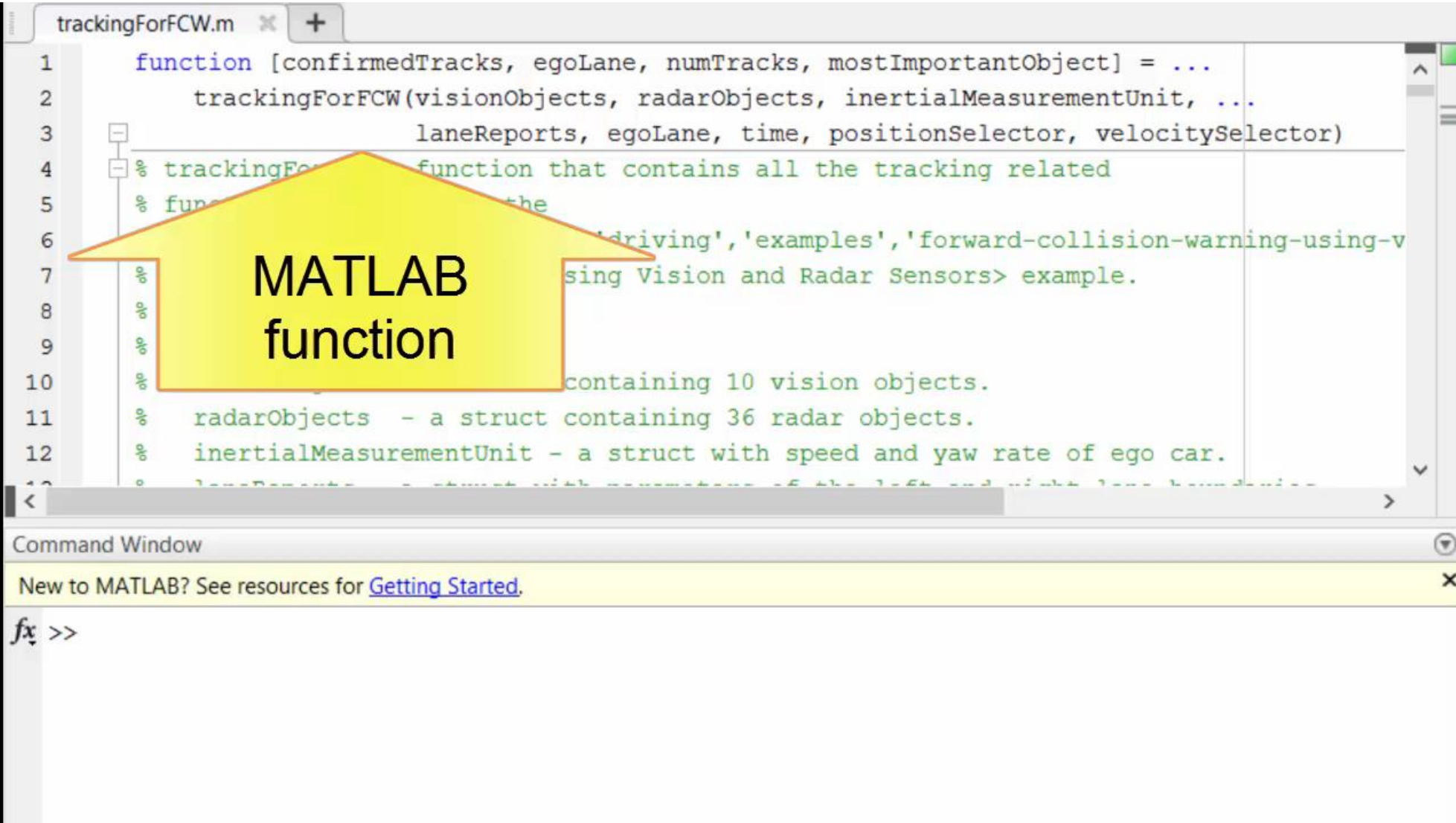
Image Coordinates



Vehicle Coordinates



Generate C code for algorithm



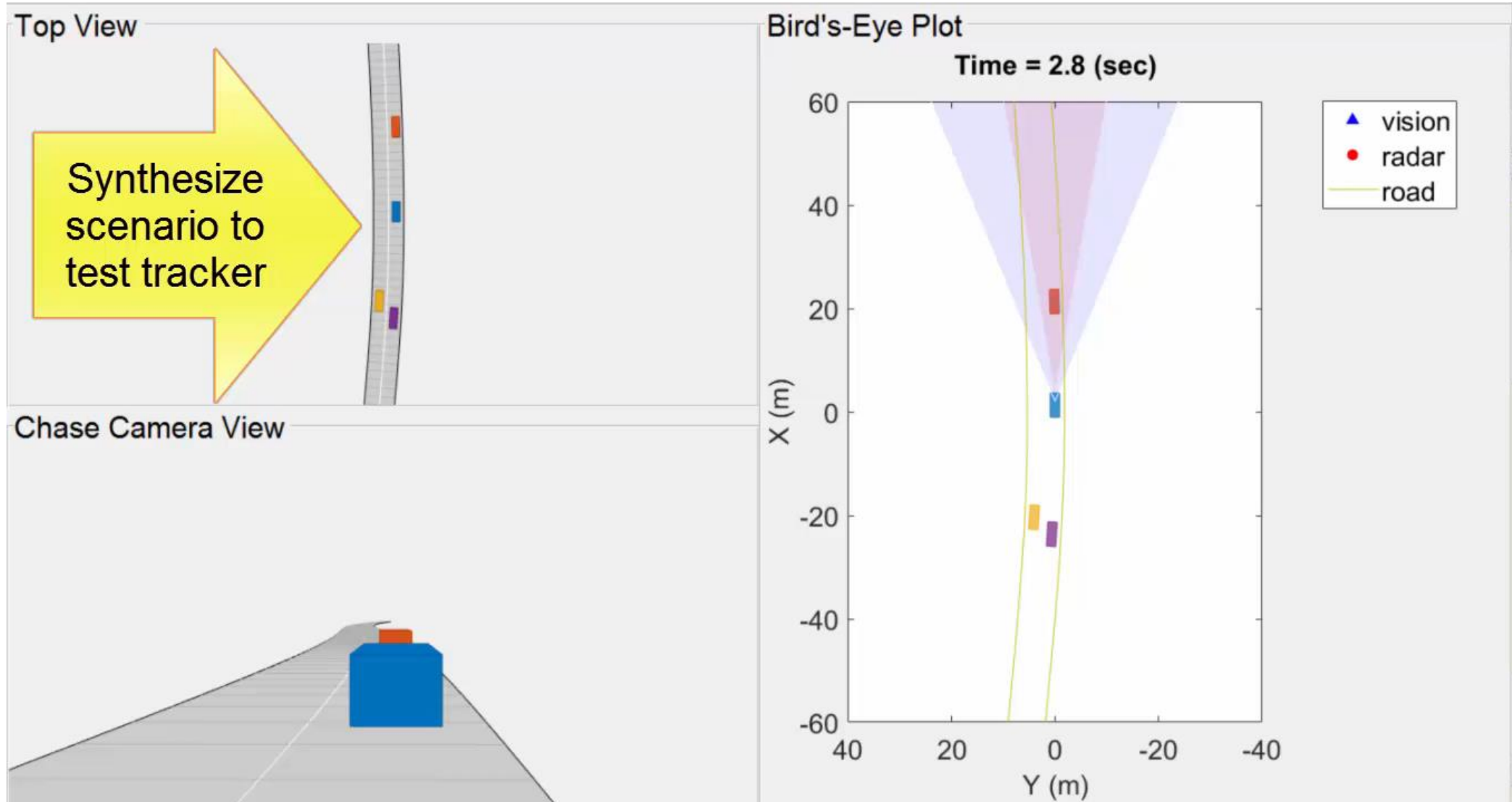
```
1 function [confirmedTracks, egoLane, numTracks, mostImportantObject] = ...
2 trackingForFCW(visionObjects, radarObjects, inertialMeasurementUnit, ...
3 laneReports, egoLane, time, positionSelector, velocitySelector)
4 % trackingForFCW function that contains all the tracking related
5 % functions for the
6 'driving', 'examples', 'forward-collision-warning-using-v
7 sing Vision and Radar Sensors> example.
8 %
9 %
10 % containing 10 vision objects.
11 % radarObjects - a struct containing 36 radar objects.
12 % inertialMeasurementUnit - a struct with speed and yaw rate of ego car.
13 % laneReports - a struct with parameters of the left and right lane boundaries
```

