MATLAB EXPO 2017
KOREA

4월 27일, 서울

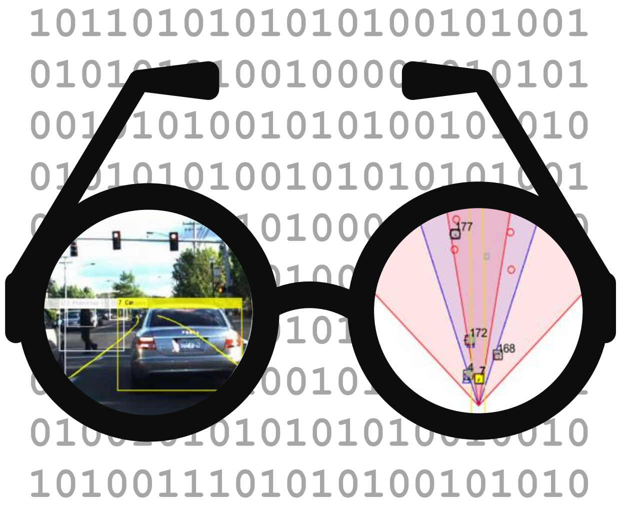
등록 하기 matlabexpo.co.kr

Automated Driving System Toolbox 소개

이제훈 차장

R2017a

Common Questions from Automated Driving Engineers



How can I

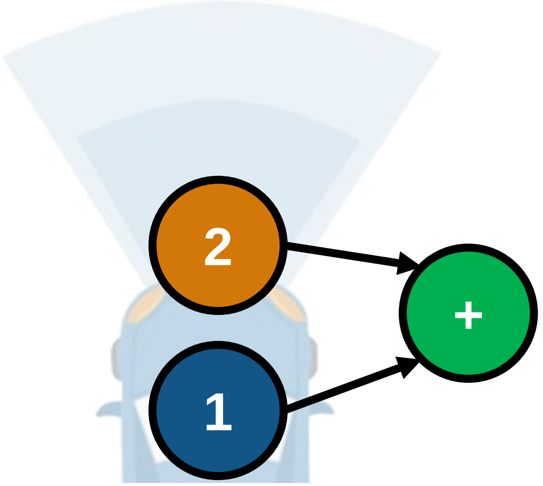
Visualize

Sensor data?

How can I

design and verify

Perception algorithms?

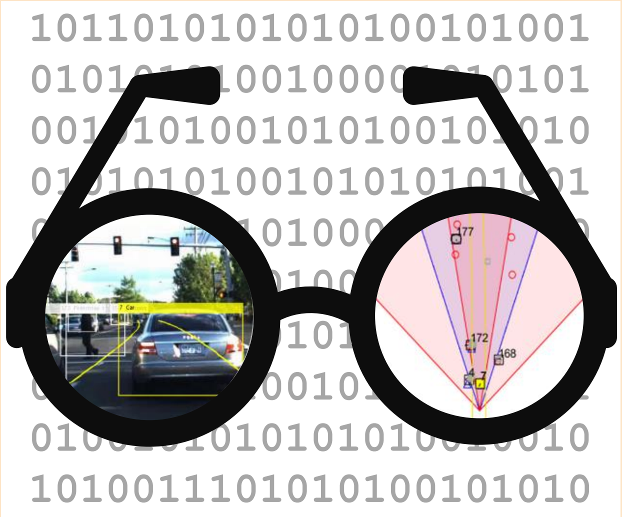


How can I

design and verify

Sensor fusion?

Common Questions from Automated Driving Engineers



How can I

Visualize

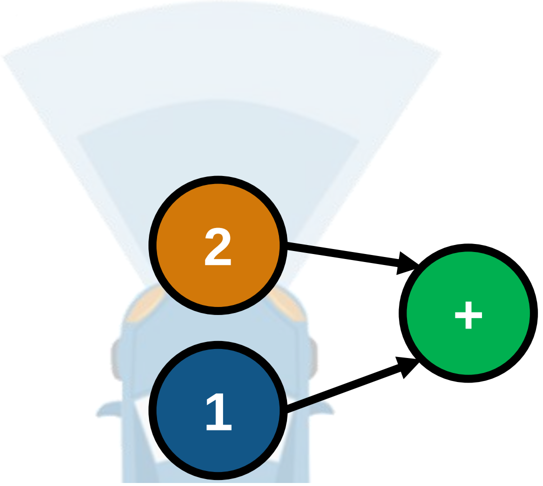
Sensor data?

How can I

design and verify

Perception

algorithms?



How can I

design and verify

Sensor fusion?

Automated Driving **Sensor data**

Camera

Radar

Object Detection

Lidar

Sensor fusion & Tracking

IMU

IMU: Inertial Measurement Unit

Automated Driving **Sensor data**

Camera (640 x 480 x 3)

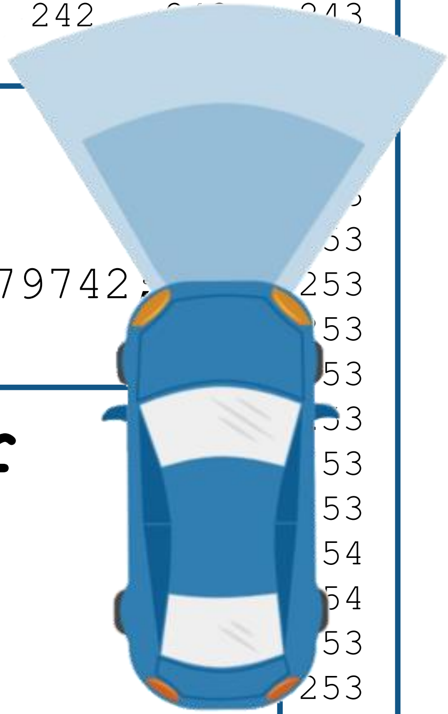
```
239    239    237    238    241    241    241    242
252    252    251    252    252    253    253
```

Vision Detector

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

Lane Detector

```
Left
  IsValid:        1
  Confidence:     3
  BoundaryType:   3
  Offset:         1.68
  HeadingAngle:   0.002
  Curvature:      0.000
Right
  IsValid:        1
```



Radar Detector

```
SensorID       = 2;
Timestamp      = 146163469640752l;
NumDetections  = 23;
Detection
  TrackID
  TrackSt
  Positio
  Velocit
  Amplitu
Detection
  TrackID
  TrackSt
```

Lidar (47197 x 3)

```
-12.2911    1.4790
-14.8852    1.7755
-18.8020    2.2231
-25.7033    3.0119
 -0.0632    0.0815
 -0.0978    0.0855
 -0.2814    0.1064
```

