

MATLAB EXPO 2018

Automated Driving Development
with MATLAB[®] and Simulink[®]

MANOHAR REDDY M



Using Model-Based Design to develop high quality and reliable Active Safety & Automated Driving Systems

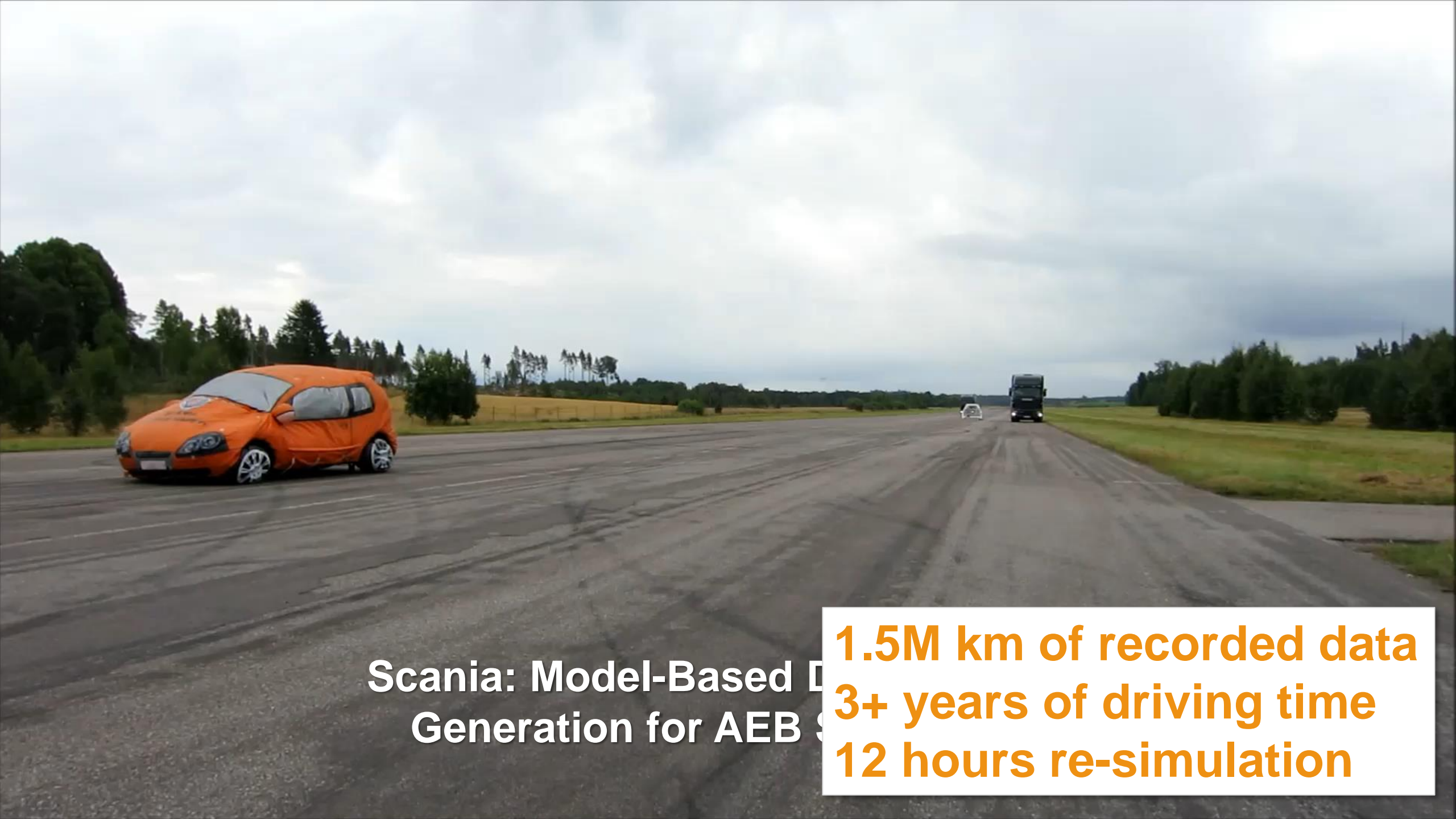
Jonny Andersson

Senior Engineer
Scania

Thorsten Gerke

Automotive Industry Manager EMEA
MathWorks





Scania: Model-Based Data
Generation for AEB

1.5M km of recorded data
3+ years of driving time
12 hours re-simulation

Voyage Develops Longitudinal Controls for Self-Driving Taxis

Challenge

Develop a controller that enables a self-driving car to maintain a target velocity and keep a safe distance from obstacles.

Solution

Use Simulink[®] to design a longitudinal model predictive controller. Tune parameters based on experimental data imported into MATLAB[®]. Deploy the controller as an ROS node using Robotics System Toolbox[™]. Generate source code with Simulink Coder[™], and package it as a Docker container.

Results

- Development speed tripled
- Easy integration with open-source software
- Simulink algorithms delivered as production software