Command Window

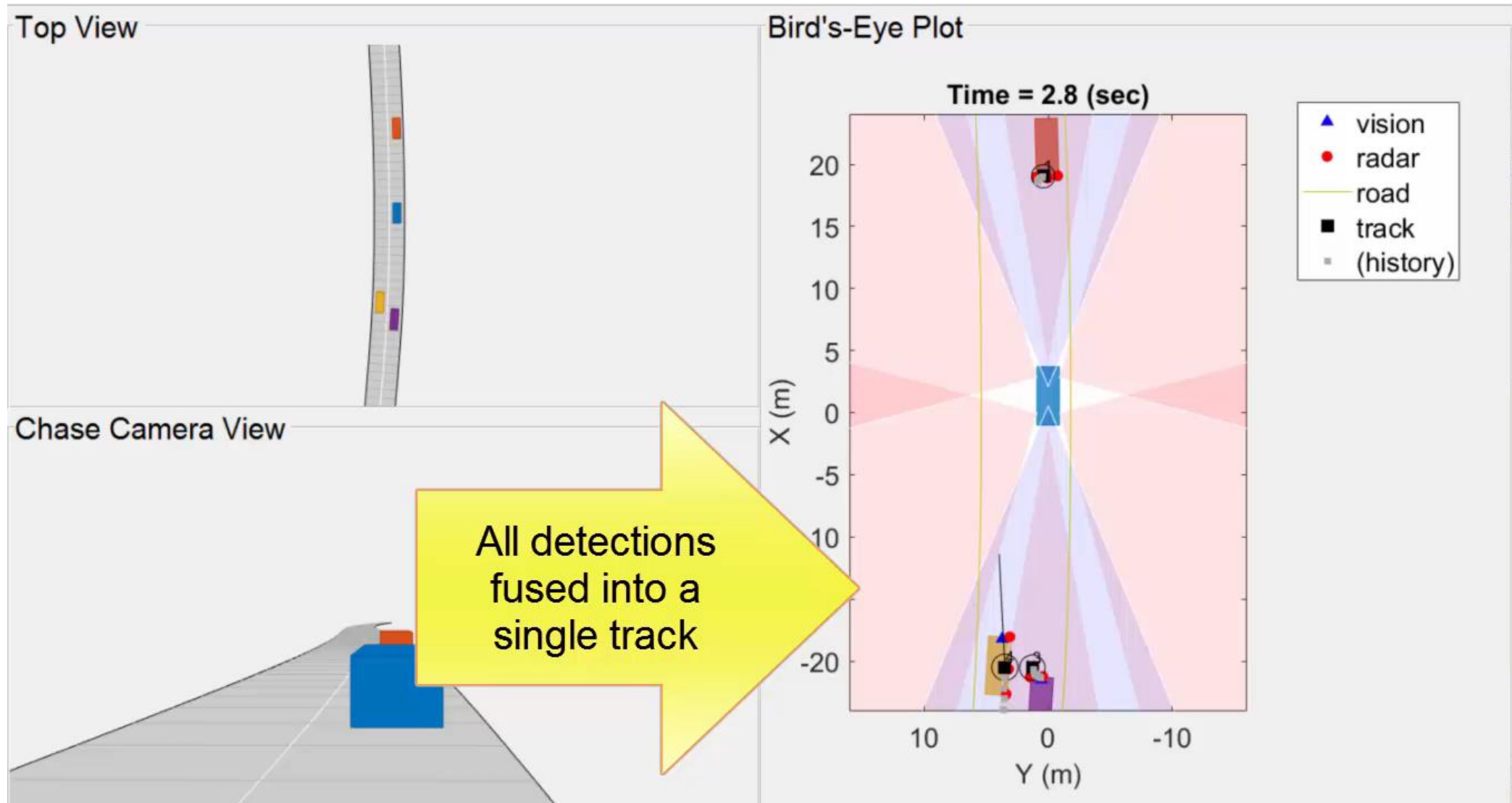
New to MATLAB? See resources for [Getting Started](#).