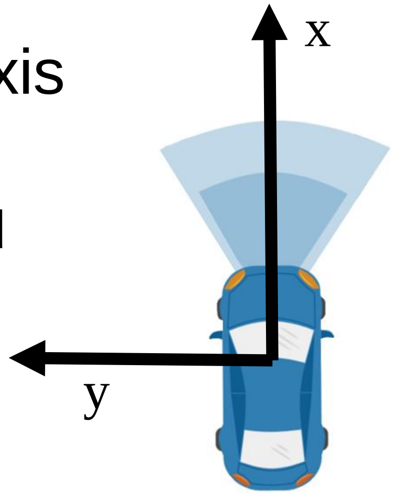
Inertial Measurement Unit

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

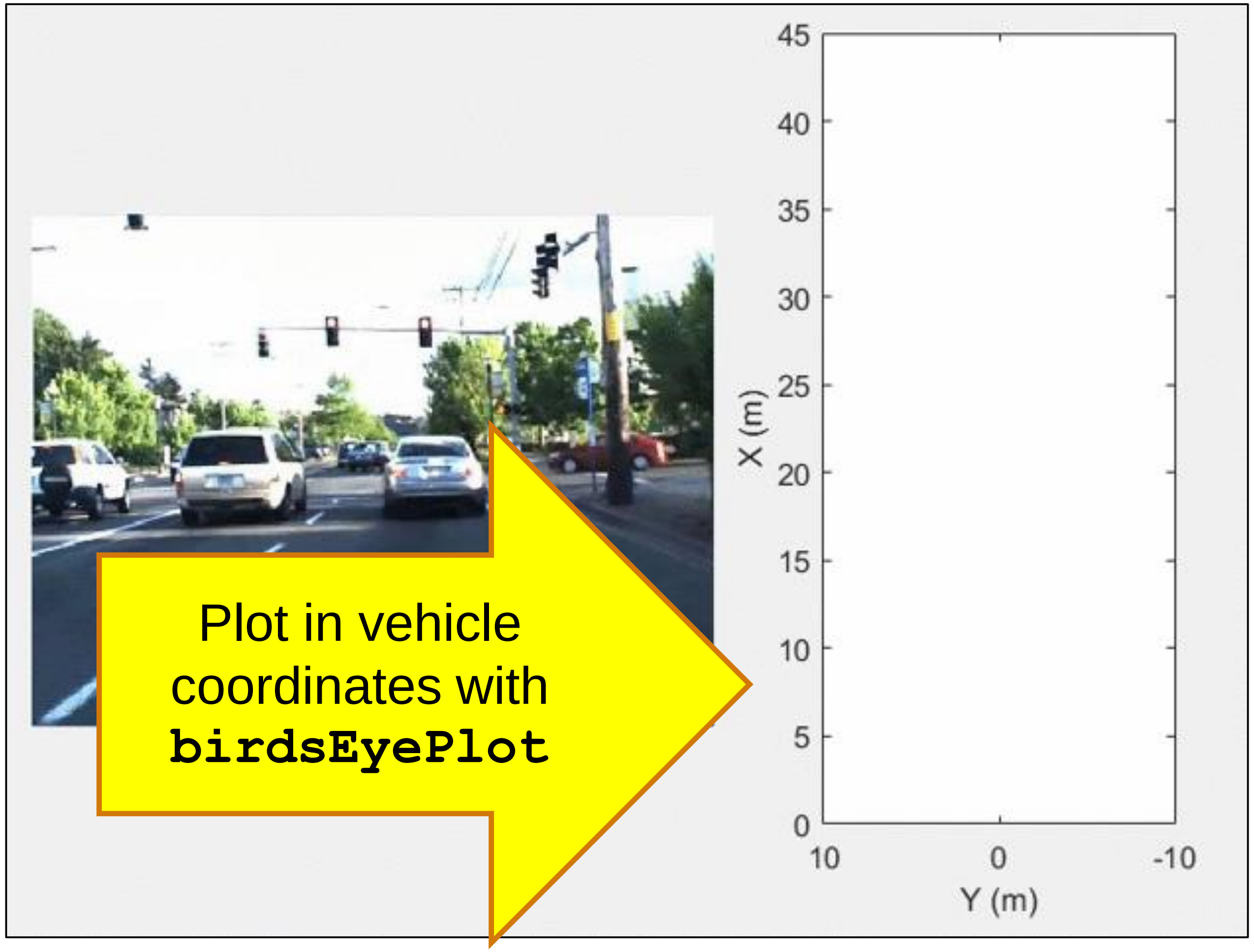
Visualize sensor data

Visualize **Sensor data** in vehicle coordinates

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

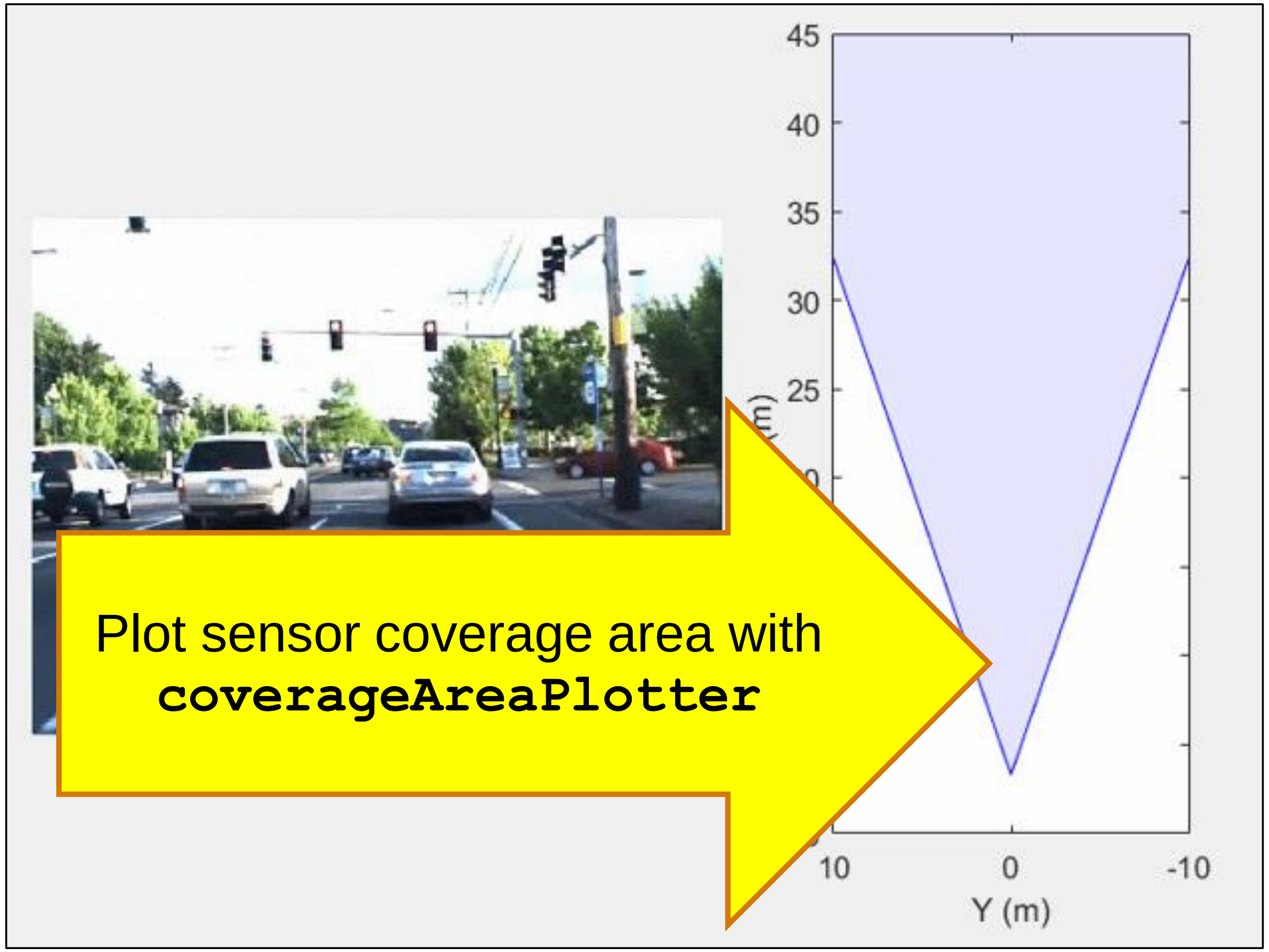



```
%% Plot in vehicle coordinates
ax2 = axes(...
    'Position',[0.6 0.12 0.4 0.85]);
bep = birdsEyePlot(...
    'Parent',ax2,...
    'Xlimits',[0 45],...
    'Ylimits',[-10 10]);
legend('off');
```



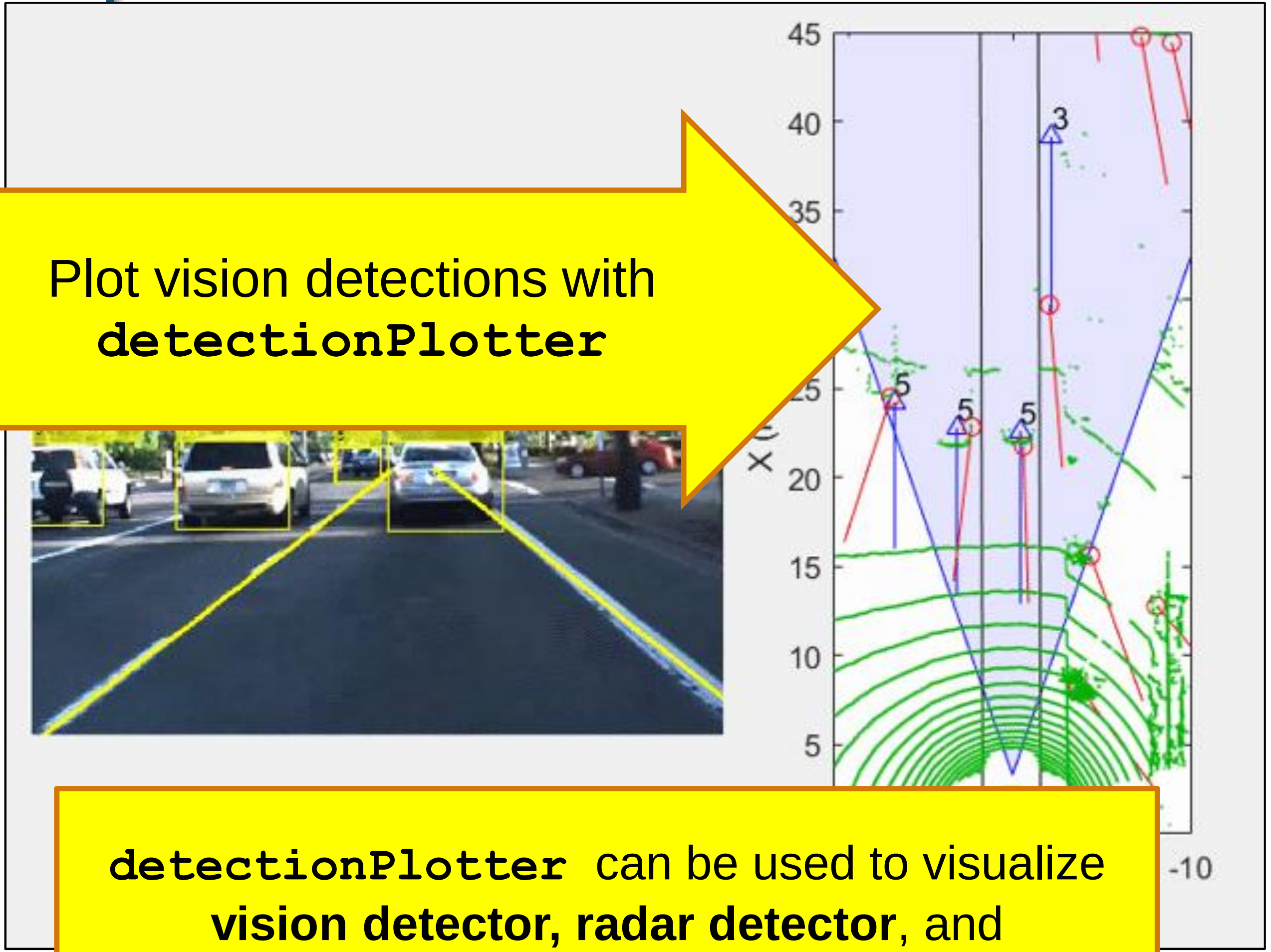
Visualize **Sensor data** - expected coverage area

```
%% 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 **Sensor data** - detected objects (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);
```

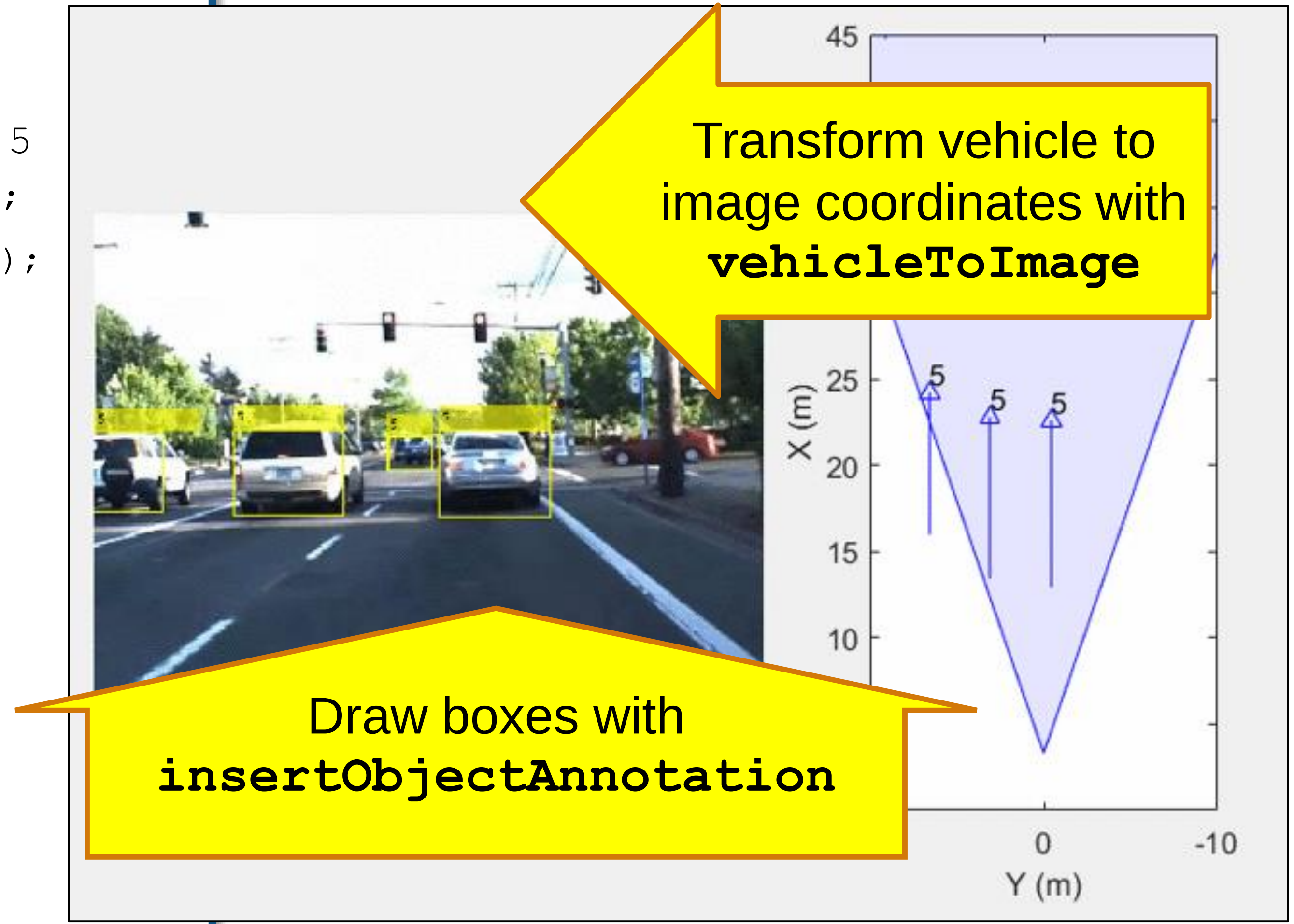


Plot vision detections with **`detectionPlotter`**

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

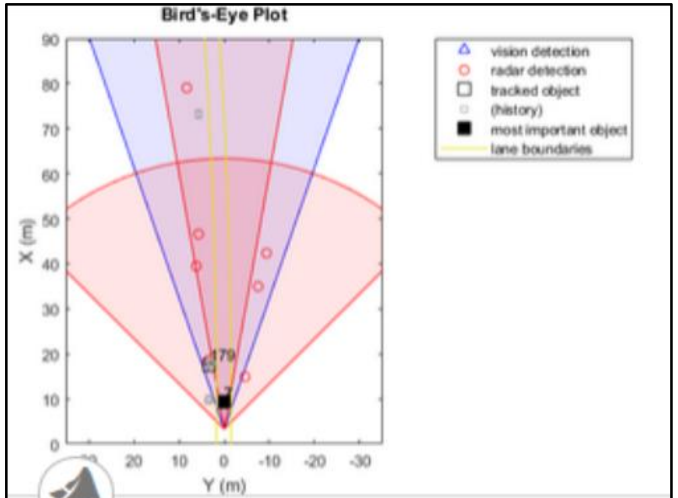
Visualize **Sensor data** - detected objects (image coordinates)

```
%% 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;
```