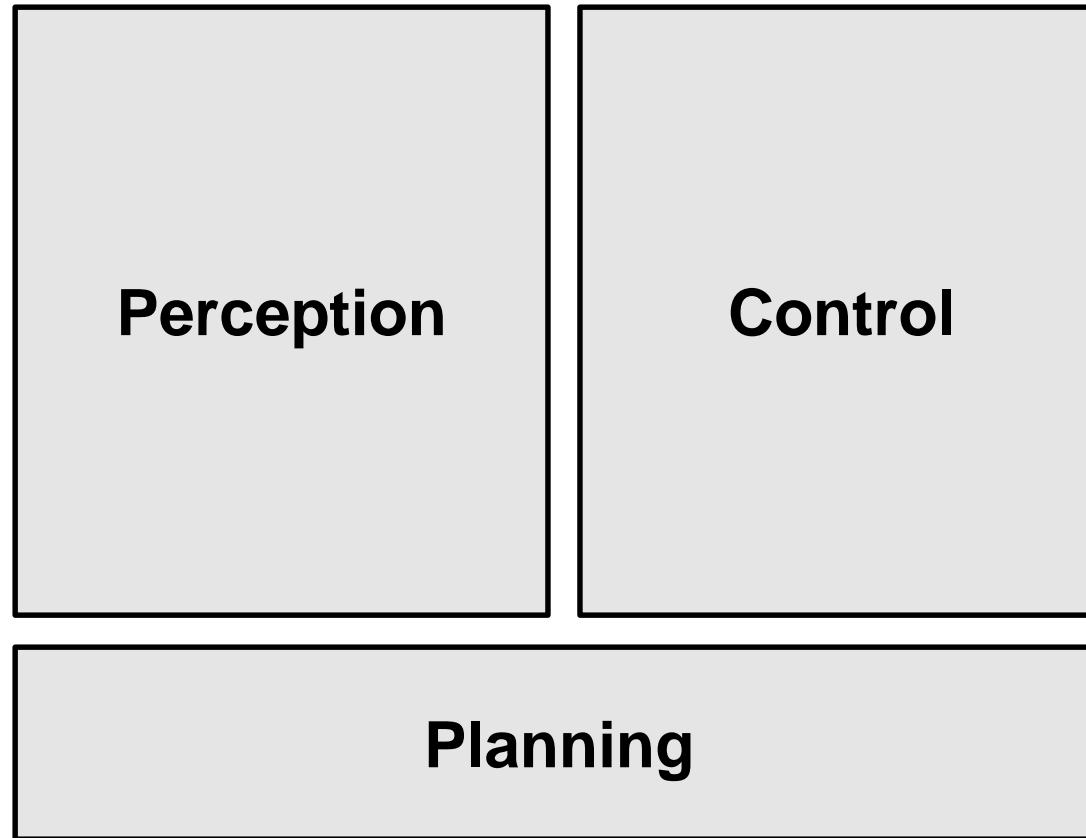


Voyage's self-driving car in San Jose, California.

"We were searching for a prototyping solution that was fast for development and robust for production. We decided to go with Simulink for controller development and code generation, while using MATLAB to automate development tasks."

- Alan Mond, Voyage

How can you use MATLAB and Simulink to develop automated driving algorithms?



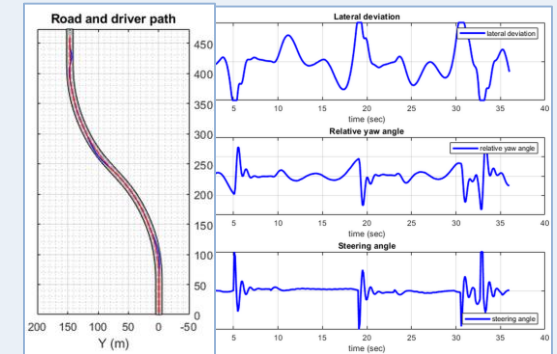
Examples of how you can use MATLAB and Simulink to develop automated driving algorithms

Deep learning



Perception

Sensor models & model predictive control



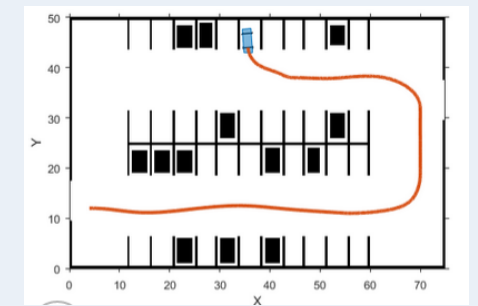
Control

Sensor fusion with live data



Planning

Path planning



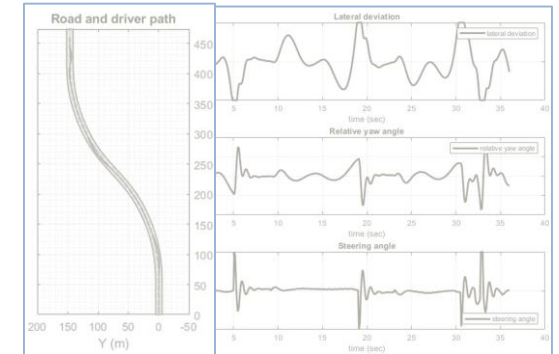
How can you use MATLAB and Simulink to develop perception algorithms?