fx >>

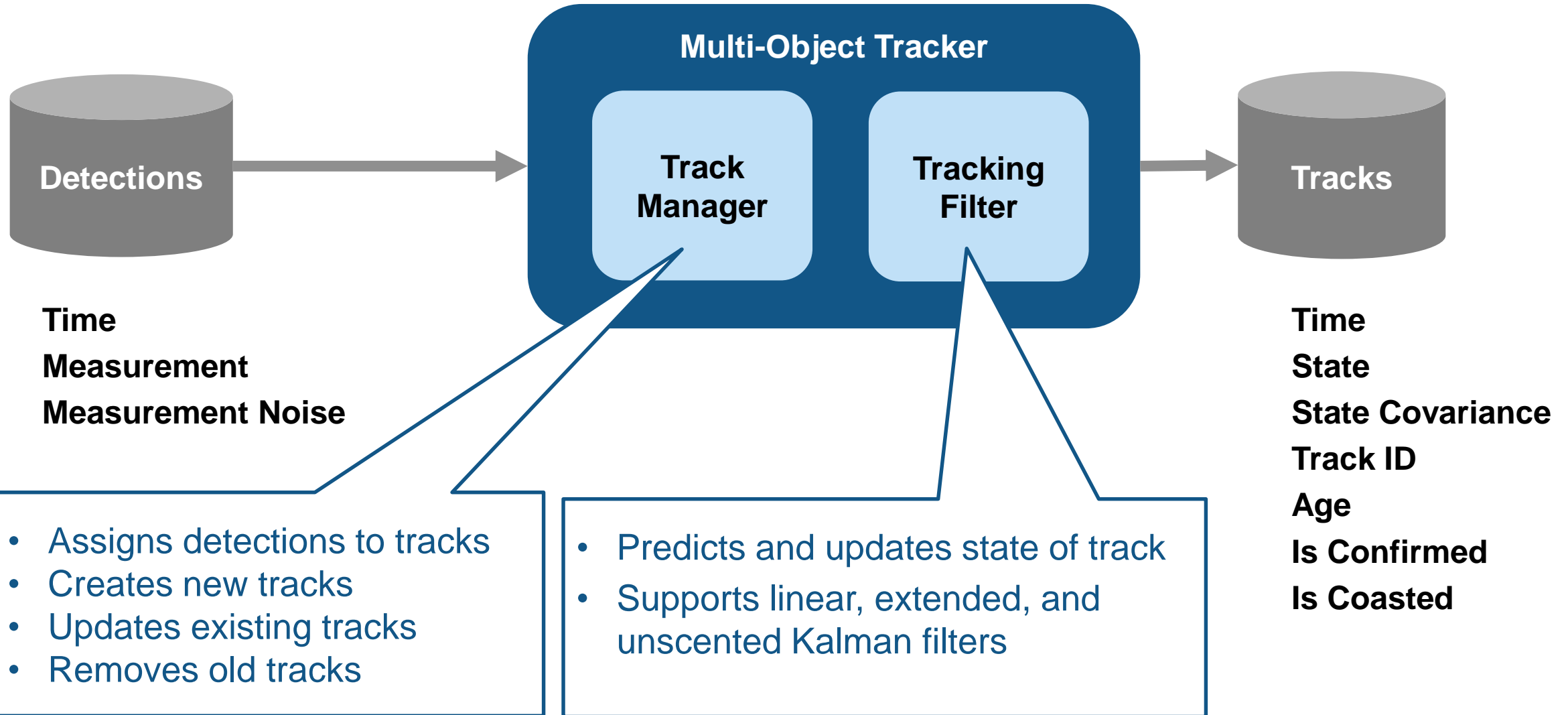
Synthesize scenario to test tracker



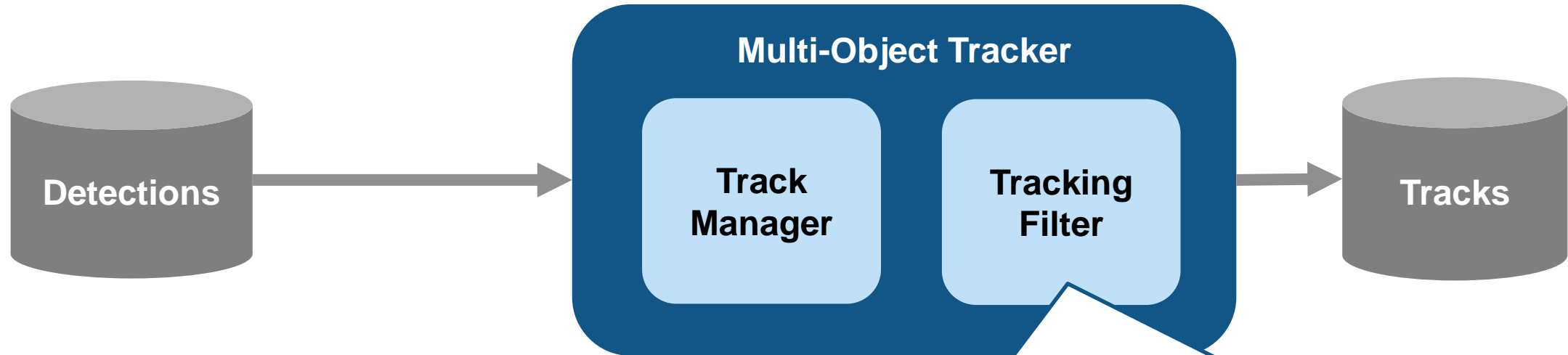
Test tracker against synthesized data



Track multiple object detections

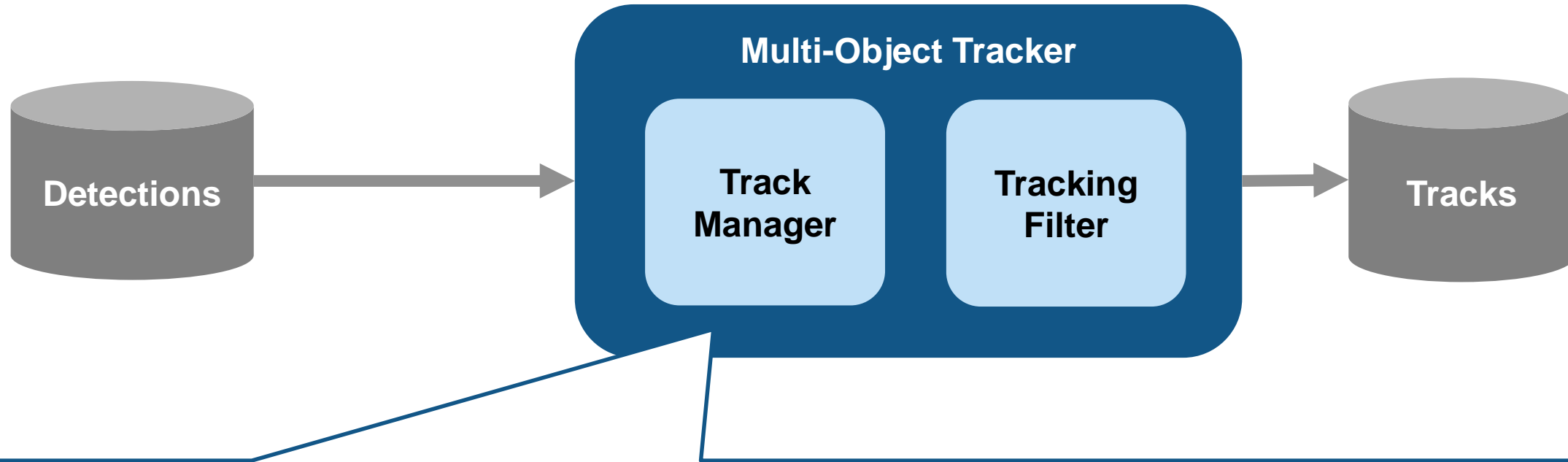


Examples of Kalman Filter (KF) initialization functions



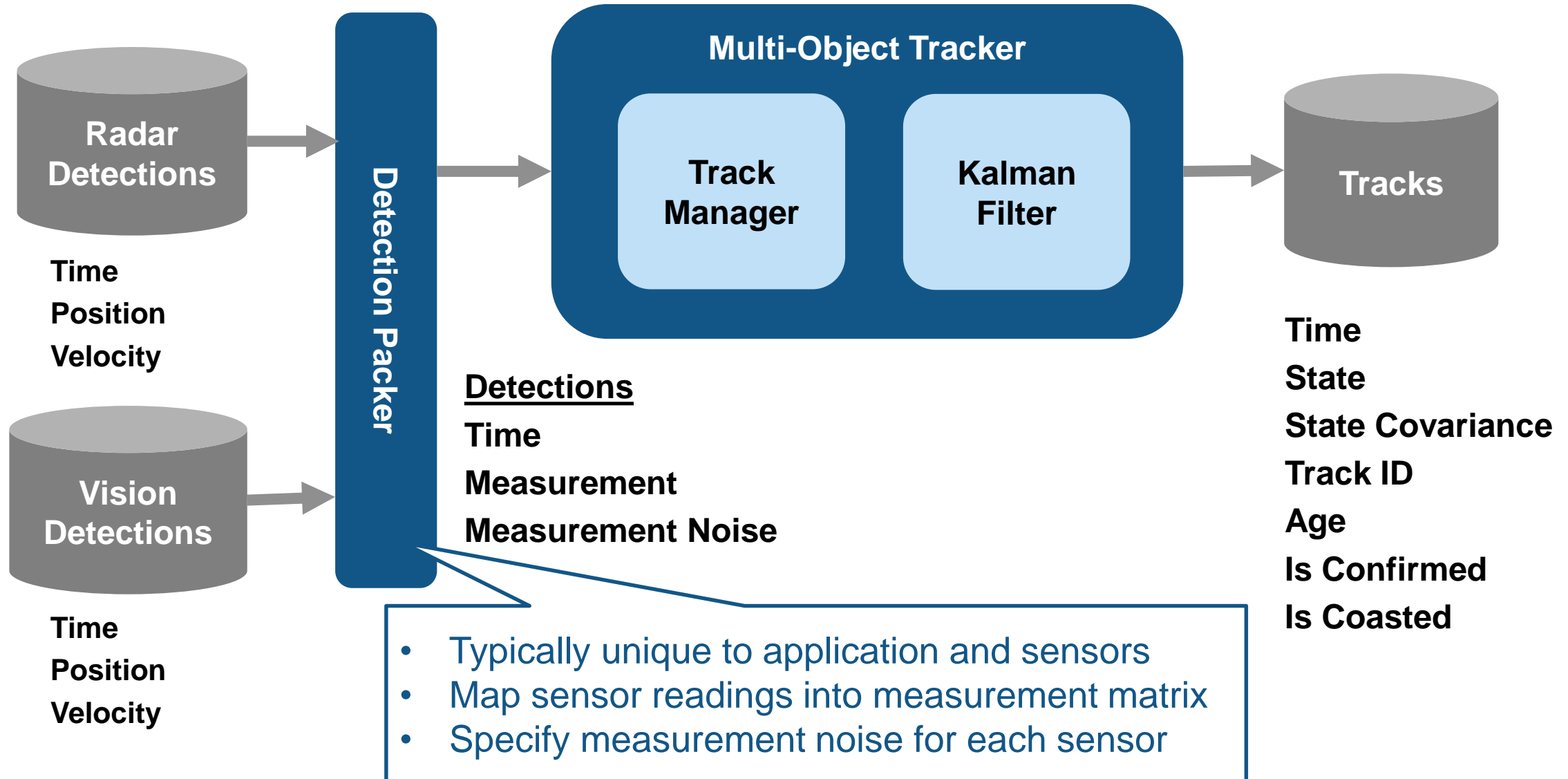
	Linear KF (trackingKF)	Extended KF (trackingEKF)	Unscented KF (trackingUKF)
Constant velocity	<code>initcvkf</code>	<code>initcvekf</code>	<code>initcvukf</code>
Constant acceleration	<code>initcakf</code>	<code>initcaekf</code>	<code>initcaukf</code>
Constant turn	Not applicable	<code>initctekf</code>	<code>initctukf</code>

Example of configuring multi-object tracker

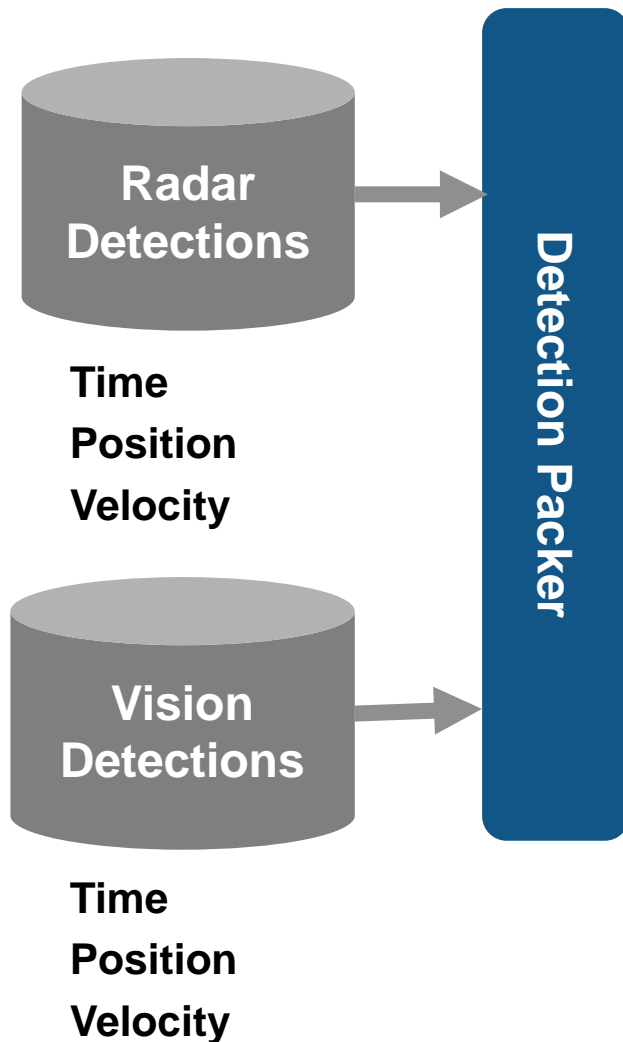


```
tracker = multiObjectTracker(...  
    'FilterInitializationFcn', @initcaekf, ... % Handle to tracking Kalman filter  
    'AssignmentThreshold',    35,... % Normalized distance from track for assignment  
    'ConfirmationParameters',[2 ... % Min number of assignments for confirmation  
                             3],... % Min number of updates for confirmation  
    'NumCoastingUpdates',    5); % Threshold for track deletion
```

Fuse and track multiple detections from different sensors



Example of packing sensor data into detection measurements



```
% Example sensor data
```