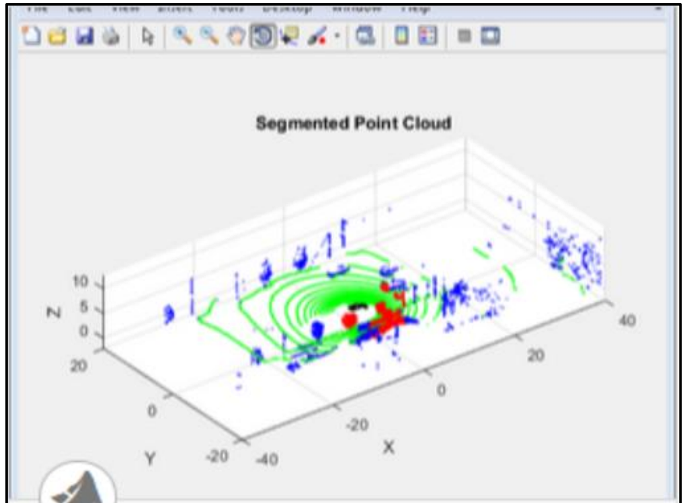
Learn more about visualizing vehicle data

by exploring examples in the Automated Driving System Toolbox R2017a



Visualize Sensor Coverage, Detections, and Tracks

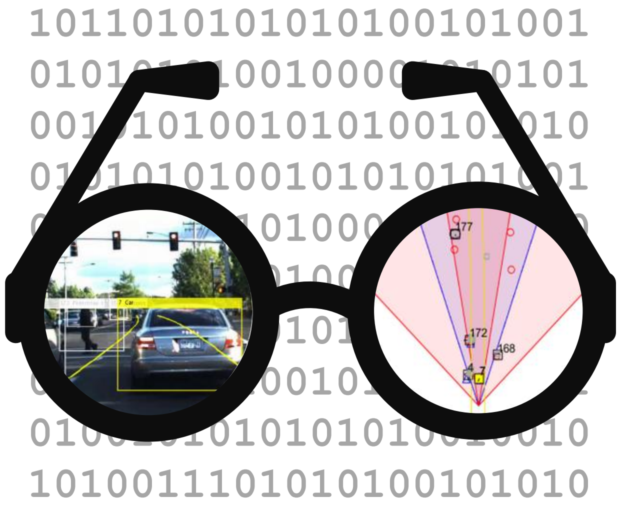
Annotate Video Using Detections in Vehicle Coordinates



Ground Plane and Obstacle Detection Using Lidar

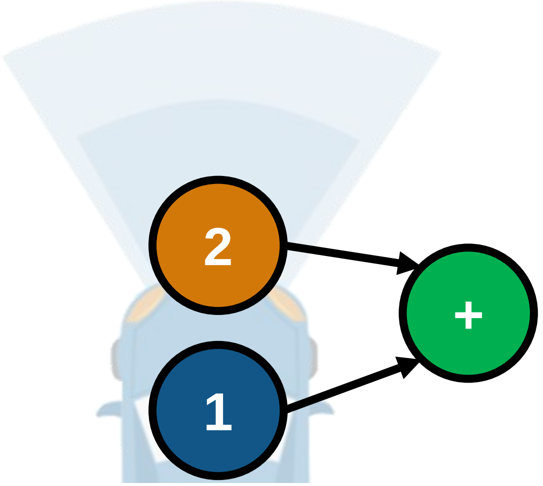
- **Plot object detectors in vehicle coordinates**
  - Vision & radar detector
  - Lane detectors
  - Detector coverage areas
- **Transform between vehicle and image coordinates**
- **Plot lidar point cloud**

Common Questions from Automated Driving Engineers



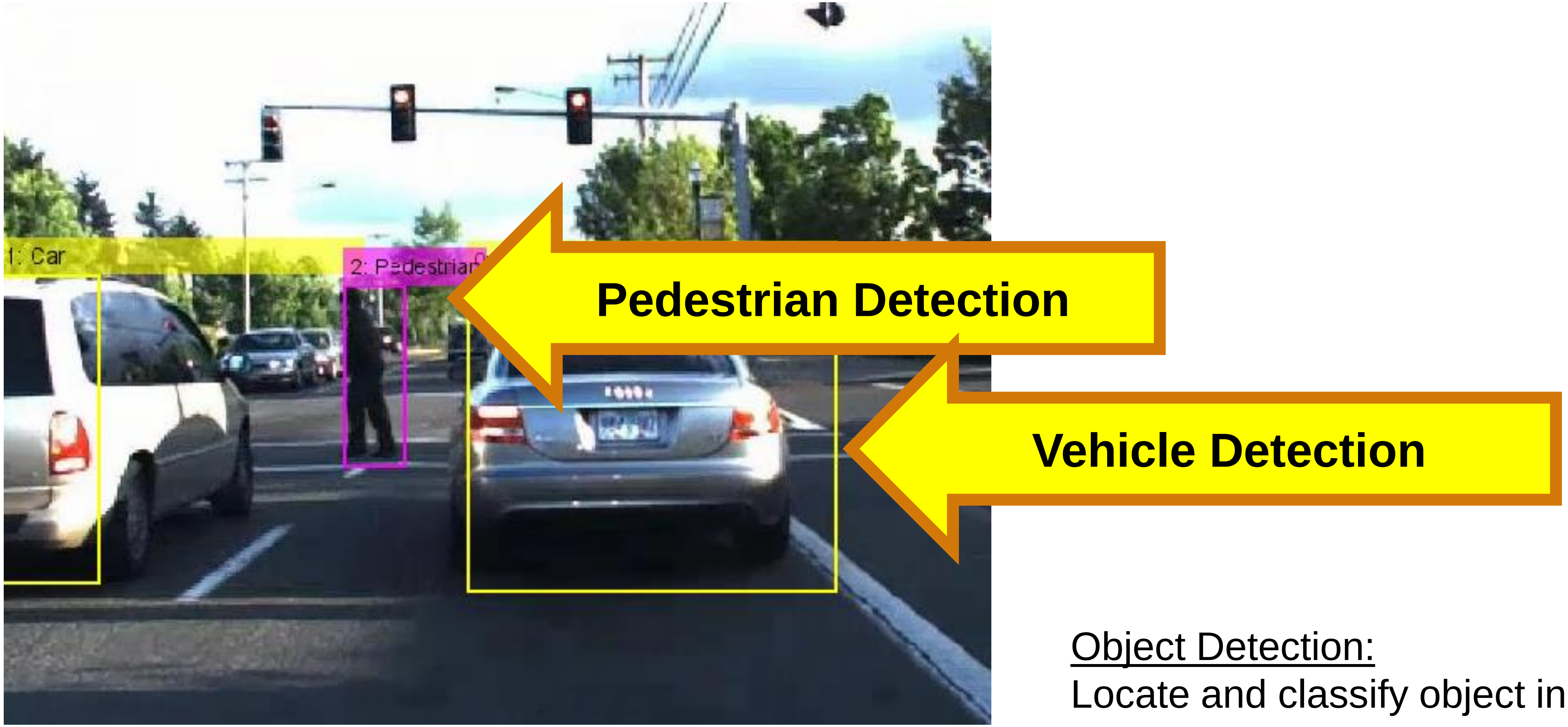
How can I
Visualize
Sensor data?

How can I
design and verify
Perception algorithms?



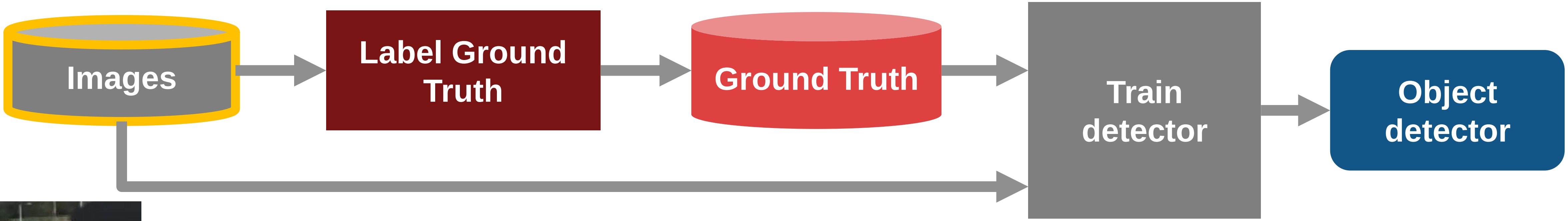
How can I
design and verify
Sensor fusion?

Automated Driving **Perception Algorithms**



<u>Object Detection:</u>
Locate and classify object in image

MATLAB Tools to Train Detectors

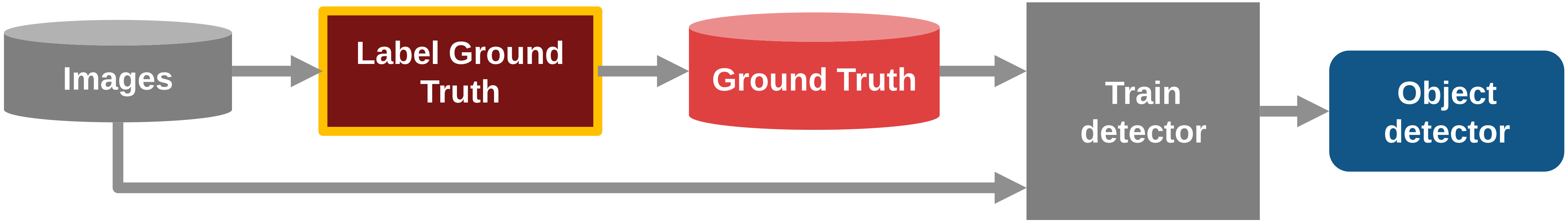


```
imageDS = imageDatastore(dir)
```

Easily manage large sets of images

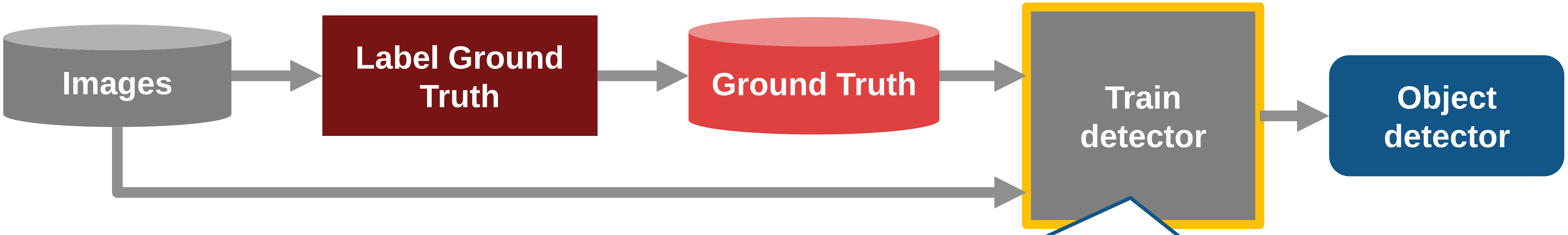
- **Single line of code to access images**
- **Operates on disk, database, big-data file system**