Deep learning



Perception

Sensor models & model predictive control



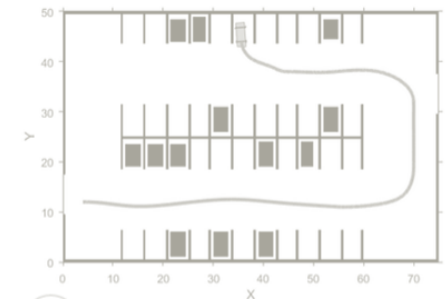
Control

Sensor fusion with live data



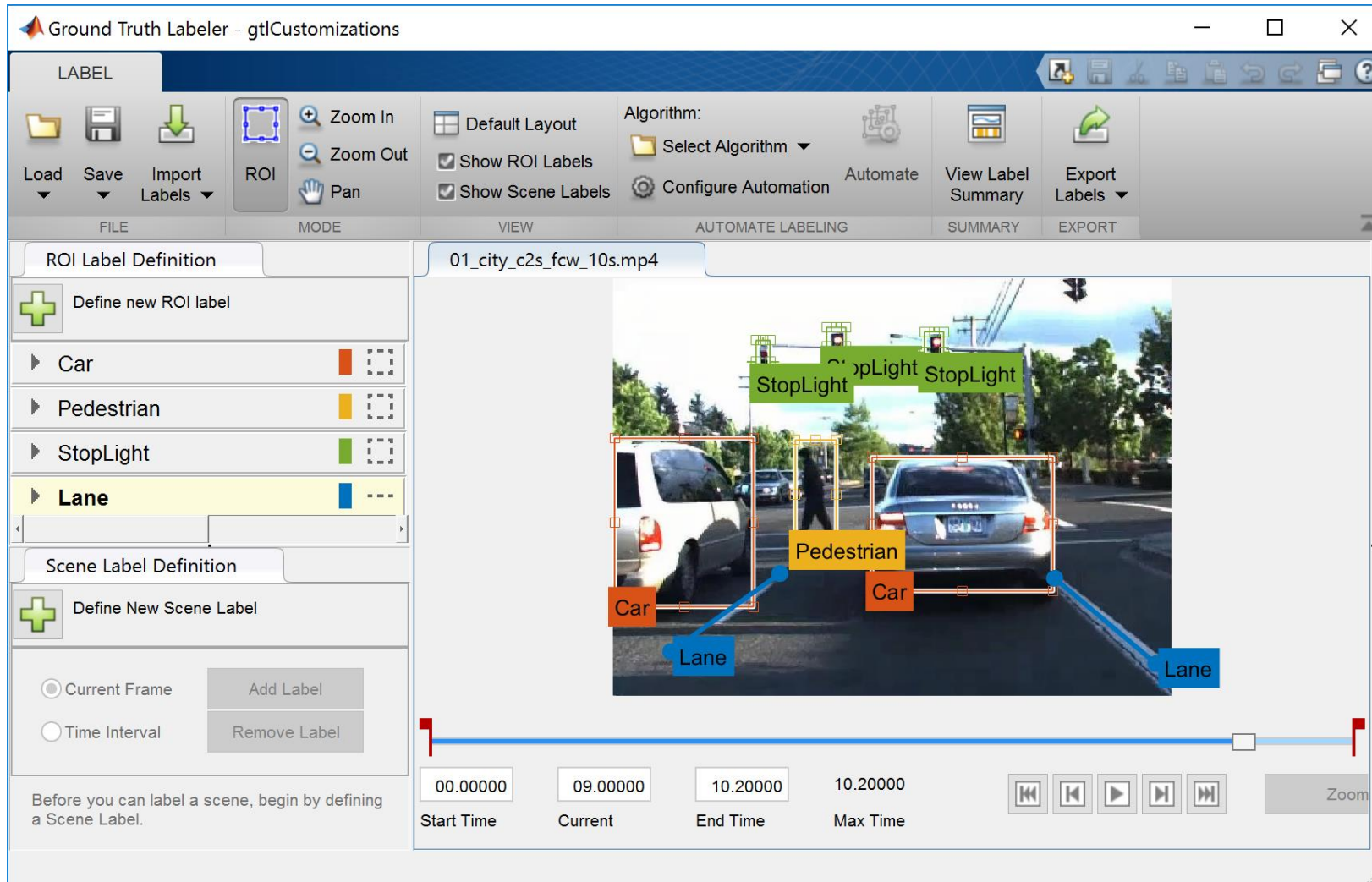
Planning

Path planning



Automated Driving System Toolbox introduced: Ground Truth Labeling App to label video data

R2017a



Automate labeling lanes with Ground Truth Labeler

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

- Top Bar:** Includes window management icons and a toolbar with a green play button labeled "Run automation algorithm", which is highlighted by a large yellow arrow.
- Left Panel (ROI Label Definition):** Contains a list of labels: "laneMarker" (green), "Road" (blue), and "Sky" (orange). Below this is the "Scene Label Definition" section with options for "Current Frame" and "Time Interval", and a list of scene labels: "Sunny" (green) and "Cloudy" (red).
- Center:** A video player showing a first-person view from a vehicle on a road. The video title is "caltech_cordova1.avi".
- Right Panel (Auto Lane Detection):** Contains instructions: "Load a MonoCamera configuration object from the workspace using the settings panel", "Specify additional parameters in the settings panel", "Run the algorithm", and "Manually inspect and modify results if needed".
- Bottom:** A timeline control with buttons for "Start Time", "Current", "End Time", and "Max Time", along with playback controls and a "Zoom Out Time Interval" button.

Automated Driving System Toolbox introduced examples to: Accelerate the process of Ground Truth Labeling

R2017a

Define Ground Truth Data for Video or Image Sequences

R2017a

Automate Ground Truth Labeling of Lane Boundaries

R2017a

Connect Lidar Display to Ground Truth Labeler

- **Label detections** with Ground Truth Labeler App
- **Add your own automation algorithm** to Ground Truth Labeler App
- **Extend connectivity** of Ground Truth Labeler App to visualize lidar data

Specify attributes and sublabels in Ground Truth Labeler App

R2018a

ROI Label Definition

Label Sublabel Attribute

▶ cyclist	■	□
▶ bicycle	■	□
▶ vehicle	■	□

Ground Truth Labeler

LABEL

Load Save Import Labels Label Zoom In Zoom Out Show ROI Labels Show Scene Labels Pan

Algorithm: Select Algorithm Automate View Label Summary Export Labels

FILE MODE VIEW AUTOMATE LABELING SUMMARY EXPORT

vippedtracking.mp4

ROI Label Definition

Label Sublabel Attribute

▶ cyclist

▶ bicycle

▶ vehicle


Scene Label Definition

Define new scene label

Current Frame Add Label

Time Interval Remove Label

To label a scene, you must first define a



Attributes and Sublabels

Attributes

Attributes for cyclist:

bikeType bicycle

action inMotion



Automate labeling pixels with Ground Truth Labeler

The screenshot displays the Ground Truth Labeler application window. The main interface is divided into several sections:

- Top Toolbar:** Includes buttons for 'Label', 'Zoom In', 'Zoom Out', 'Pan', 'Settings', 'Run', 'Stop', 'Undo Run', 'Accept', and 'Cancel'. A yellow arrow points to the 'Run' button with the text 'Automate pixel labeling'.
- Left Panel (ROI Label Definition):** Lists labels such as 'laneMarker', 'Road', and 'Sky'. The 'Road' label is currently selected and highlighted in yellow.
- Left Panel (Scene Label Definition):** Includes options for 'Current Frame' and 'Time Interval', and lists scene labels like 'Sunny' and 'Cloudy'.
- Center View:** A video frame from 'caltech_cordova1.avi' showing a road scene with the road surface automatically labeled in blue.
- Bottom Panel:** A timeline with 'Start Time' (01.33334), 'Current' (01.33334), 'End Time' (02.47726), and 'Max Time' (08.33334). It also features playback controls and a 'Zoom Out Time Interval' button.
- Right Panel:** Contains instructional text:
 - Review and Modify:** Review automated labels over the interval using playback controls. Modify/delete/add ROIs that were not satisfactorily automated at this stage. If the results are satisfactory, click Accept to accept the automated labels.
 - Accept/Cancel:** If results of automation are satisfactory, click Accept to accept all automated labels and return to manual labeling. If results of automation are not satisfactory, click Cancel to return to manual labeling without saving automated labels.

Learn more about developing deep learning perception algorithms with these examples

R2017b

Semantic Segmentation Using Deep Learning

- Train **free space detection network** using deep learning
Computer Vision System Toolbox™

R2018a

Automate Ground Truth Labeling for Semantic Segmentation

- Add **semantic segmentation automation** algorithm to Ground Truth Labeler App
Automated Driving System Toolbox™

R2018a

Code Generation for Semantic Segmentation Network

- Generate **CUDA® code** to execute directed acyclic graph network on an NVIDIA GPU
GPU Coder™

Robotics and Autonomous Systems	
14:30	Demystifying Deep Learning <i>Dr. Amod Anandkumar, MathWorks</i>

Robotics and Autonomous Systems	
15:30	Deploying Deep Neural Networks to Embedded GPUs and CPUs <i>Rishu Gupta, Ph.D, MathWorks</i>

Free Space Detection Using Semantic Segmentation



How can you use MATLAB and Simulink to develop perception algorithms?