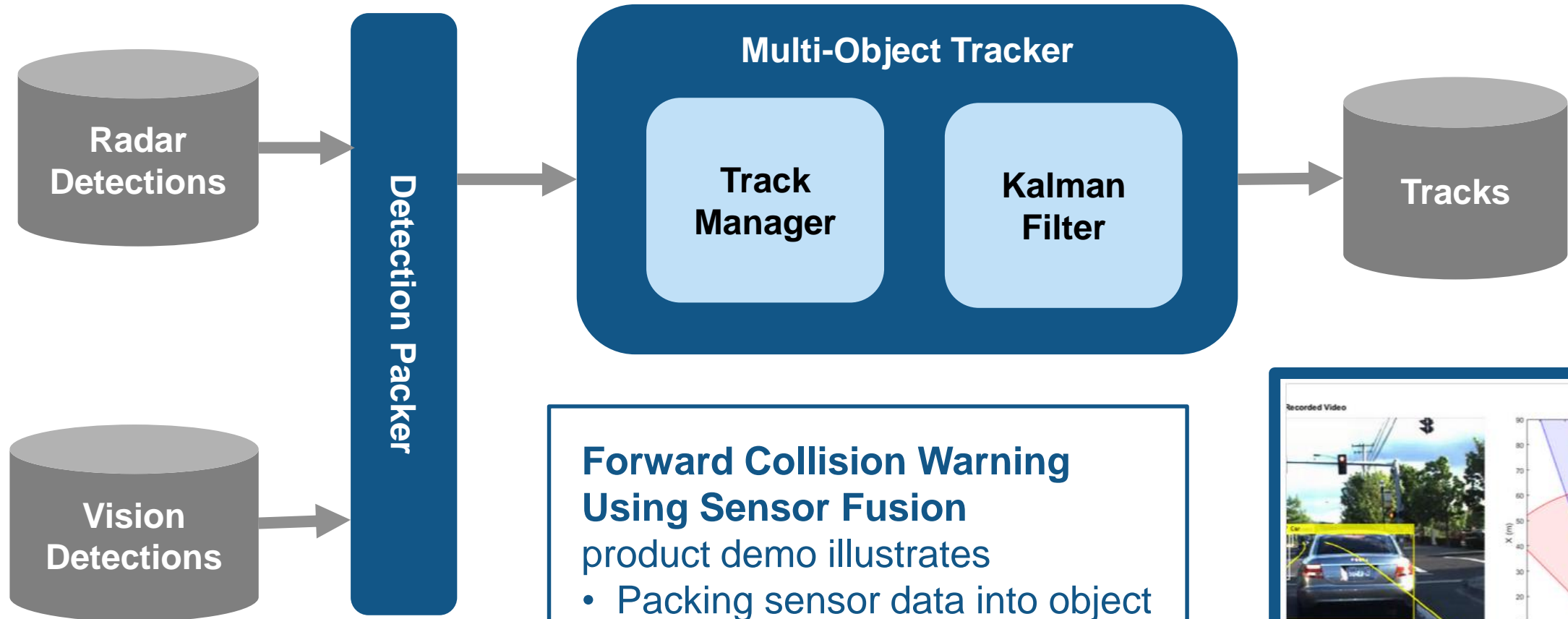
```
radar.Time      = 0.049;      % (sec)
radar.Position  = [10 0.5];  % [x y] (m)
radar.Velocity  = [4.4 0.1]; % [x y] (m/sec)
vision.Time     = 0.051;     % (sec)
vision.Position = [11 0.1];  % [x y] (m)
vision.Velocity = 4.1;       % [x]   (m/sec)
```

```
% Pack to constant acceleration measurement format:
% [positionX; velocityX; positionY; velocityY]
```

```
packedDetections(1) = objectDetection(radar.Time,...
    [radar.Position(1); radar.Velocity(1);...
    radar.Position(2); radar.Velocity(2)],...
    'MeasurementNoise', diag([1,1,2,10]));
```

```
packedDetections(2) = objectDetection(vision.Time,...
    [vision.Position(1); vision.Velocity(1);...
    vision.Position(2); 0],...
    'MeasurementNoise', diag([2,1,1,10]));
```

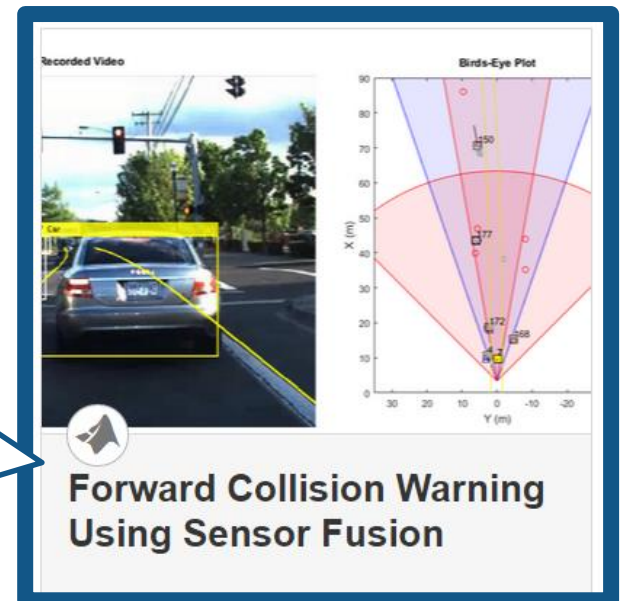
Explore demo to learn more about fusing detections



Forward Collision Warning Using Sensor Fusion

product demo illustrates

- Packing sensor data into object detections
- Initializing Kalman filter
- Configuring multi-object tracker



Generate C code for algorithm with MATLAB Coder

```
trackingForFCW_kernel.m × +
1 function [confirmedTracks, egoLane, numTracks, mostImportantObject] = ...
2 trackingForFCW_kernel(visionObjects, radarObjects, inertialMeasurementUnit, ...
3 laneReports, egoLane, time, positionSelector, velocitySelector)
```

Generate C code with
codegen

```
File: trackingForFCW_kernel.c
1629 */
1630 void trackingForFCW_kernel(const struct0_T *visionObjects, const struct2_T
1631 *radarObjects, const struct4_T *inertialMeasurementUnit, const struct5_T
1632 *laneReports, struct7_T *egoLane, double time, const double positionSelector
1633 [12], const double velocitySelector[12], mxArray_struct8_T *confirmedTracks,
1634 double *numTracks, struct10_T *mostImportantObject)
```

Generate C-code for algorithm with MATLAB Coder

```
%% Create variables that will be used to specify example input arguments
[visionObjects, radarObjects, imu, lanes] = ...
    helperReadSensorRecordingsFile('01_city_c2s_fcw_10s_sensor.mat');
egoLane = struct('left', [0 0 0], 'right', [0 0 0]);
time = 0;
positionSelector = [1 0 0 0 0 0; 0 0 0 1 0 0];
velocitySelector = [0 1 0 0 0 0; 0 0 0 0 1 0];

exampleInputs = {visionObjects(1), radarObjects(1), imu(1), ...
    lanes(1), egoLane, time, ...
    positionSelector, velocitySelector};

%% Generate code for sensor fusion algorithm: trackingForFCW_kernel
codegen trackingForFCW_kernel -args exampleInputs -config:dll -launchreport
```

Install patch to generate C code from multiObjectTracker

- <https://www.mathworks.com/support/bugreports/1546972>



1546972



Summary

Code generation fails for multiObjectTracker in 'lib' or 'dll' configuration

Description

Code generation of a function that uses multiObjectTracker fails under some conditions with the following error message:

```
??? Property 'pSampleDetection.Measurement' is undefined on  
some execution paths but is used inside the called  
function.
```

Workaround

Installation instructions

Specify driving scenario and roads

```

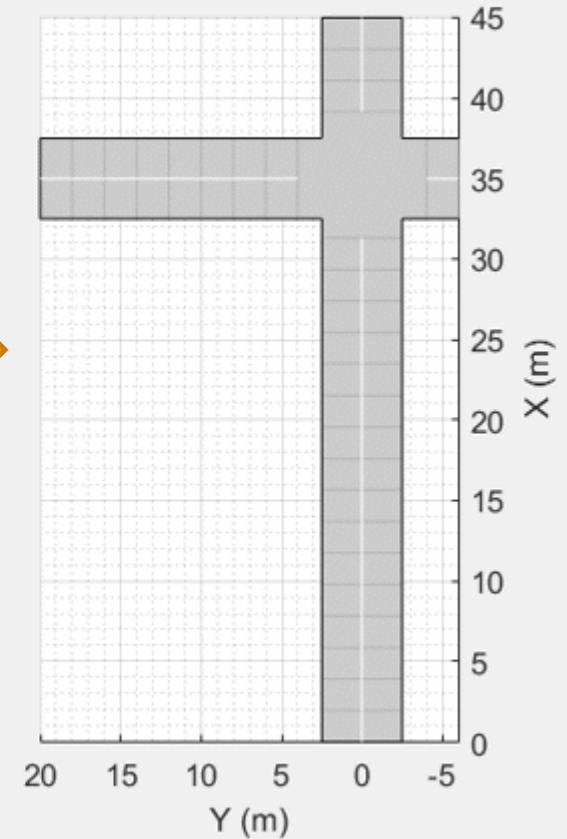
%% Create a new scenario
s = drivingScenario('SampleTime', 0.05);

%% Create road
road(s, [ 0  0; ... % Centers [x,y] (m)
        45  0], ...
        5); % Width (m)
road(s, [35  20; ...
        35 -10], ...
        5);

%% Plot scenario
p1 = uipanel('Position', [0.5 0 0.5 1]);
a1 = axes('Parent', p1);
plot(s, 'Parent', a1, ...
      'Centerline', 'on', 'Waypoints', 'on')
a1.XLim = [0 45];
a1.YLim = [-6 20];

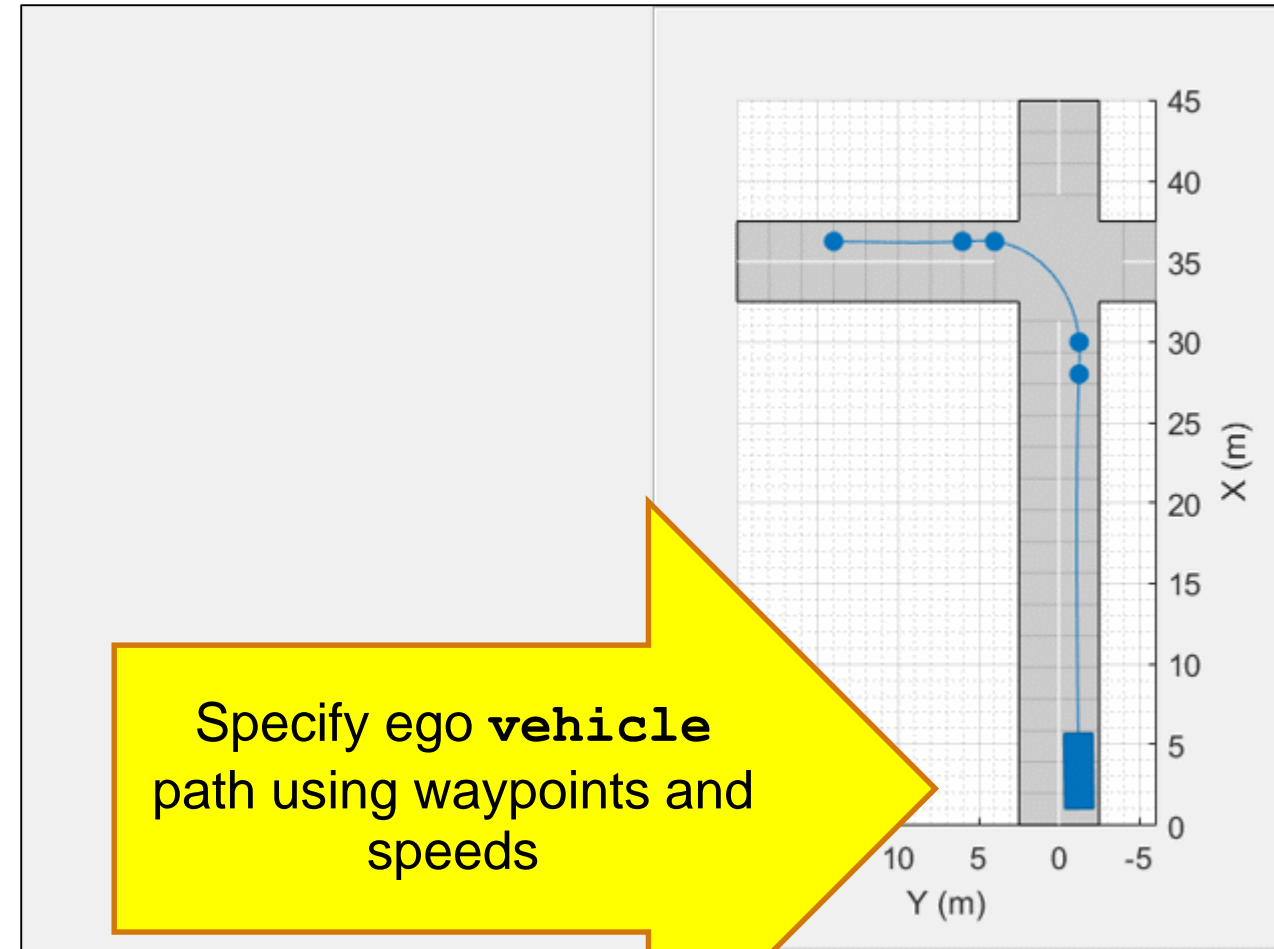
```