MATLAB Tools to Train Detectors



Automate Labeling of Ground Truth

MATLAB Tools to Train Detectors

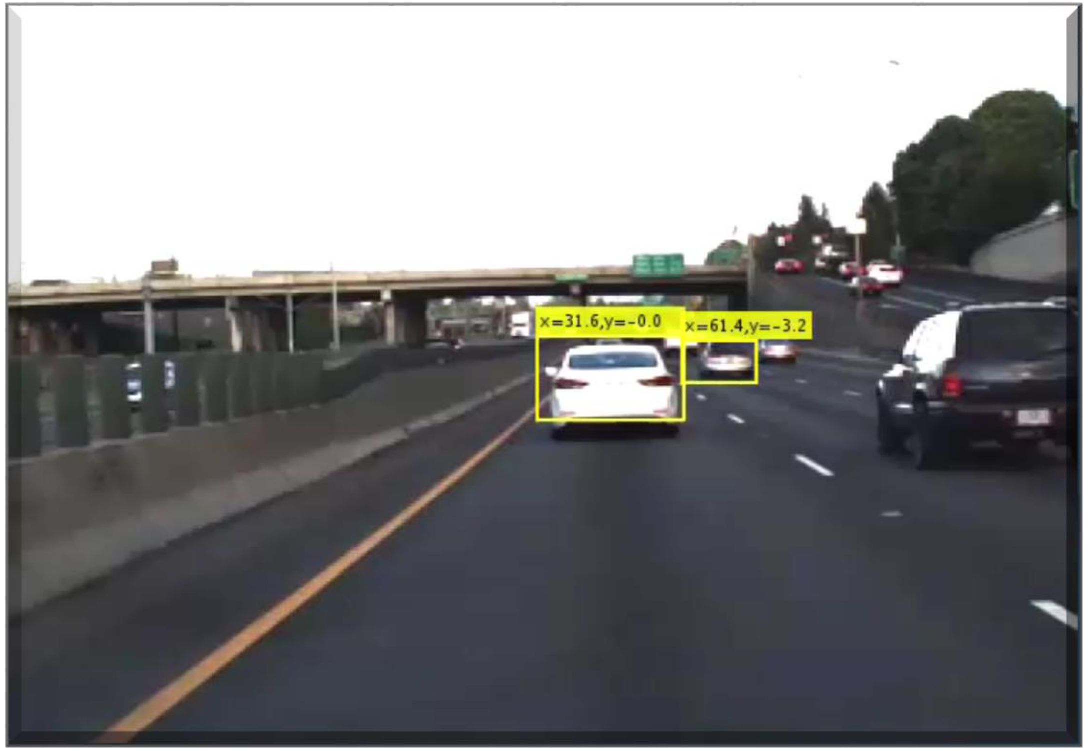


Design object detectors with the Computer Vision System Toolbox

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

Designing **Perception Algorithms**

Computer Vision Algorithms for Automated Driving



Vehicle Detection

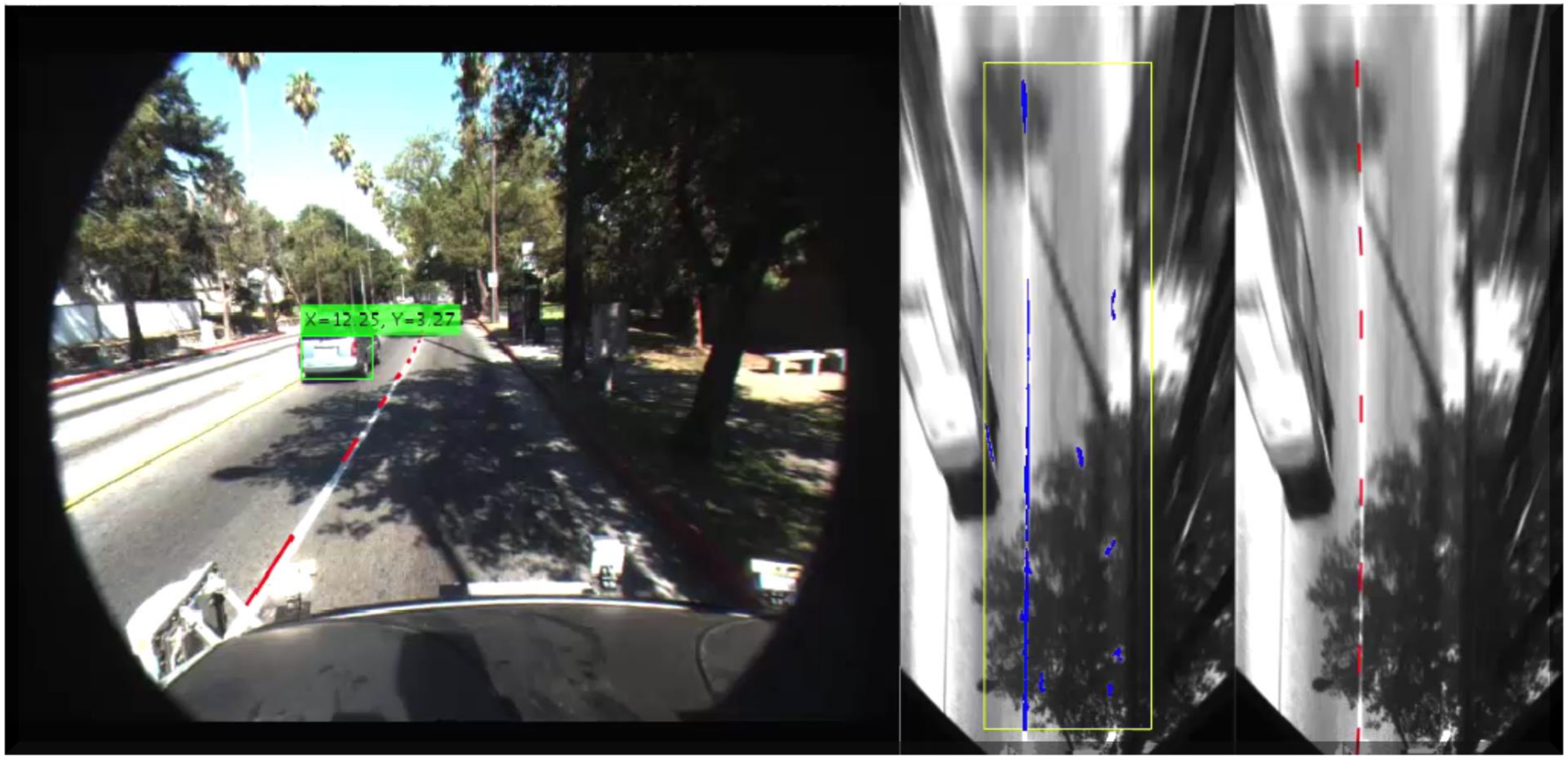
Deep learning and ACF based (pre-trained)

Pedestrian Detection

ACF and HOG/SVM based (pre-trained)

Designing **Perception Algorithms**

Additional Computer Vision Algorithms for Automated Driving



Vehicle detection

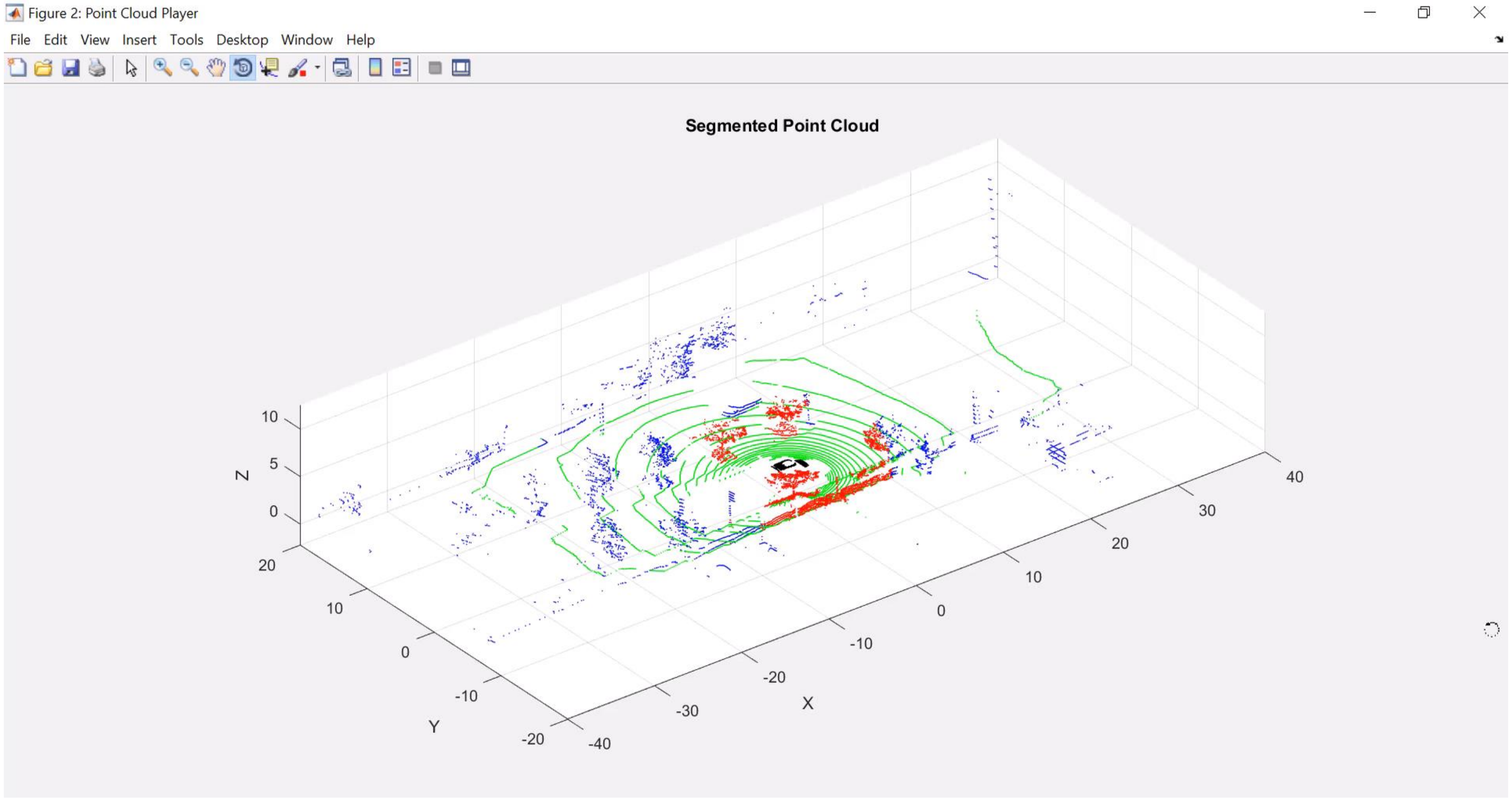
with distance estimation
using mono-camera

Lane Detection and Classification

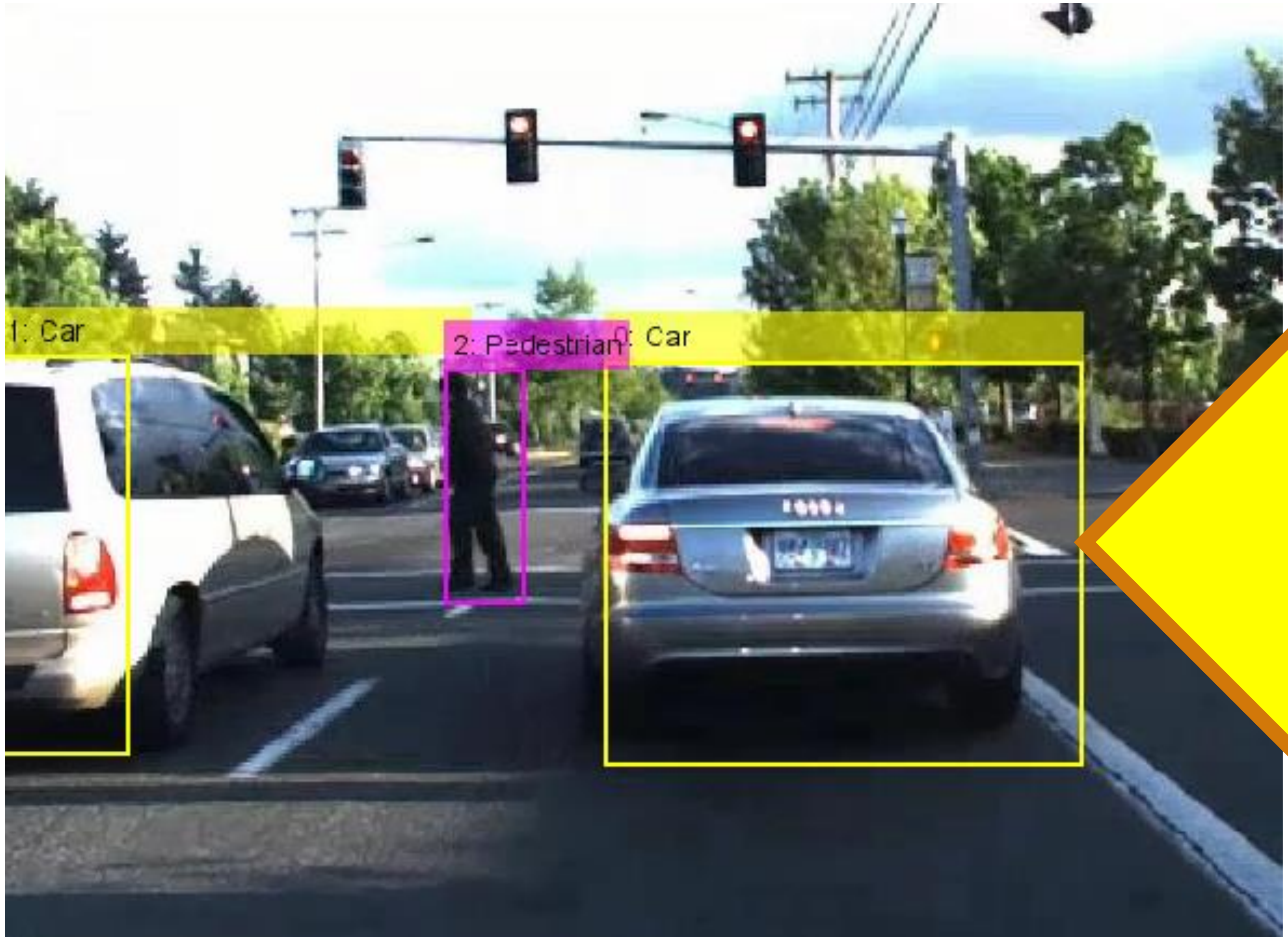
RANSAC-based lane boundary fitting
Lane boundary visualization