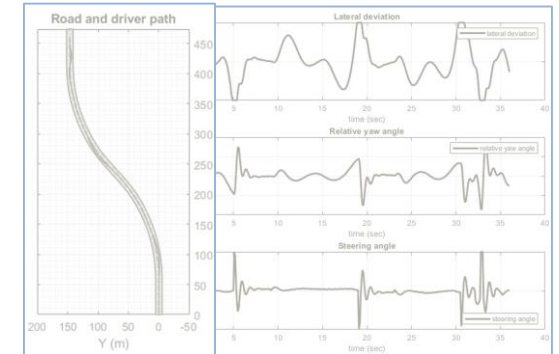
Deep learning



Perception

Control

Sensor models & model predictive control

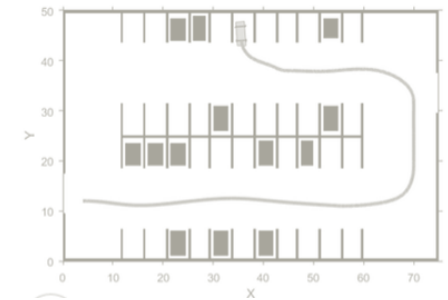


Sensor fusion with live data

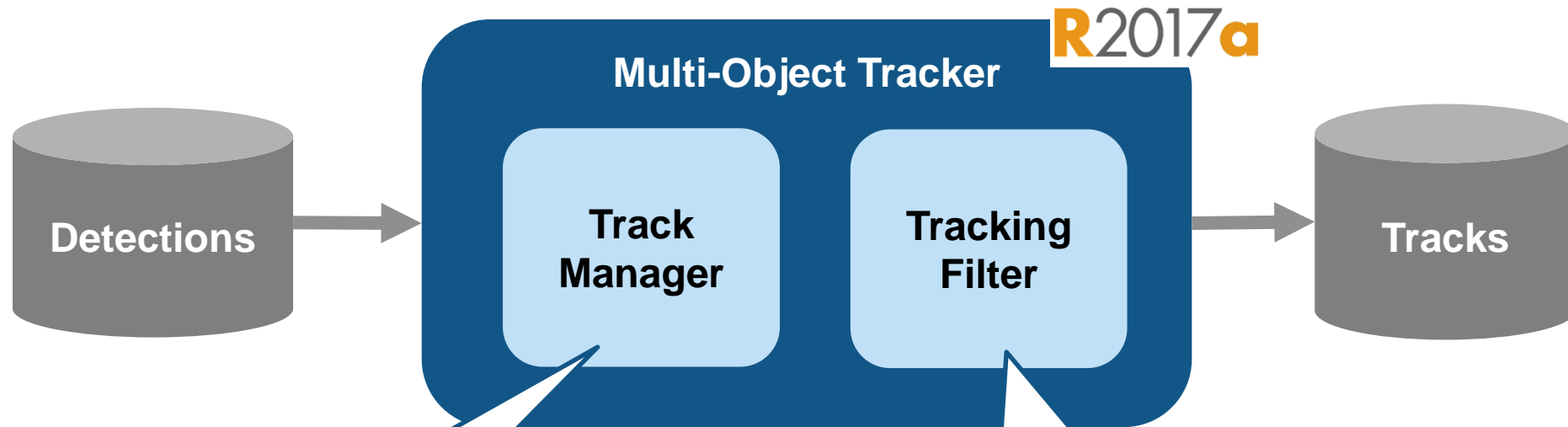


Planning

Path planning



Automated Driving System Toolbox introduced: Multi-object tracker to develop sensor fusion algorithms



- 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

Videos and Webinars

Some common questions from automated driving engineers

How can I visualize vehicle data?

19:27

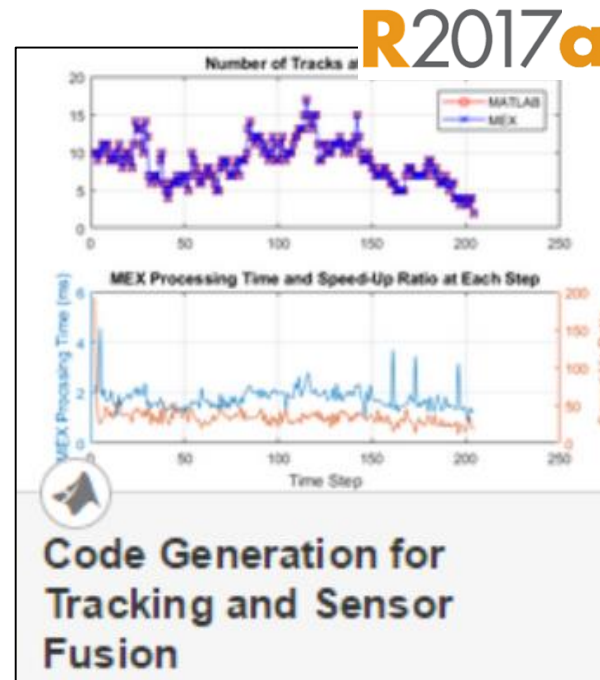
How can I fuse multiple detections?

Introduction to Automated Driving System Toolbox

Automated Driving System Toolbox introduced examples to: Develop sensor fusion algorithms with recorded data