Specify road centers and width as part of a **drivingScenario**



Add ego vehicle

```
%% Add ego vehicle
egoCar = vehicle(s);
waypoints = [ 2  -1.25;... % [x y] (m)
             28 -1.25;...
             30  -1.25;...
             36.25 4;...
             36.25 6;...
             36.25 14];
speed = 13.89; % (m/s) = 50 km/hr
path(egoCar, waypoints, speed);
```

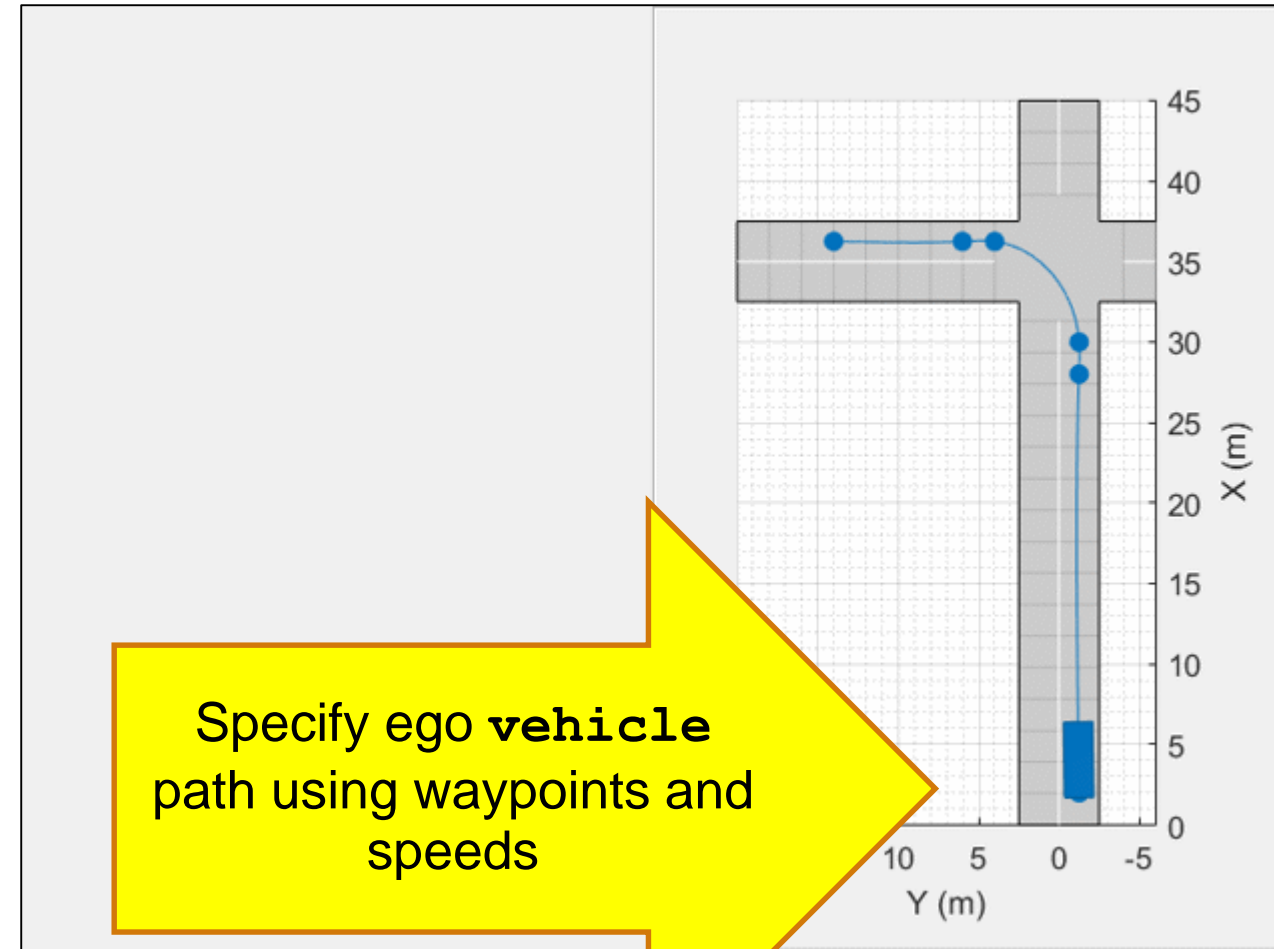


Add ego vehicle

```
%% Add ego vehicle
egoCar = vehicle(s);
waypoints = [ 2  -1.25;... % [x y] (m)
             28 -1.25;...
             30 -1.25;...
             36.25  4;...
             36.25  6;...
             36.25 14];

speed = 13.89; % (m/s) = 50 km/hr
path(egoCar, waypoints, speed);

%% Play scenario
while advance(s)
    pause(s.SampleTime);
end
```