Designing **Perception Algorithms**

LiDAR Processing Algorithms

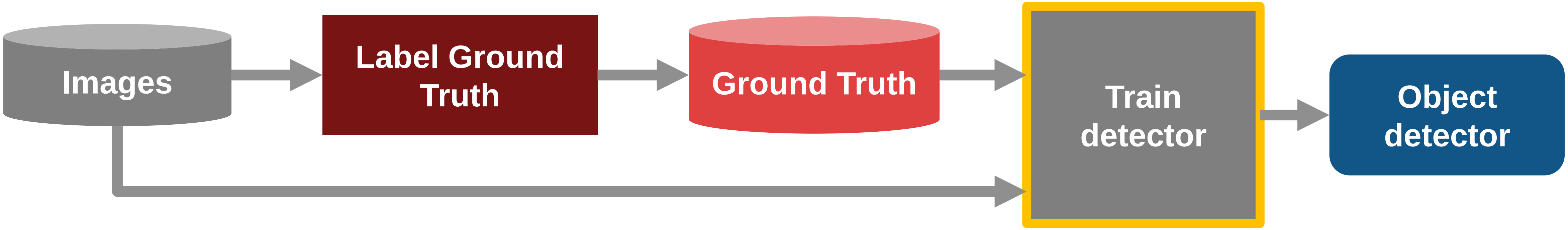


Example of Vision System Detection

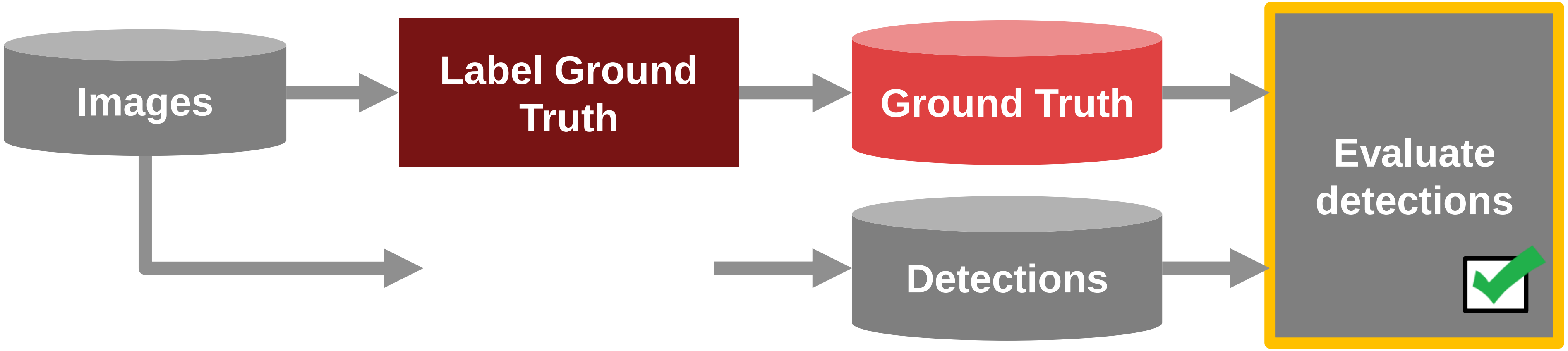


How can I verify this detection is correct?

Ground truth labeling to Train Detectors



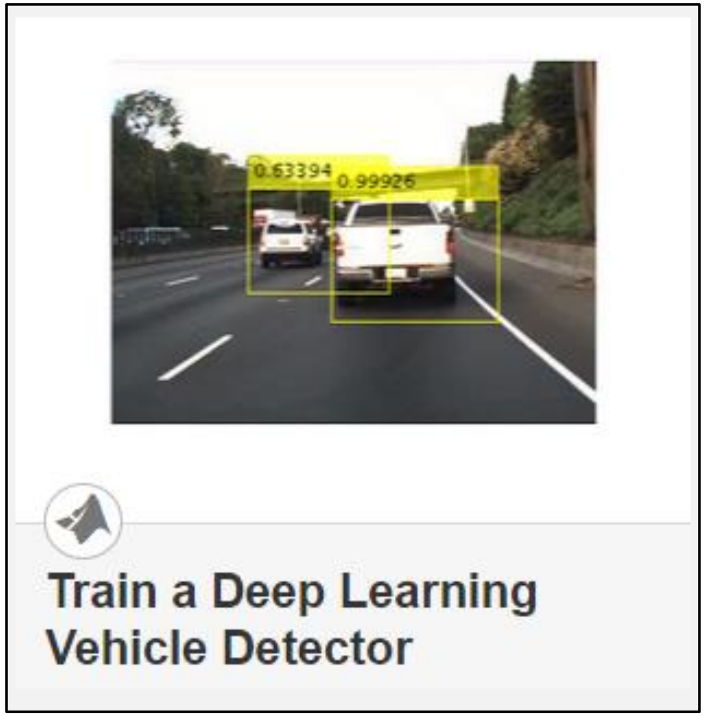
Ground truth labeling to Evaluate Detectors

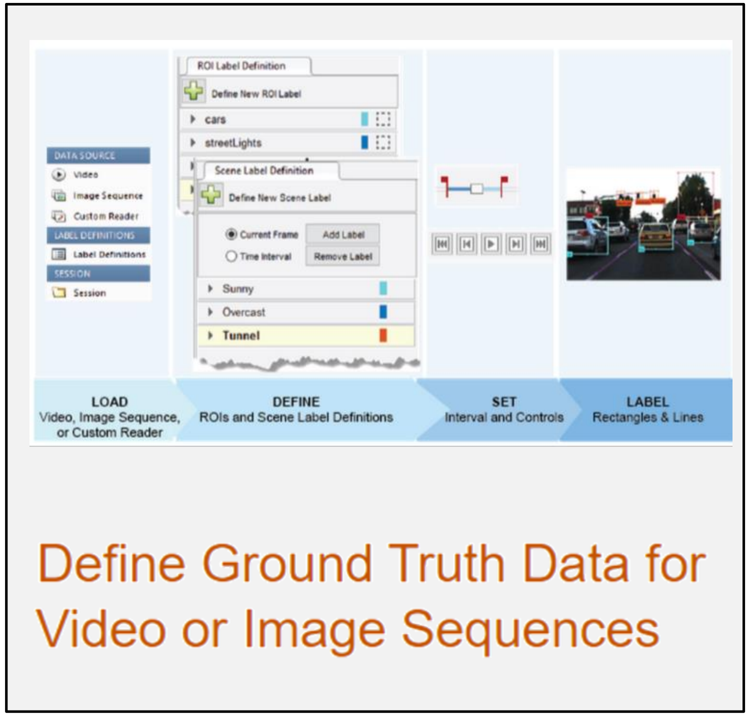


Evaluate
detections against
ground truth

Learn more about verifying perception algorithms

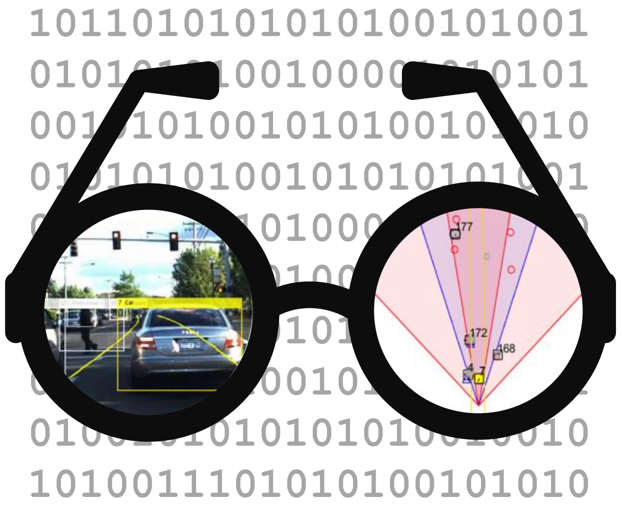
by exploring examples in the Automated Driving System Toolbox R2017a





- **Train object detector** using deep learning and machine learning techniques
- **Label detections** with Ground Truth Labeler App
- **Extend connectivity** of Ground Truth Labeler App

Common Questions from Automated Driving Engineers



How can I

Visualize

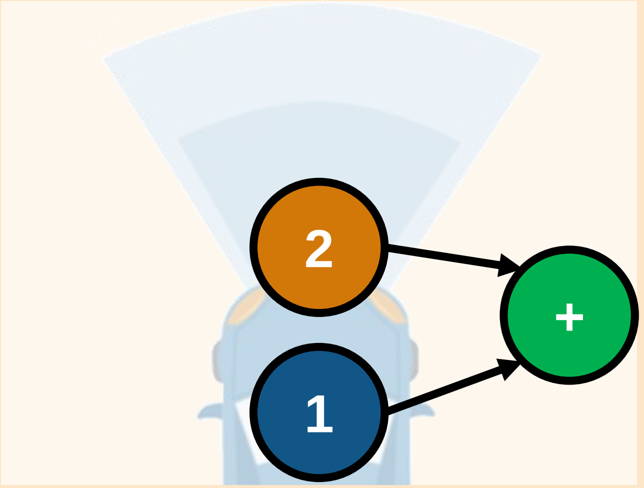
Sensor data?

How can I

design and verify

Perception

algorithms?