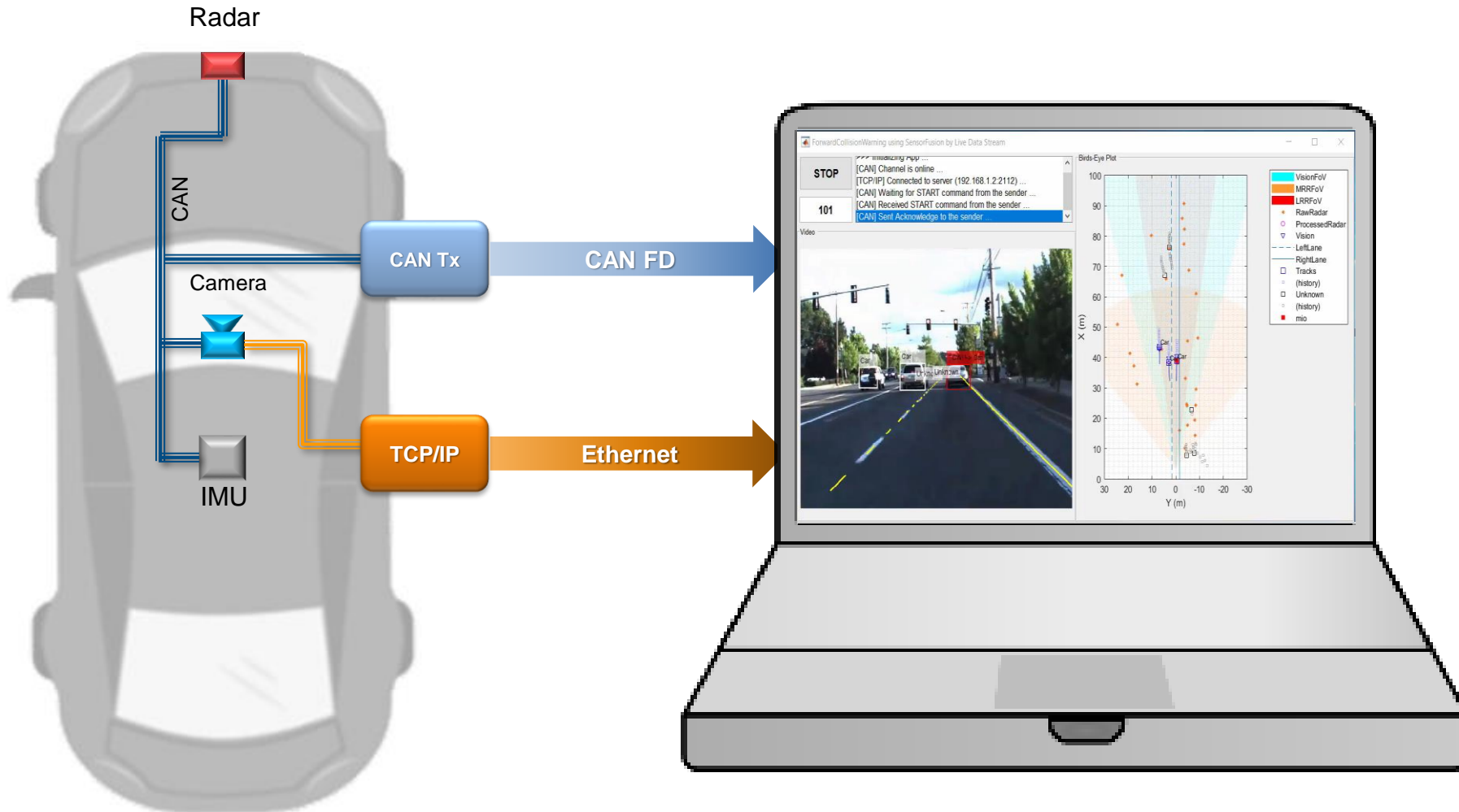


- **Design** multi-object tracker based on logged vehicle data

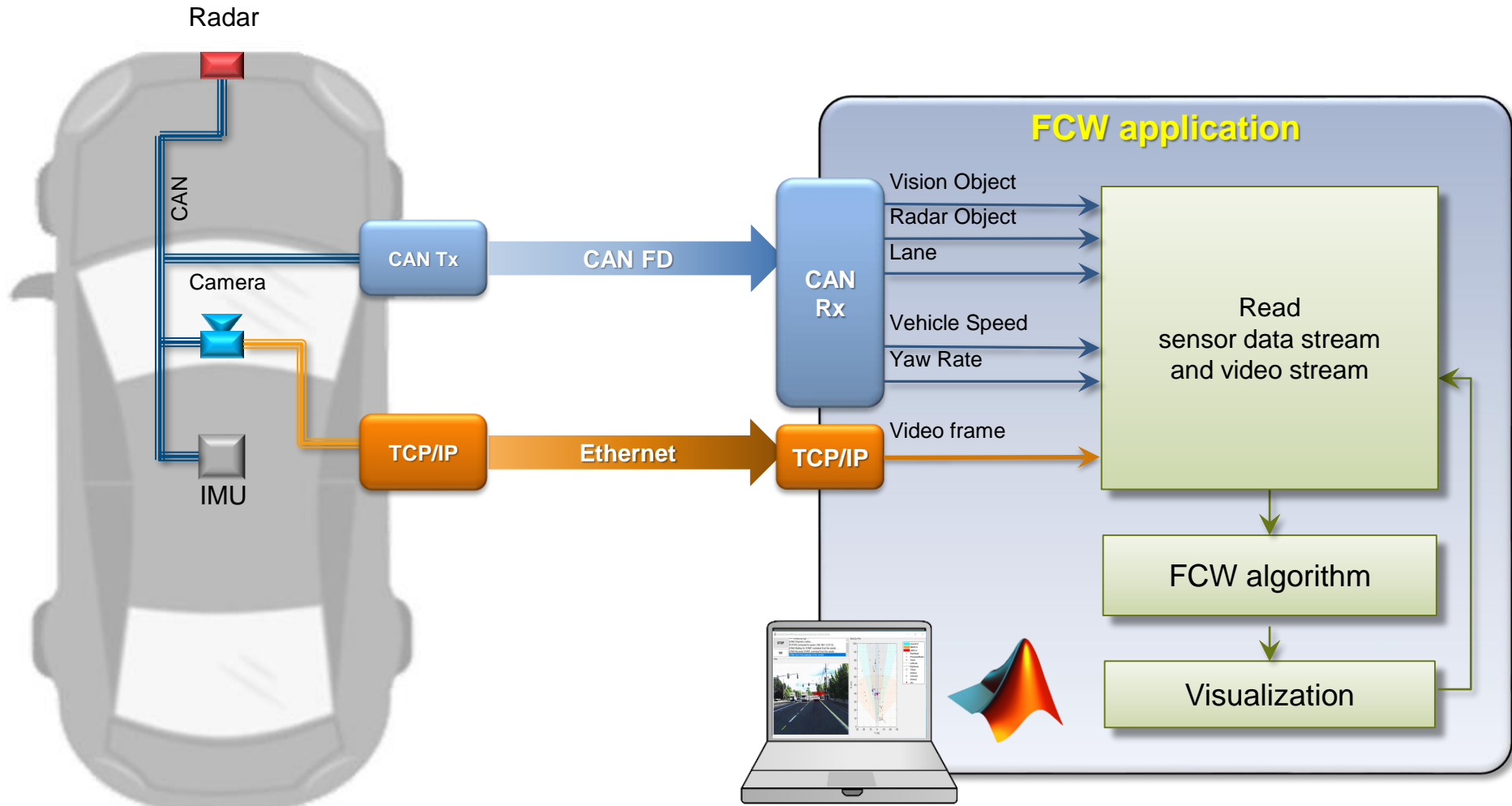


- **Generate C/C++** code from algorithm which includes a multi-object tracker