Add target vehicle and pedestrian actor

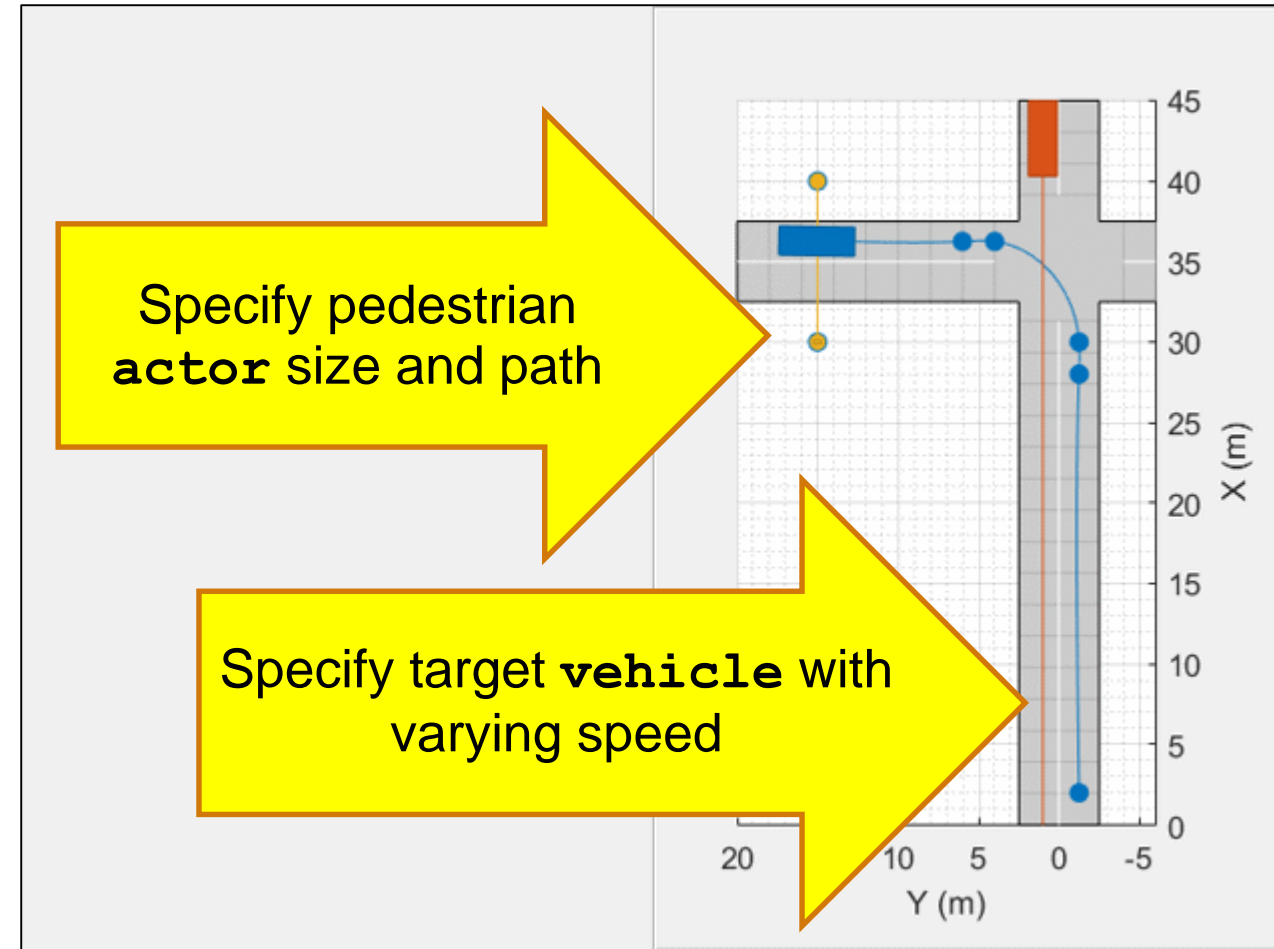
```

%% Add Target vehicle
targetVehicle = vehicle(s);
path(targetVehicle,...
    [44 1; -4 1],... % Waypoints (m)
    [5 ; 14]);      % Speeds (m/s)

%% Add child pedestrian actor
child = actor(s, 'Length',0.24,...
    'Width',0.45,...
    'Height',1.7,...
    'Position',[40 -5 0],...
    'Yaw',180);

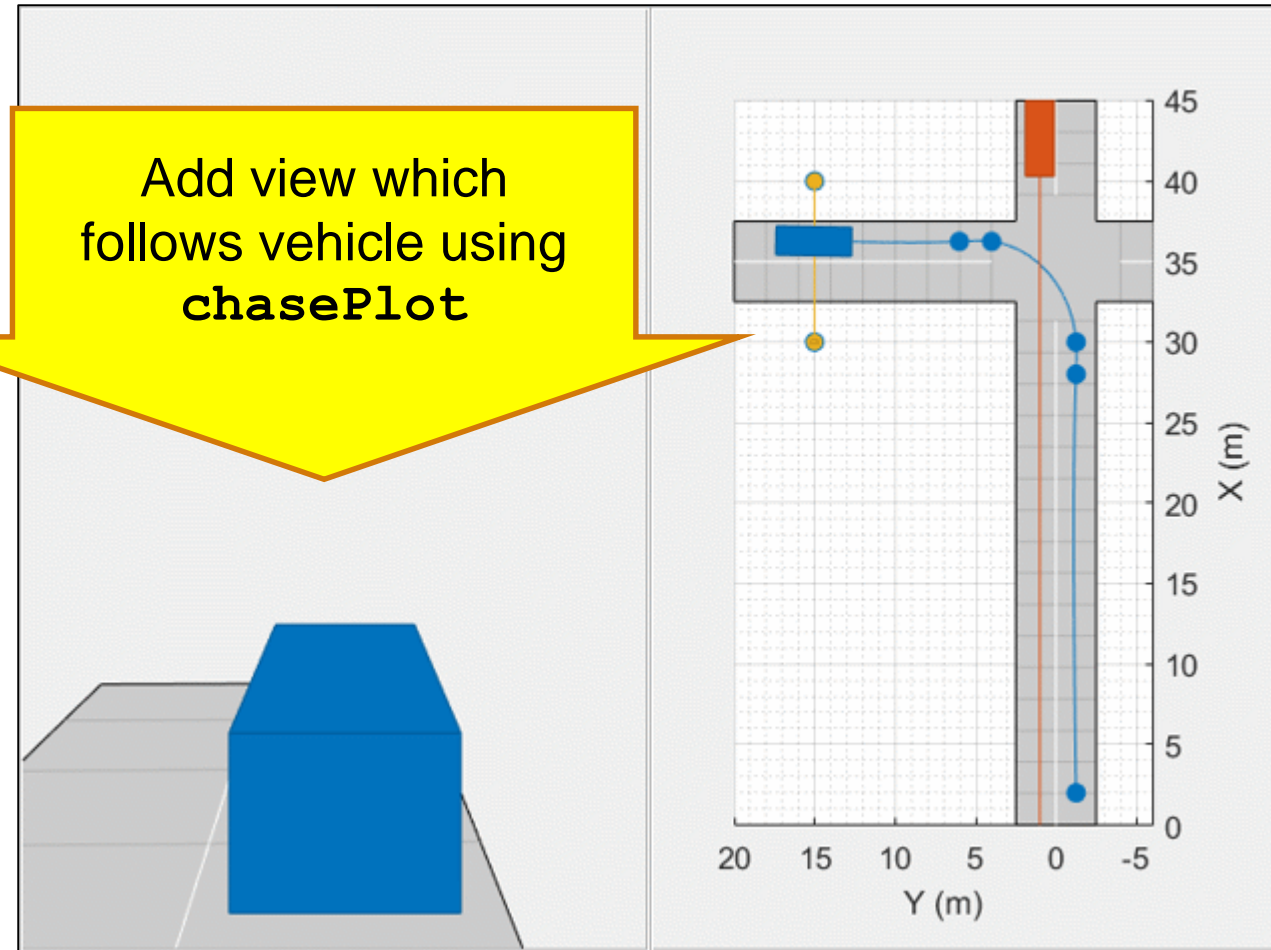
path(child,...
    [30 15; 40 15],... % Waypoints (m)
    1.39); % Speed (m/s) = 5 km/hr

```



View scenario from behind ego vehicle

```
%% Add chase view (left)
p2 = uipanel('Position',[0 0 0.5 1]);
a2 = axes('Parent',p2);
chasePlot(egoCar,...
    'Parent',a2,...
    'Centerline','on',...
    'ViewHeight',3.5,... % (m)
    'ViewLocation',[-8 0]); % [x y] (m)
```



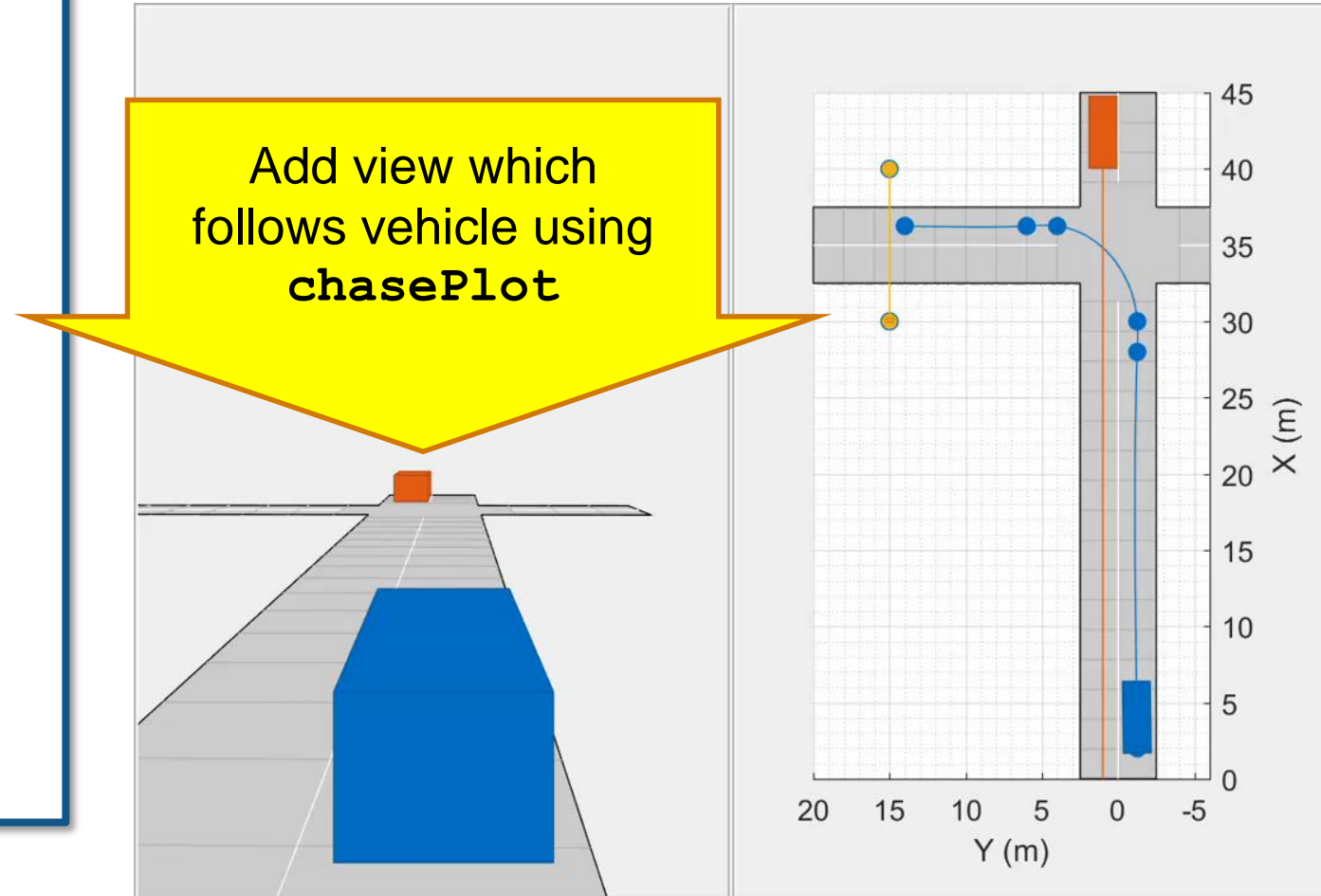
View scenario from behind ego vehicle

```

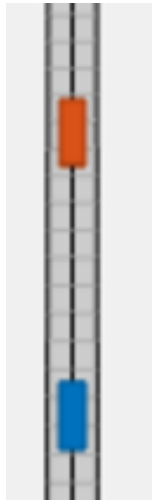
%% Add chase view (left)
p2 = uipanel('Position',[0 0 0.5 1]);
a2 = axes('Parent',p2);
chasePlot(egoCar,...
    'Parent',a2,...
    'Centerline','on',...
    'ViewHeight',3.5,... % (m)
    'ViewLocation',[-8 0]); % [x y] (m)

%% Play scenario
restart(s)
while advance(s)
    pause(s.SampleTime);
end