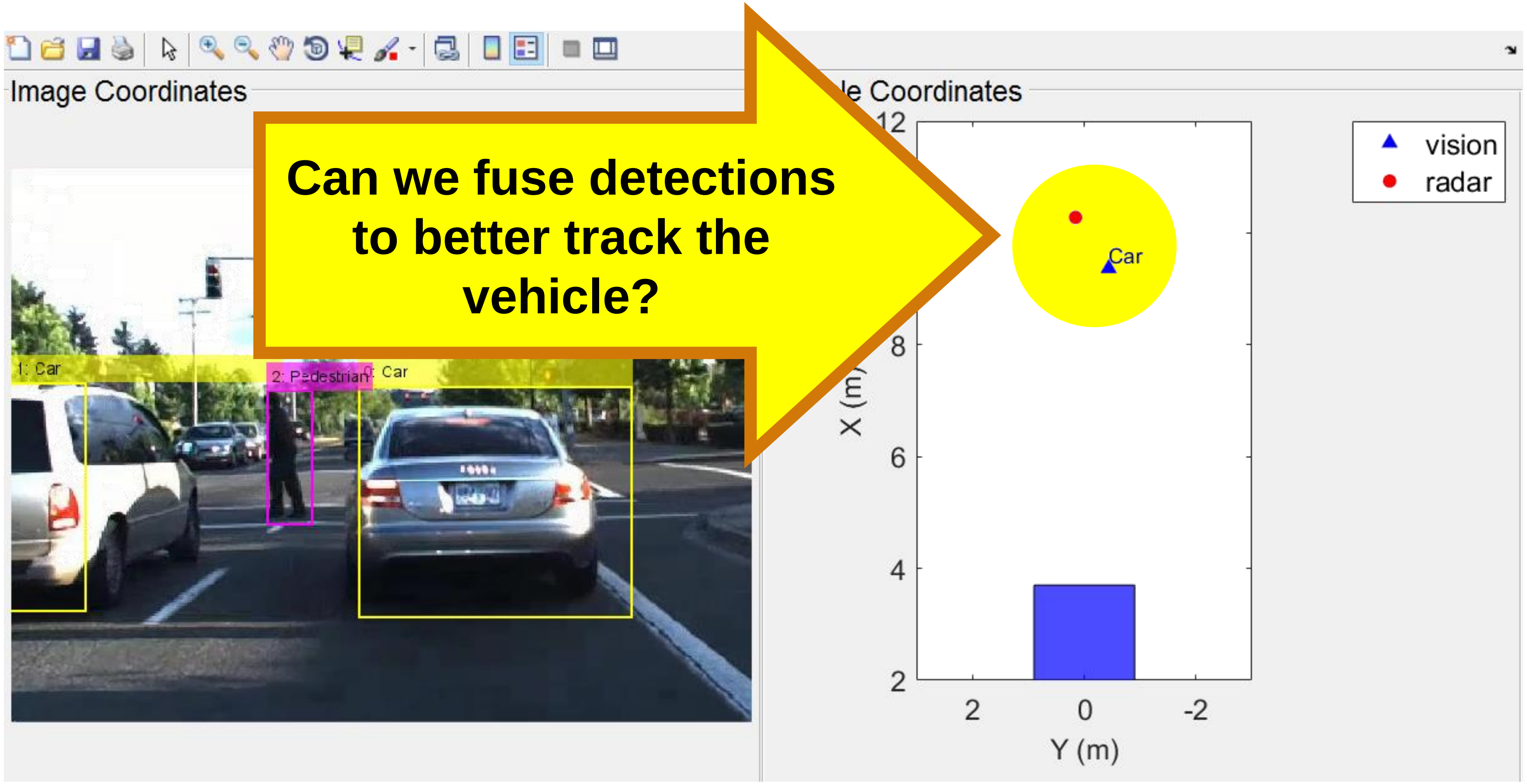


How can I

design and verify

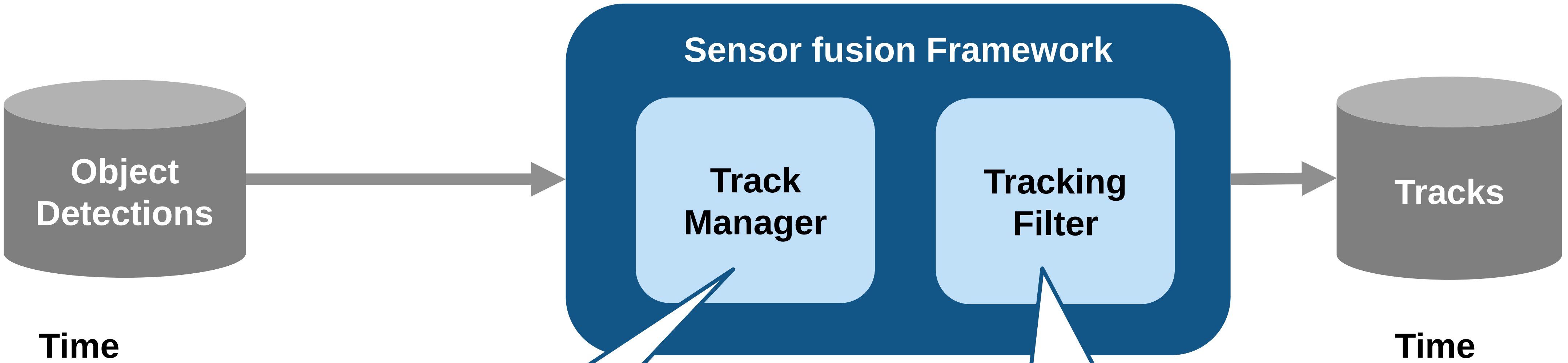
Sensor fusion?

Automated Driving **Sensor fusion** with radar and vision



Design
multi-object tracker

Sensor fusion framework



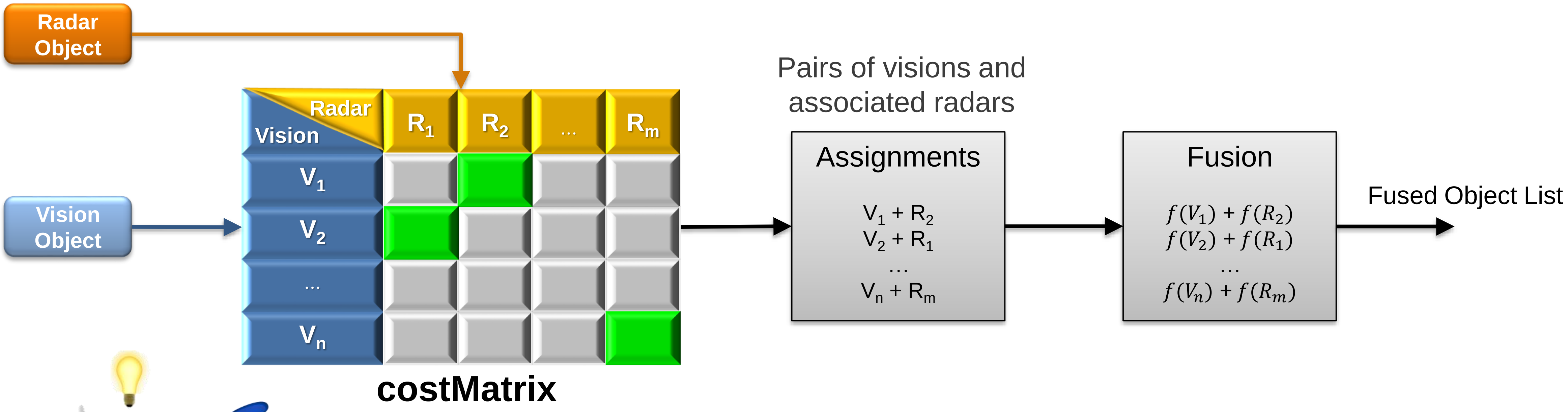
Time
Measurement
Measurement Noise

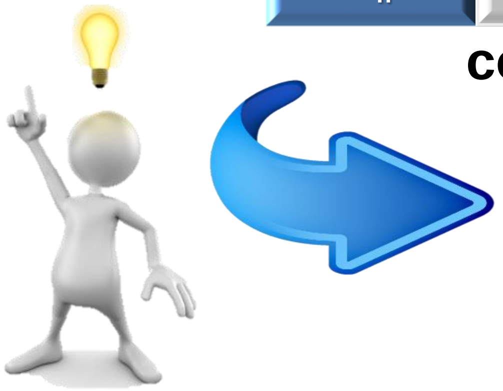
- Assigns detections to tracks
- Creates new tracks
- Updates existing tracks
- Removes old tracks

- Predicts and updates state of track
- Supports linear, extended, and unscented Kalman filters

Time
State
State Covariance
Track ID
Age
Is Confirmed
Is Coasted

Sensor fusion - Data Association





```
[assignments, unassignedVisions, unassignedRadars] = ...
    assignDetectionsToTracks(costMatrix, param.costOfNonAssignment);
```

Sensor fusion - Kalman Filter

Initial state & covariance

$\hat{\mathbf{x}}_0$, $\mathbf{P}_0$

Previous state & covariance

$\hat{\mathbf{x}}_{k-1}$, $\mathbf{P}_{k-1}$

$k \rightarrow k-1$

Current becomes previous

Time Update ("Predict")

(1) Predict state based on physical model and previous state

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k$$

(2) Predict error covariance matrix

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}$$

$\mathbf{u}$: Control variable matrix

$\mathbf{w}$: Process (state) noise

$\mathbf{P}_k^- = E[\mathbf{e}_k^- \mathbf{e}_k^{-T}]$, $\mathbf{e}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^-$: Process (state) covariance matrix (estimation error)

$\mathbf{Q} = E[\mathbf{w}\mathbf{w}^T]$: Process noise covariance matrix

$\mathbf{A}$: State matrix relates the state at the previous, k-1 to the state at the current, k

Measurement Update ("Correct")