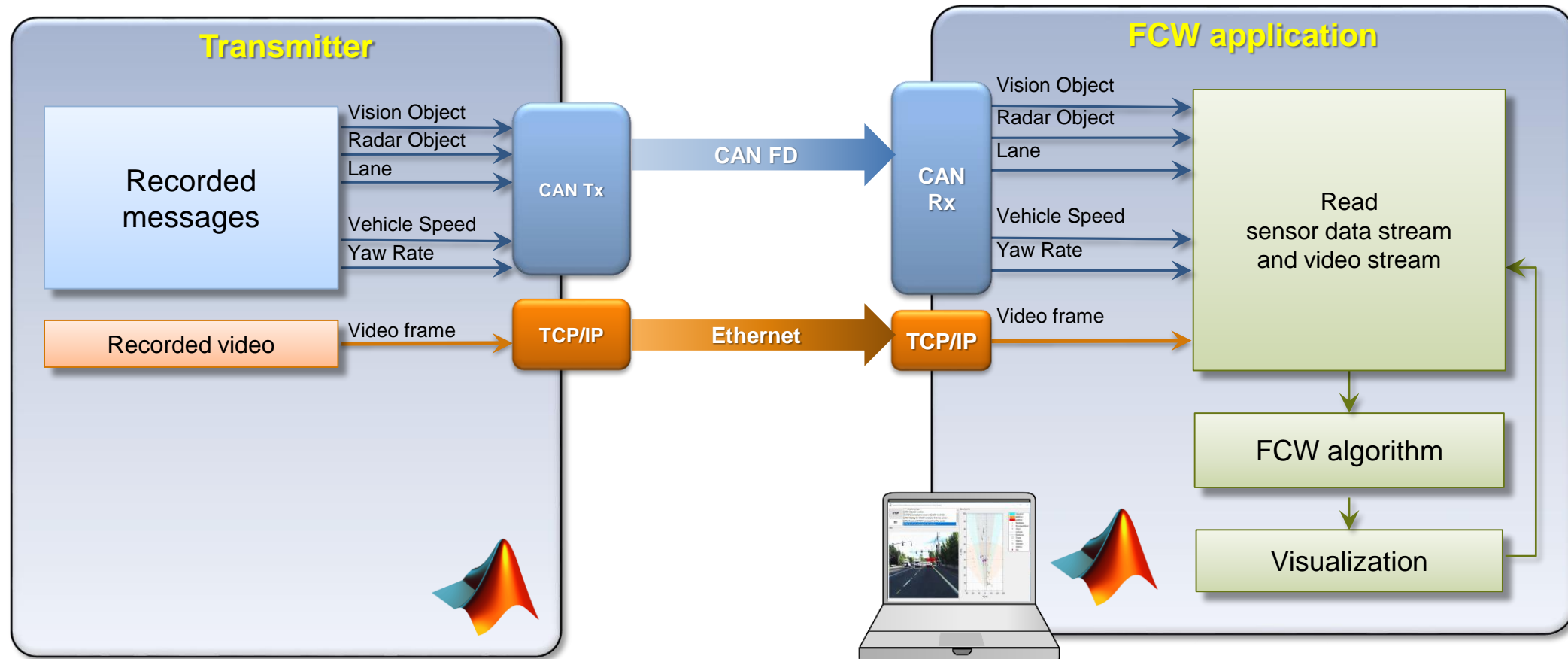
How can I test my sensor fusion algorithm with live data?



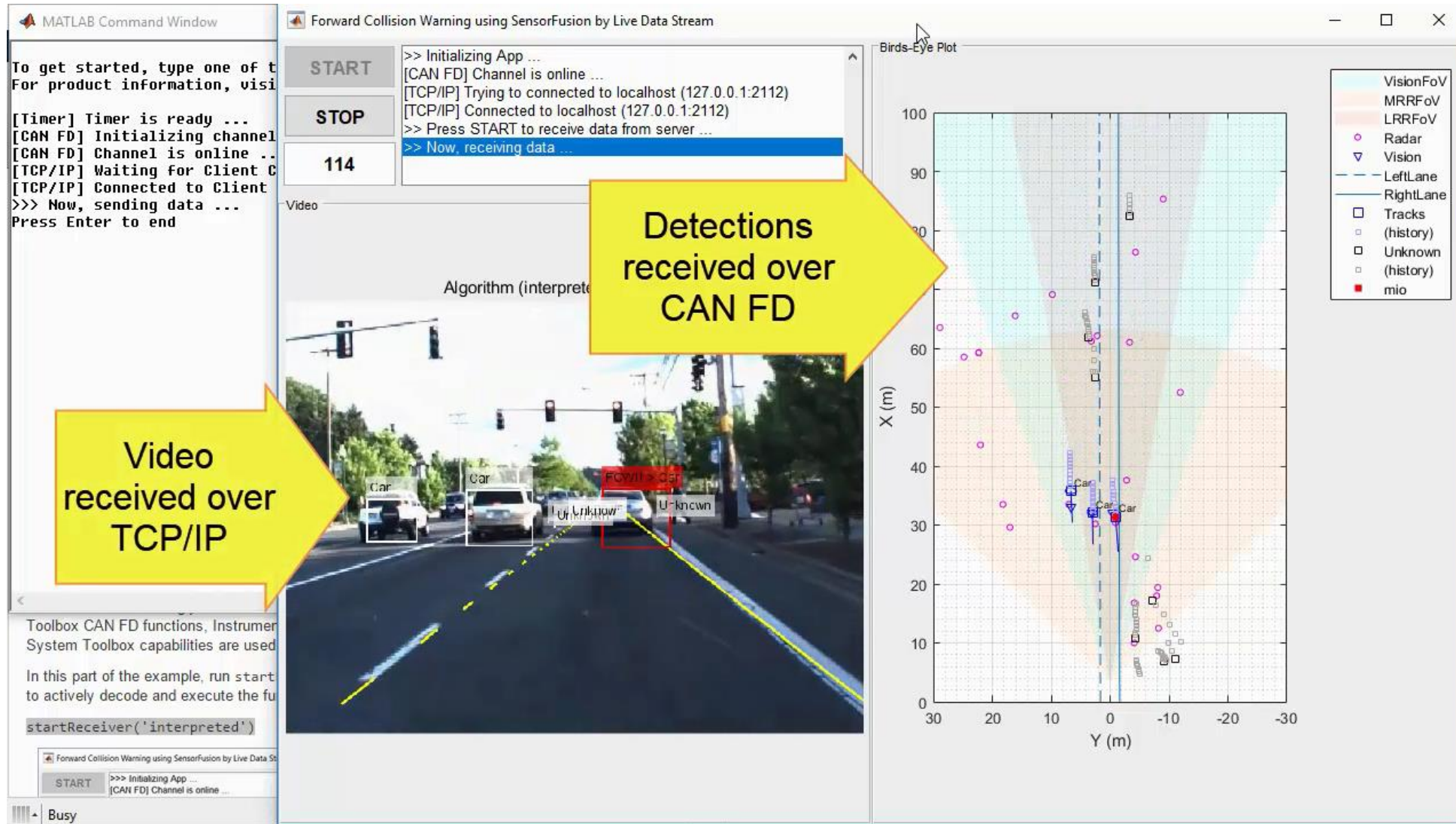
Test forward collision warning algorithm with live data from vehicle