```

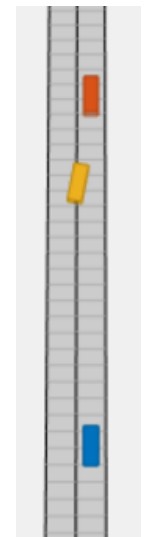


Simulate effects of vision detection sensor



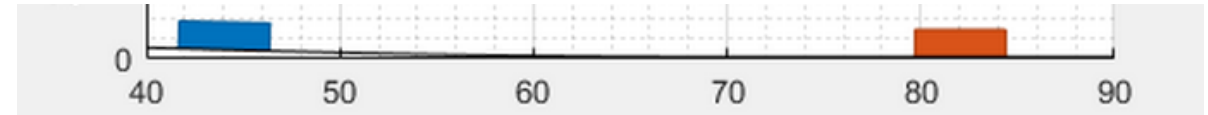
Range effects

- Range measurement accuracy degrades with distance to object
- Angle measurement accuracy consistent throughout coverage area



Occlusion effects

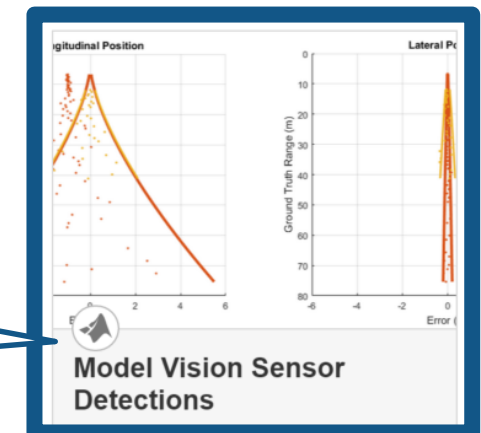
- Partially or completely occluded objects are not detected



Road elevation effects

- Objects may not be detected if they appear above the horizon line
- Large range measurement errors can be introduced for detected objects at different road elevations

Model Vision Sensor Detections
product demo illustrates these effects



Model vision detection sensor

```
%% Create vision detection generator
sensor = visionDetectionGenerator(...
    'SensorLocation', [0.75*egoCar.Wheelbase 0], ...
    'Height', 1.1, ...
    'Pitch', 1, ...
    'Intrinsics', cameraIntrinsics(...
        800,...           % Focal length
        [320 240],...    % Principal point
        [480 640]), ... % Image size
    'RadialDistortion',[0 0], ...
    'TangentialDistortion',[0 0]), ...
    'UpdateInterval', s.SampleTime, ...
    'BoundingBoxAccuracy', 5, ...
    'MaxRange', 150, ...
    'ActorProfiles', actorProfiles(s));
```

Extrinsic mounting parameters

Coverage area is determined based on **cameraIntrinsics**

Model radar detection sensor using **radarDetectionGenerator**

Create birds eye plot to view sensor detections

```

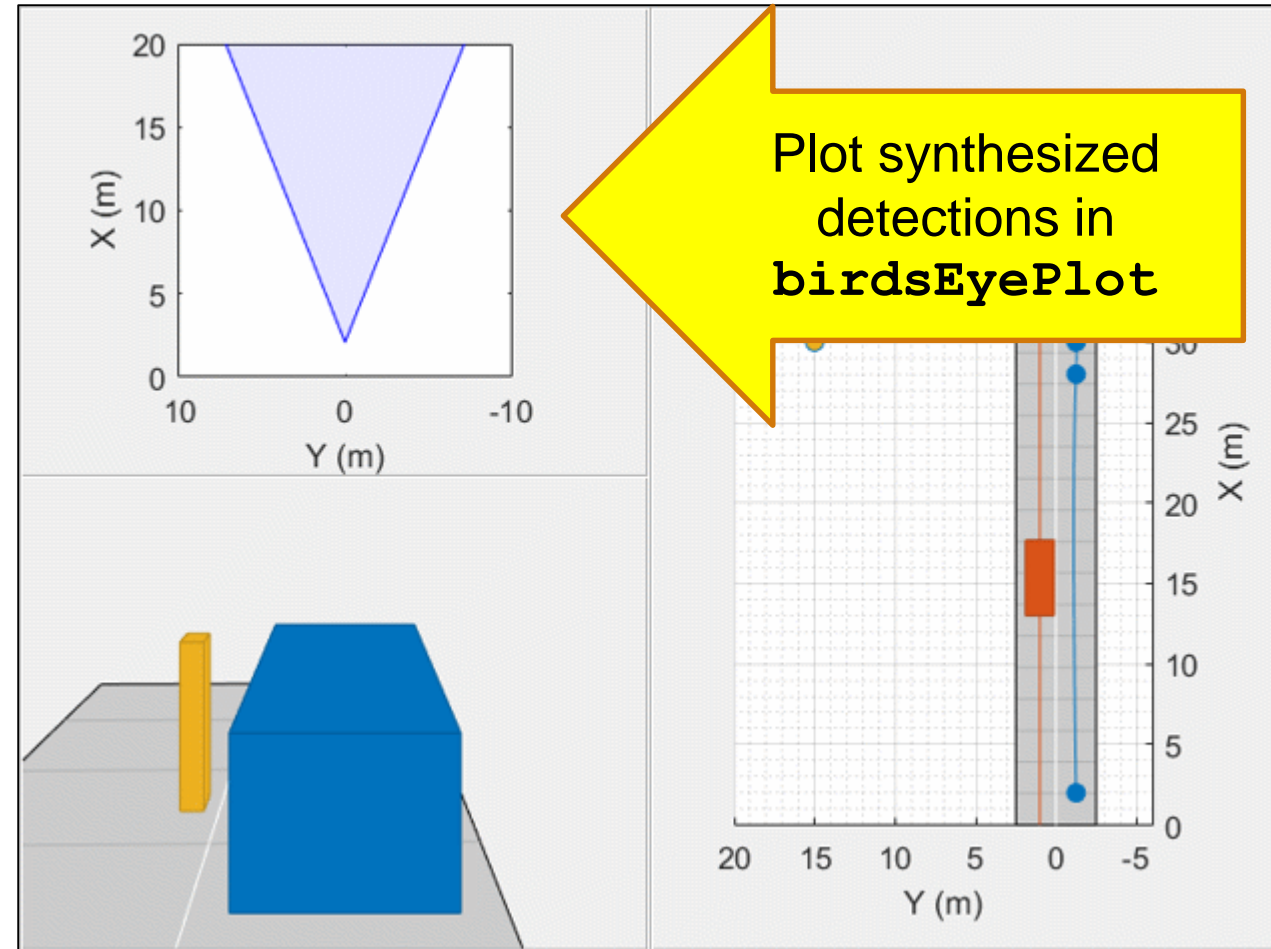
%% Add sensor birds eye plot (top left)
p3 = uipanel('Position',[0 0.5 0.5 0.5]);
a3 = axes('Parent',p3);
bep = birdsEyePlot('Parent',a3,...
    'Xlimits', [0 20],...
    'Ylimits', [-10 10]);
legend(a3,'off');

% Create plotters
covPlot = coverageAreaPlotter(bep,...
    'FaceColor','blue',...
    'EdgeColor','blue');
plotCoverageArea(covPlot,...
    sensor.SensorLocation,sensor.MaxRange,...
    sensor.Yaw,sensor.FieldOfView(1))

detPlot = detectionPlotter(bep,...
    'MarkerEdgeColor','blue',...
    'Marker','^');

truthPlot = outlinePlotter(bep);