(1) Compute Kalman gain

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R})^{-1}$$

(2) Update estimate state with measurement

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-)$$

(3) Update the error covariance matrix

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-$$

minimize $\mathbf{P}_k$

Measurement

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

$\mathbf{v}$: Measurement noise

$\mathbf{H}$: Output matrix relates the state to the measurement

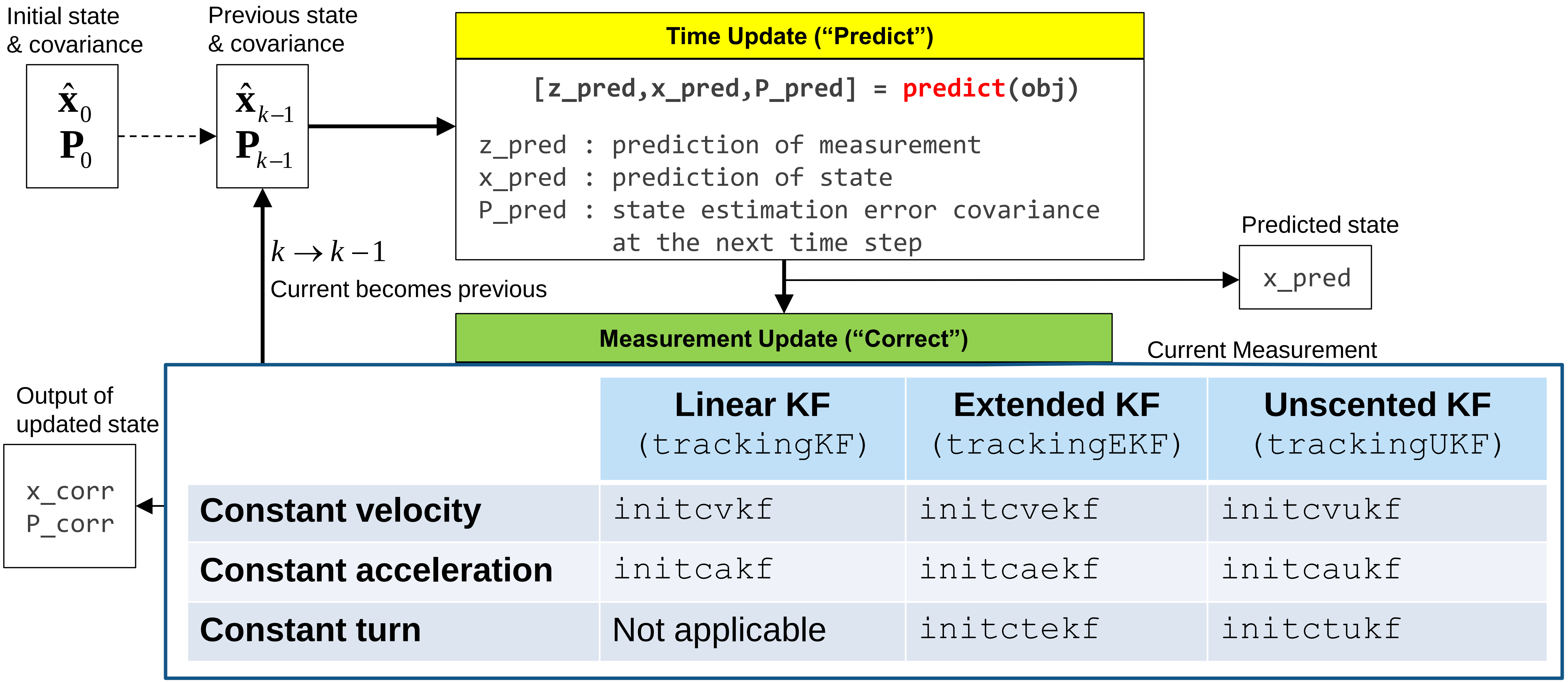
Output of updated state

$\hat{\mathbf{x}}_k$, $\mathbf{P}_k$

$\mathbf{K}$: Kalman gain

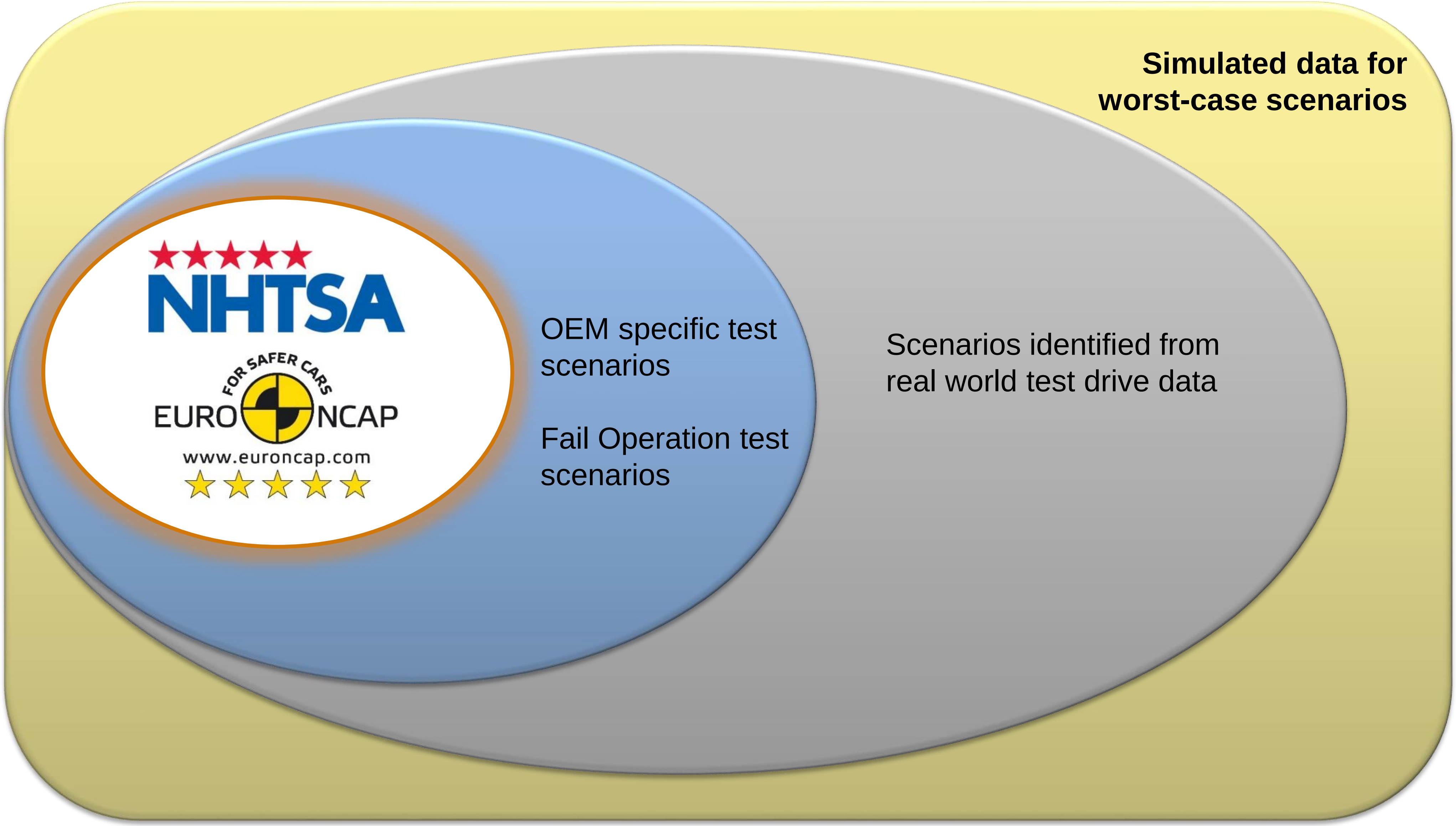
$\mathbf{R}$: Sensor noise covariance matrix (measurement error) ← From sensor spec or experiment

Sensor fusion - Kalman Filter



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

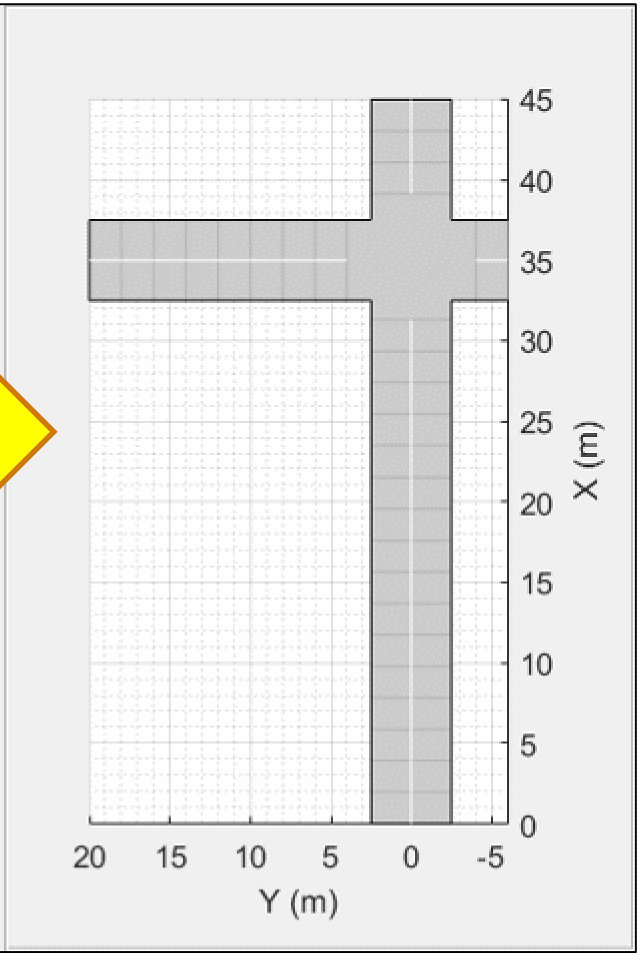
Synthesize Driving Scenario for **Sensor fusion**



Synthesize Driving Scenario for **Sensor fusion**

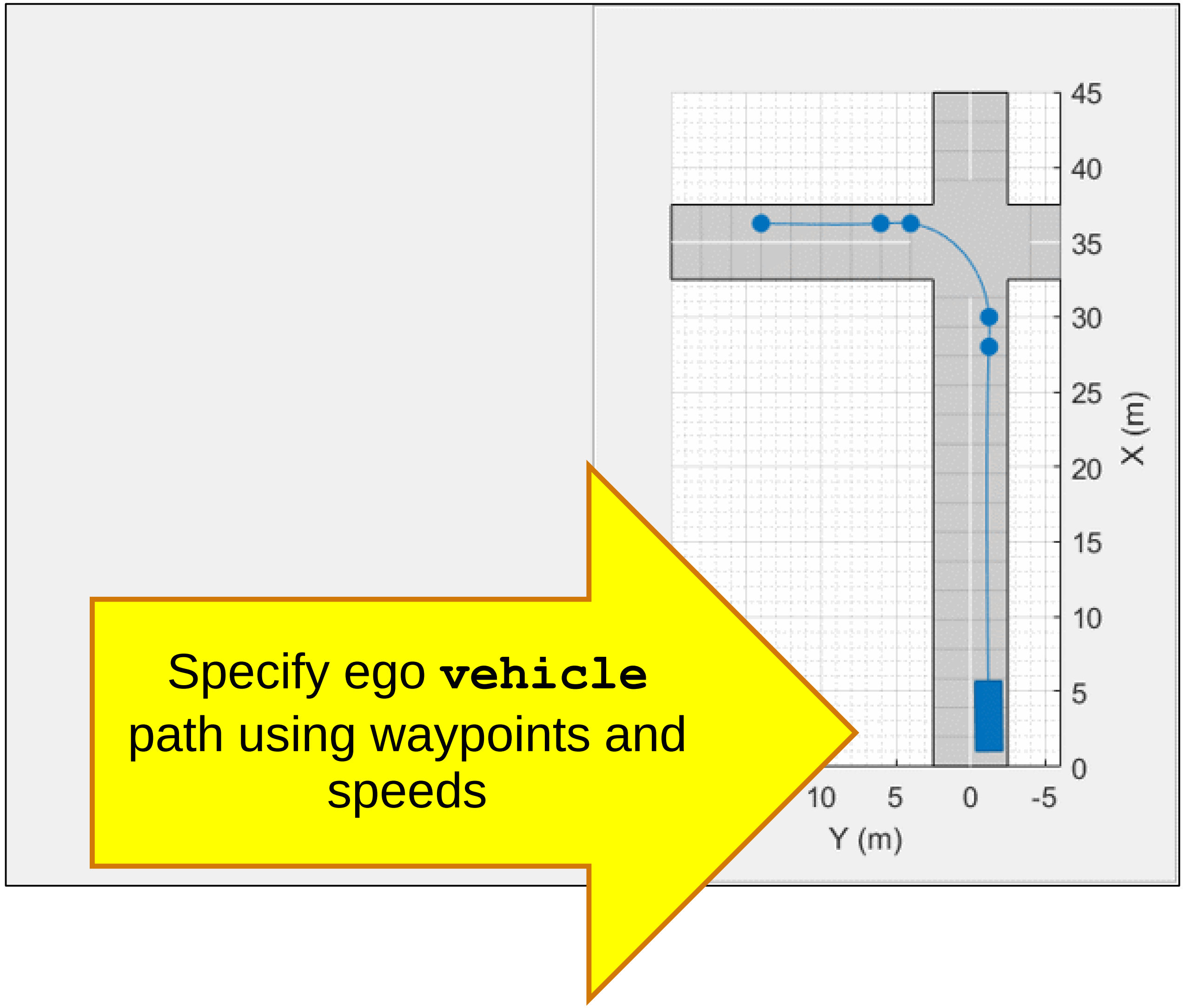
```
%% Create a new scenario
s = drivingScenario('SampleTime', 0.05);

%% 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**



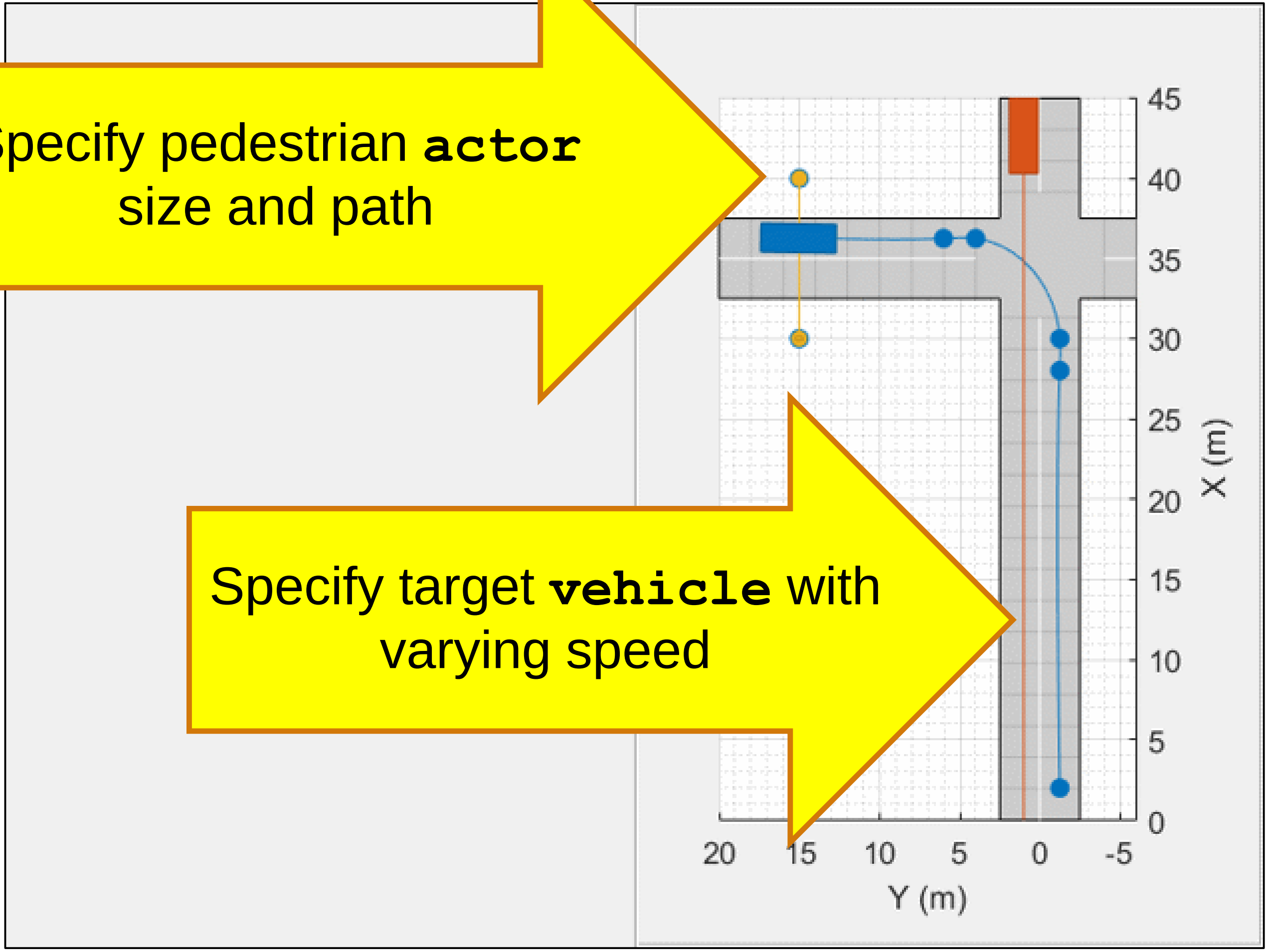
Synthesize Driving Scenario for **Sensor fusion**