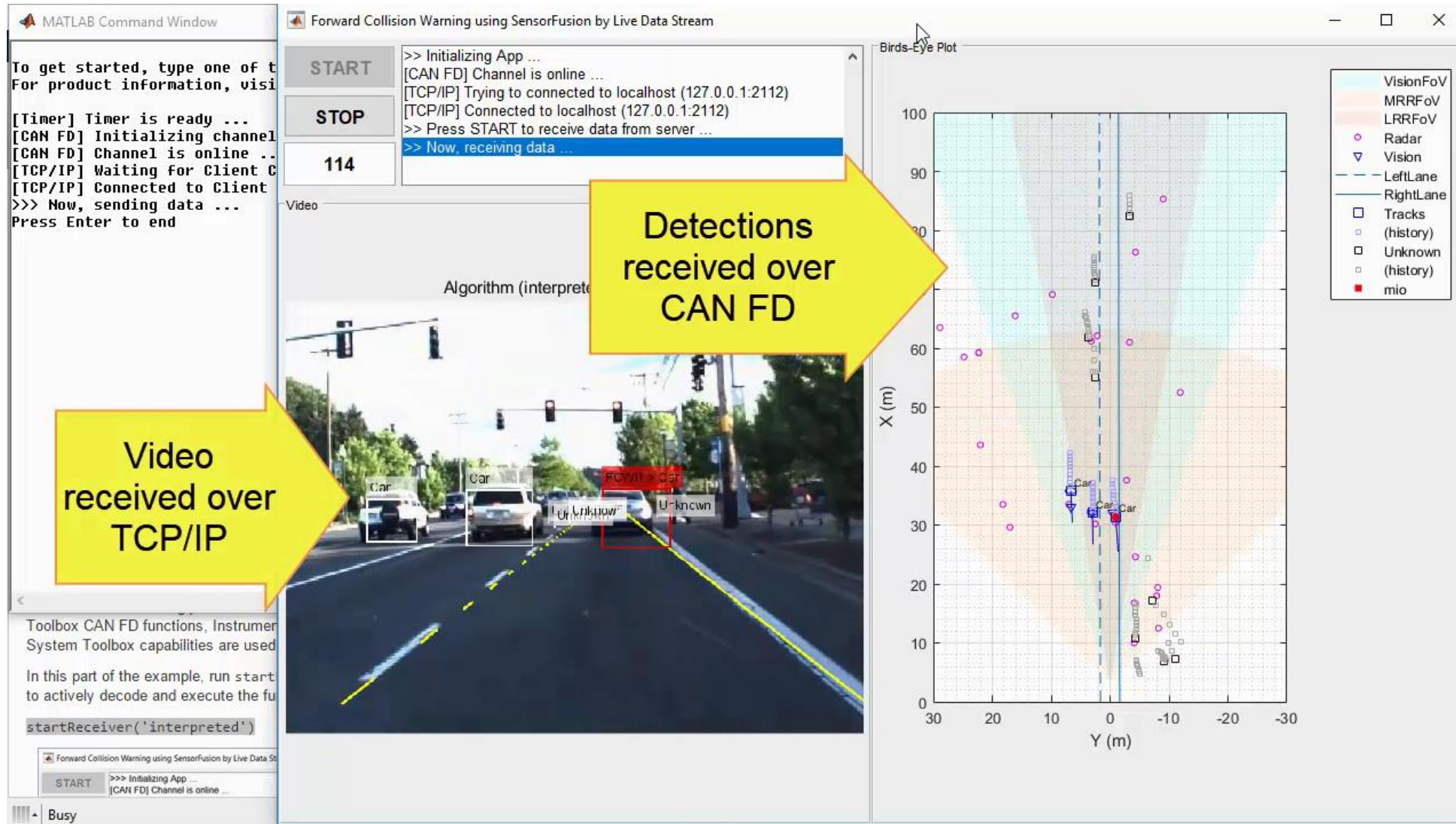
Test forward collision warning algorithm with live data from “surrogate” vehicle



Send live CAN FD and TCP/IP data



Receive live CAN FD and TCP/IP data



Generate C/C++ code for algorithm

MATLAB Coder Report Viewer - C:\MATLABExamples\VNT\codegen\lib\trackingForFCW_kernel\html\report.mldata

REPORT

Back Forward Find Trace Code Edit In MATLAB Package Code

MATLAB SOURCE

Function List Call Tree

- trackingForFCW_kernel.m
 - trackingForFCW_kernel
 - calculateGroundSpeed
 - fcwmeas > 1
 - fcwmeas > 2
 - fcwmeasjac > 1
 - fcwmeasjac > 2
 - findMostImportantObject
 - findNonClutterRadarObjec
 - initConstantAccelerationFi
 - processDetections