```



Play scenario with sensor models

```

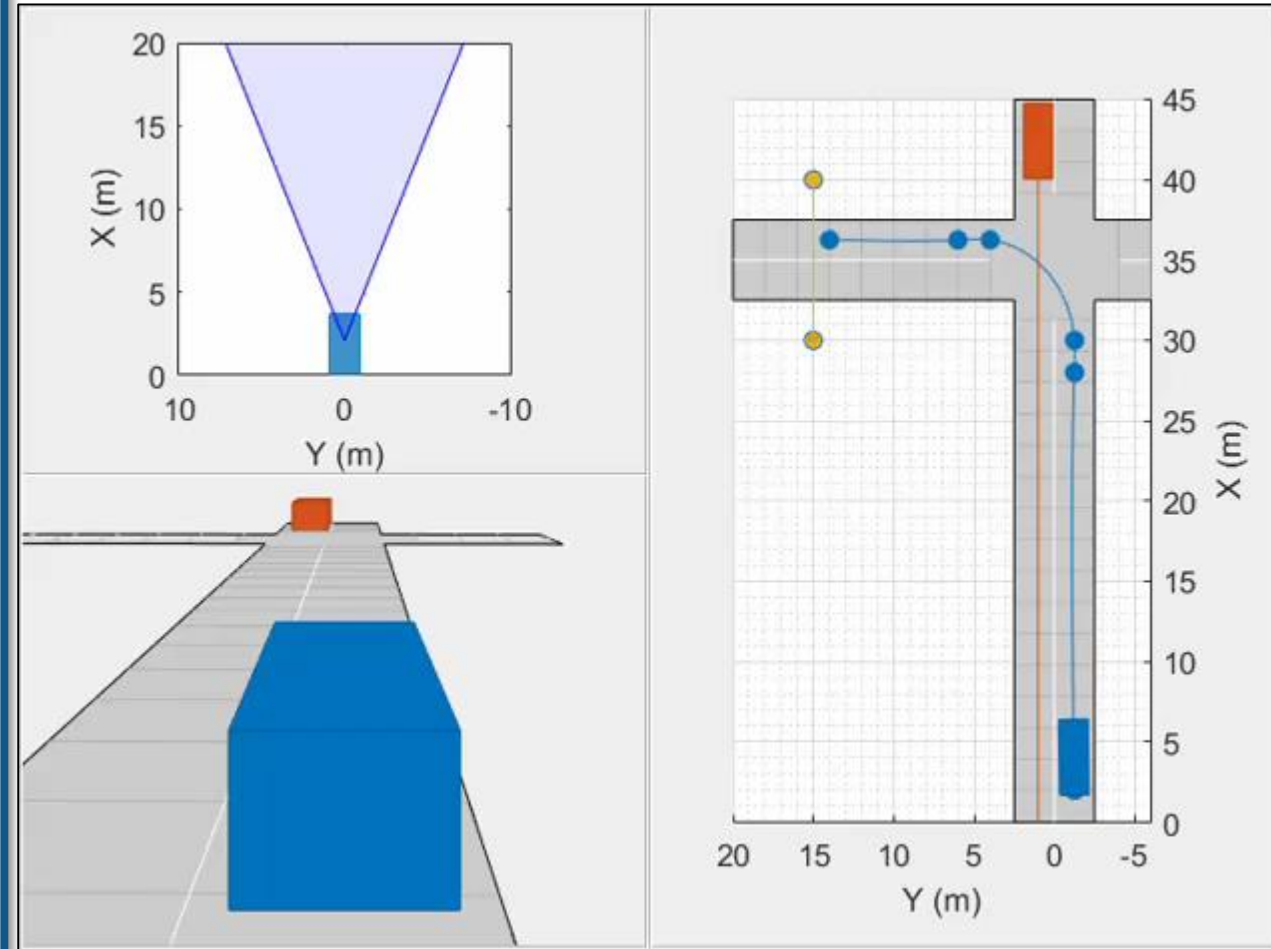
restart(s)
while advance(s)
    % Get detections in ego vehicle coordinates
    det = sensor(targetPoses(egoCar),...
                s.SimulationTime);

    % Update plotters
    if isempty(det)
        clearData(detPlot)
    else % Unpack measurements to position/velocity
        pos = cellfun(@(d)d.Measurement(1:2),...
                    det, 'UniformOutput', false);
        vel = cellfun(@(d)d.Measurement(4:5),...
                    det, 'UniformOutput', false);

        plotDetection(detPlot,...
                    cell2mat(pos'),' , cell2mat(vel)');
    end

    [p, y, l, w, oo, c] = targetOutlines(egoCar);
    plotOutline(truthPlot,p,y,l,w,...
                'OriginOffset', oo, 'Color', c);
end
end

```

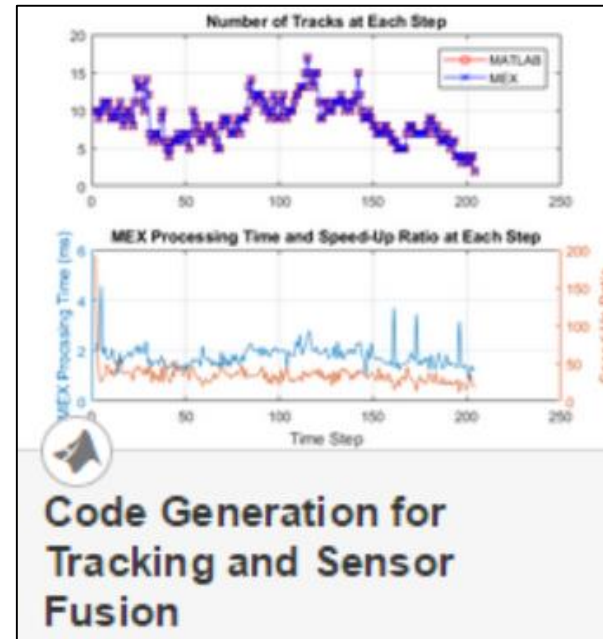


Learn more about sensor fusion

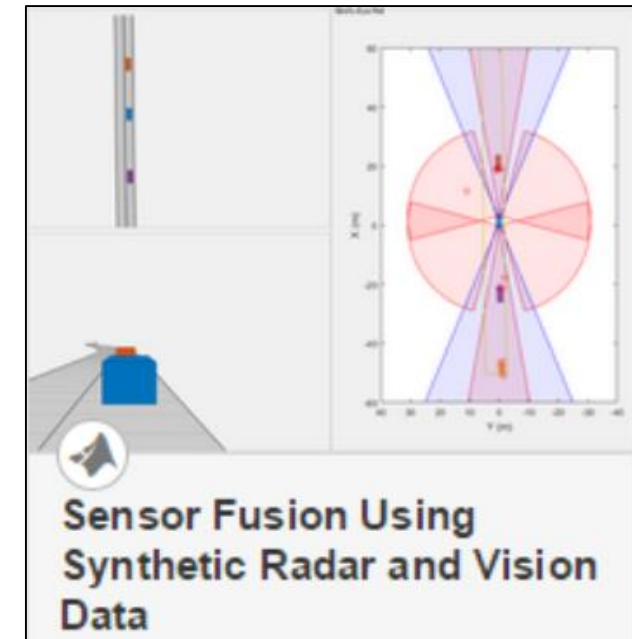
by exploring examples in the Automated Driving System Toolbox



- **Design** multi-object tracker based on logged vehicle data

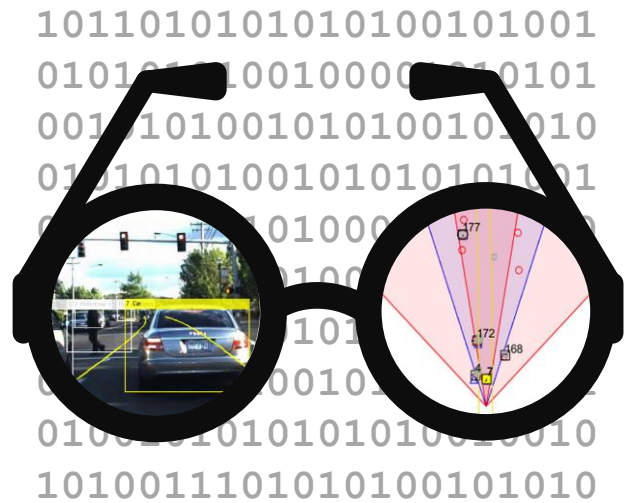


- **Generate C/C++** code from algorithm which includes a multi-object tracker



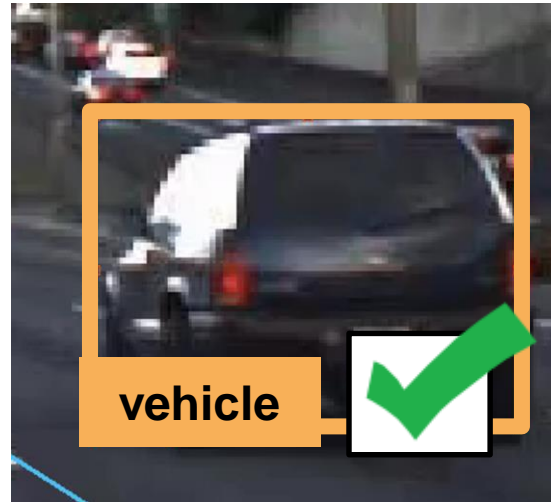
- **Synthesize driving scenario** to test multi-object tracker

The Automated Driving System Toolbox helps you...



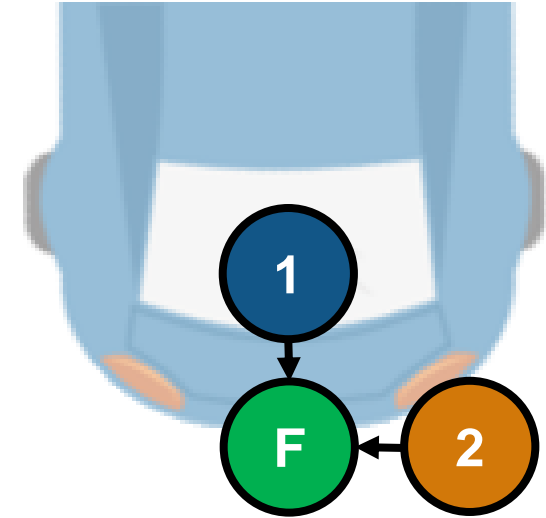
Visualize vehicle data

- Plot sensor detections
- Plot coverage areas
- Transform between image and vehicle coordinates



Detect objects in images

- Train deep learning networks
- Label ground truth
- Connect to other tools



Fuse multiple detections

- Design multi-object tracker
- Generate C/C++
- Synthesize driving scenarios