```
%% 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
```



Synthesize Driving Scenario for **Sensor fusion**

```
%% 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


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



Specify pedestrian **actor** size and path

Specify target **vehicle** with varying speed

Synthesize Driving Scenario for **Sensor fusion**

```
radarSensor =

  radarDetectionGenerator with properties:

              SensorIndex: 1
           UpdateInterval: 0.1000

           SensorLocation: [3.4000 0]
                   Height: 0.2000
                      Yaw: 0
                    Pitch: 0
                     Roll: 0

              FieldOfView: [20 5]
                 MaxRange: 150
          RangeRateLimits: [-100 100]

     DetectionProbability: 0.9000
           FalseAlarmRate: 1.0000e-06

  Show all properties
```

Measurement rate

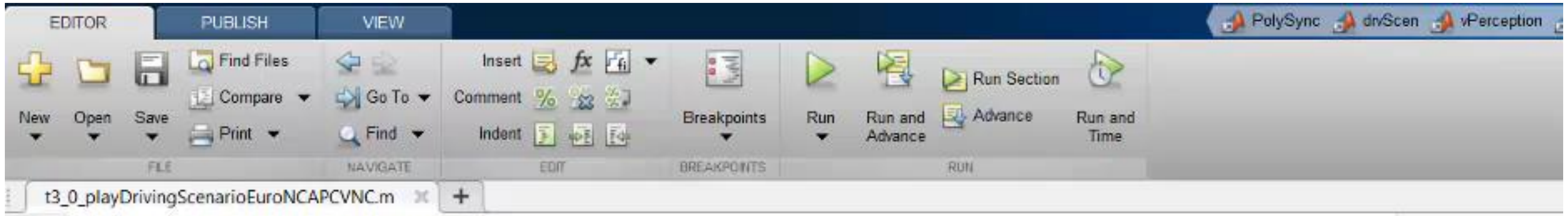
Mounting position on car

Most common params, e.g. coverage

More params (resolution, bias, etc.…)

```
visionSensor =

  visionDetectionGenerator with properties:

              SensorIndex: 1
           UpdateInterval: 0.1000

           SensorLocation: [1.9000 0]
                   Height: 1.1000
                      Yaw: 0
                    Pitch: 1
                     Roll: 0
               Intrinsics: [1×1 cameraIntrinsics]

              FieldOfView: [43.6028 33.3985]
                 MaxRange: 150
                 MaxSpeed: 50
      MaxAllowedOcclusion: 0.5000
       MinObjectImageSize: [15 15]

     DetectionProbability: 0.9000
    FalsePositivesPerImage: 0.1000

  Show all properties
```

Euro NCAP TEST PROTOCOL – AEB VRU systems




```
%% Create a new scenario
s = drivingScenario;
s.SampleTime = 0.05;

%% Create road
RoadCenters = [0 0 0; 50 0 0];
road(s, RoadCenters, 10);

%% Add actors
% --- moving ego vehicle towards a child pedestrian crossing
egoCar = vehicle(s,'Position',[0.4 -1 0],'Yaw',180);
Waypoints = [0.4 -1; 36 -1]; % in meters
Speed = 13.89; % egoCar speed = 13.89 m/s = 50 km/hr
path(egoCar, Waypoints, Speed); % create egoCar path

% --- two stationary cars
vehicle(s,'Position',[35.3 -3.8 0]);
vehicle(s,'Position',[29.6 -3.8 0]);

% --- child pedestrian crossing it's path running from behind of stationary
% cars
child = actor(s,'Length',0.24,'Width',0.45,'Height',1.7,...
    'Position',[40 -5 0],'Yaw',180);
Waypoints = [40 -5; 40 10]; % in meters
Speed = 1.39; % child speed = 1.39 m/s = 5 km/hr
path(child, Waypoints, Speed); % create child path
```

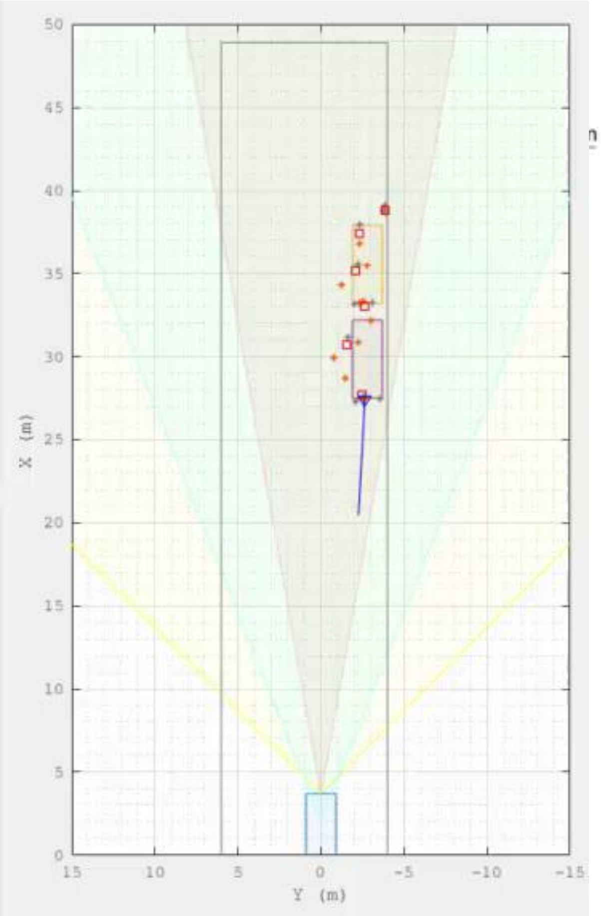
A ---

20-60

ımy H-point
under Test
:tion vehicles

(running)
›struction vehi
ıer Test and laı

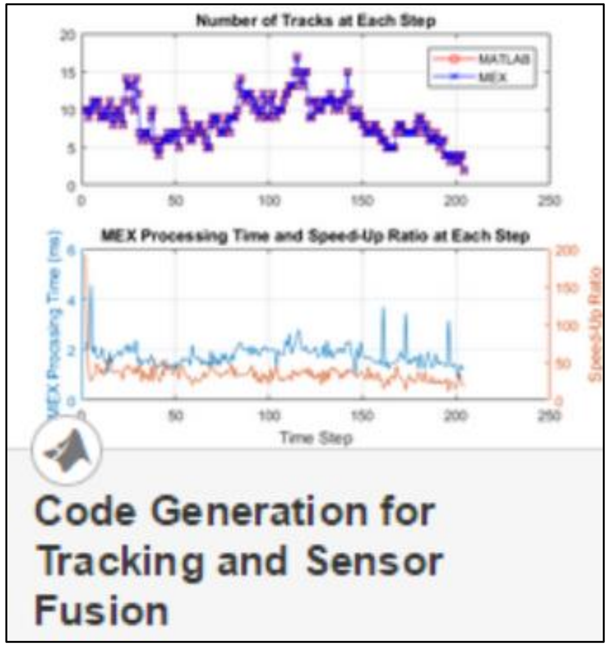
narios
›-point)

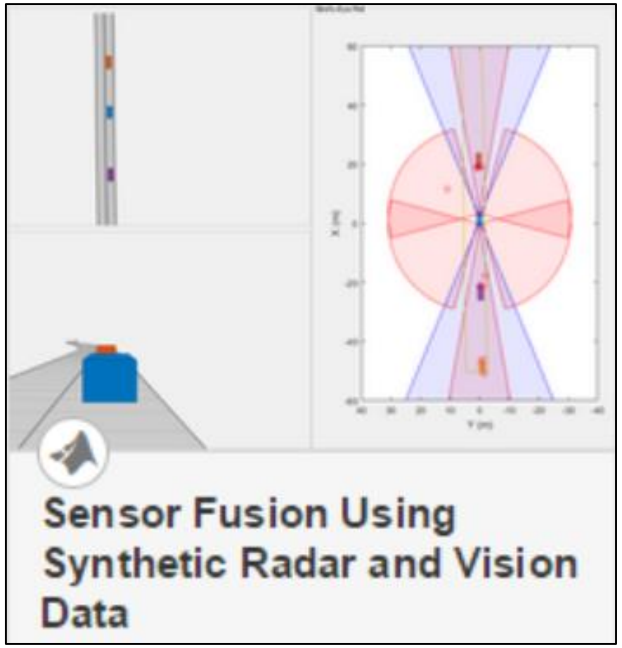


C scenario, Running Child from Nearside
from Obstruction vehicles (see Annex B)

Learn more about sensor fusion

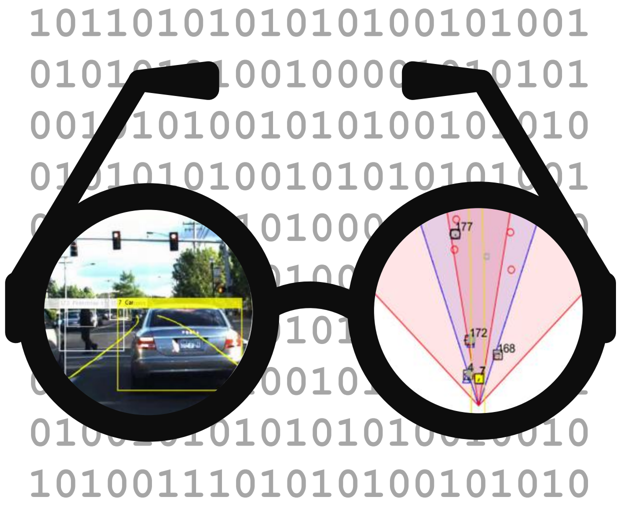
by exploring examples in the Automated Driving System Toolbox R2017a





- **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

Common Questions from Automated Driving Engineers



How can I

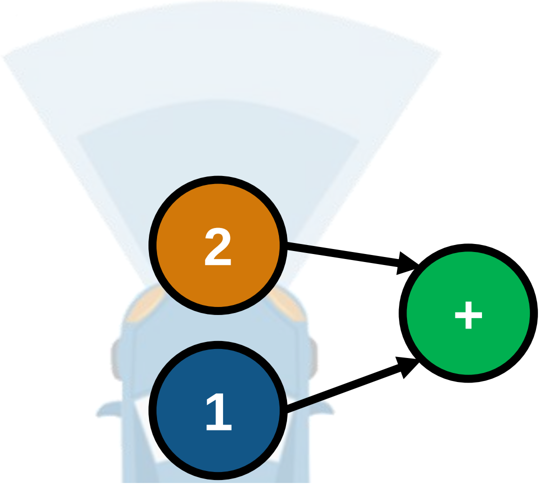
Visualize

Sensor data?

How can I

design and verify

Perception algorithms?



How can I

design and verify

Sensor fusion?

% Thank you