GENERATED CODE

- trackingEKF.h
- trackingForFCW_kernel.c
- trackingForFCW_kernel.h
- trackingForFCW_kernel_e
- trackingForFCW_kernel_e
- trackingForFCW_kernel_e
- trackingForFCW_kernel_e
- trackingForFCW_kernel_ir
- trackingForFCW_kernel_ir
- trackingForFCW_kernel_rt
- trackingForFCW_kernel_rt
- trackingForFCW_kernel_te

```

1565 *      const struct5_T *laneReports
1566 *      struct7_T *egoLane
1567 *      double time
1568 *      const double positionSelector[12]
1569 *      const double velocitySelector[12]
1570 *      emxArray_struct8_T *confirmedTracks
1571 *      double *numTracks
1572 *      struct10_T *mostImportantObject
1573 * Return Type : void
1574 */
1575 void trackingForFCW_kernel(const struct0_T *visionObjects, const struct2_T
1576 *radarObjects, const struct4_T *inertialMeasurementUnit, const struct5_T
1577 *laneReports, struct7_T *egoLane, double time, const double positionSelector
1578 [12], const double velocitySelector[12], emxArray_struct8_T *confirmedTracks,
1579 double *numTracks, struct10_T *mostImportantObject)
1580 {
1581     emxArray_objectDetection *detections;
1582     emxInit_objectDetection(&detections, 2);
1583
1584

```

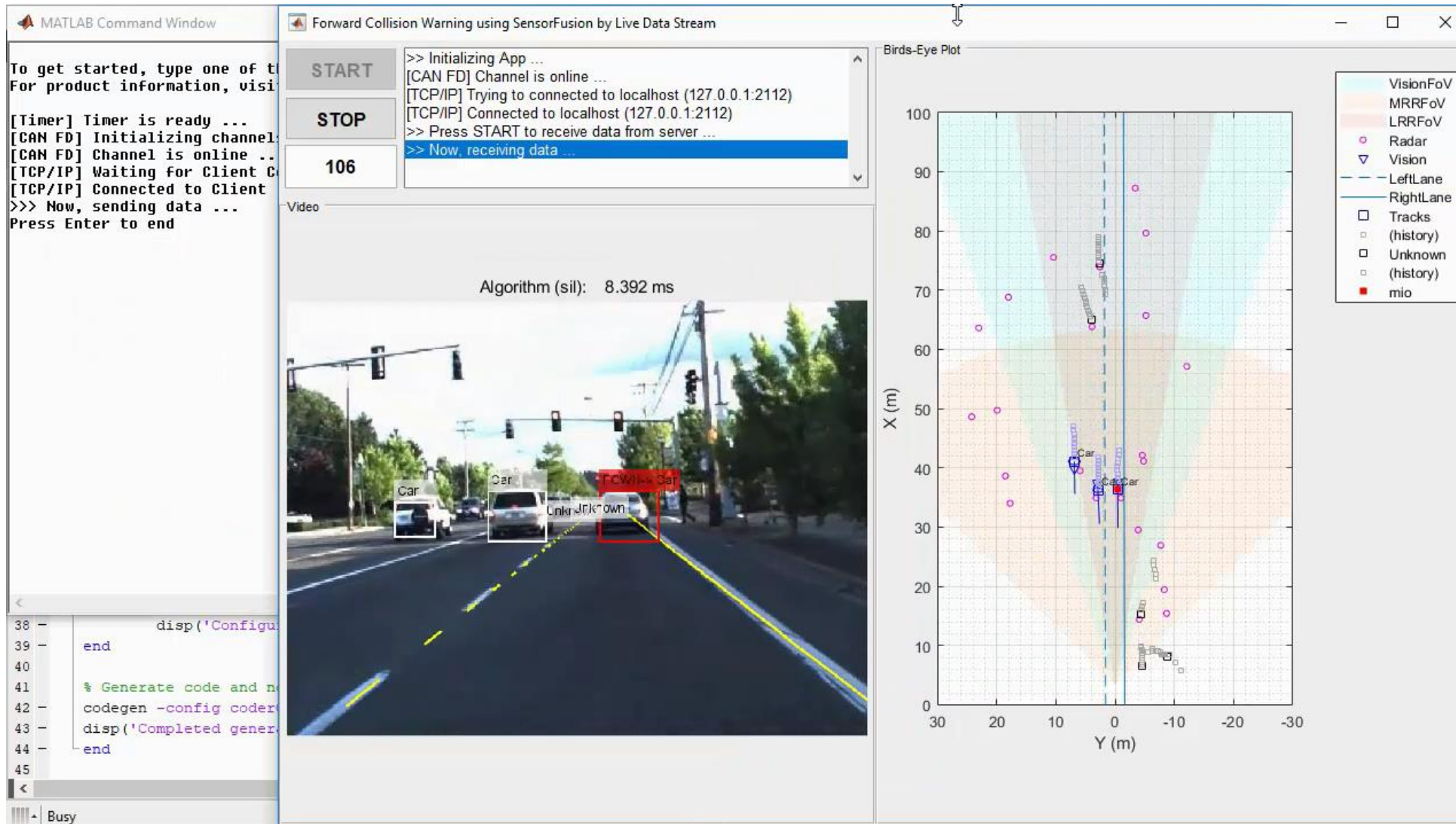
Generated C function

SUMMARY ALL MESSAGES (1) BUILD LOGS CODE INSIGHTS (0) VARIABLES

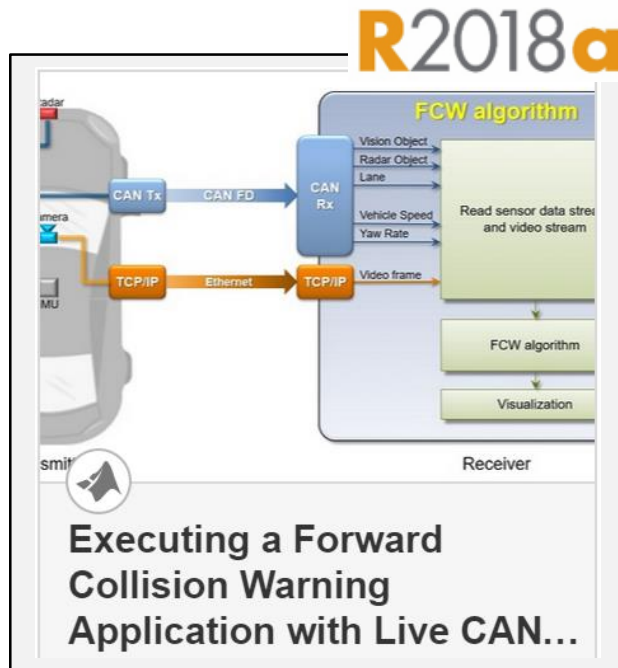
Code generation successful

Generated on: 17-Mar-2018 19:07:16
 Build type: Static Library
 Output file: C:\MATLABExamples\VNT\codegen\lib\trackingForFCW_kernel\trackingForFCW_kernel.lib
 Processor: Generic->MATLAB Host Computer

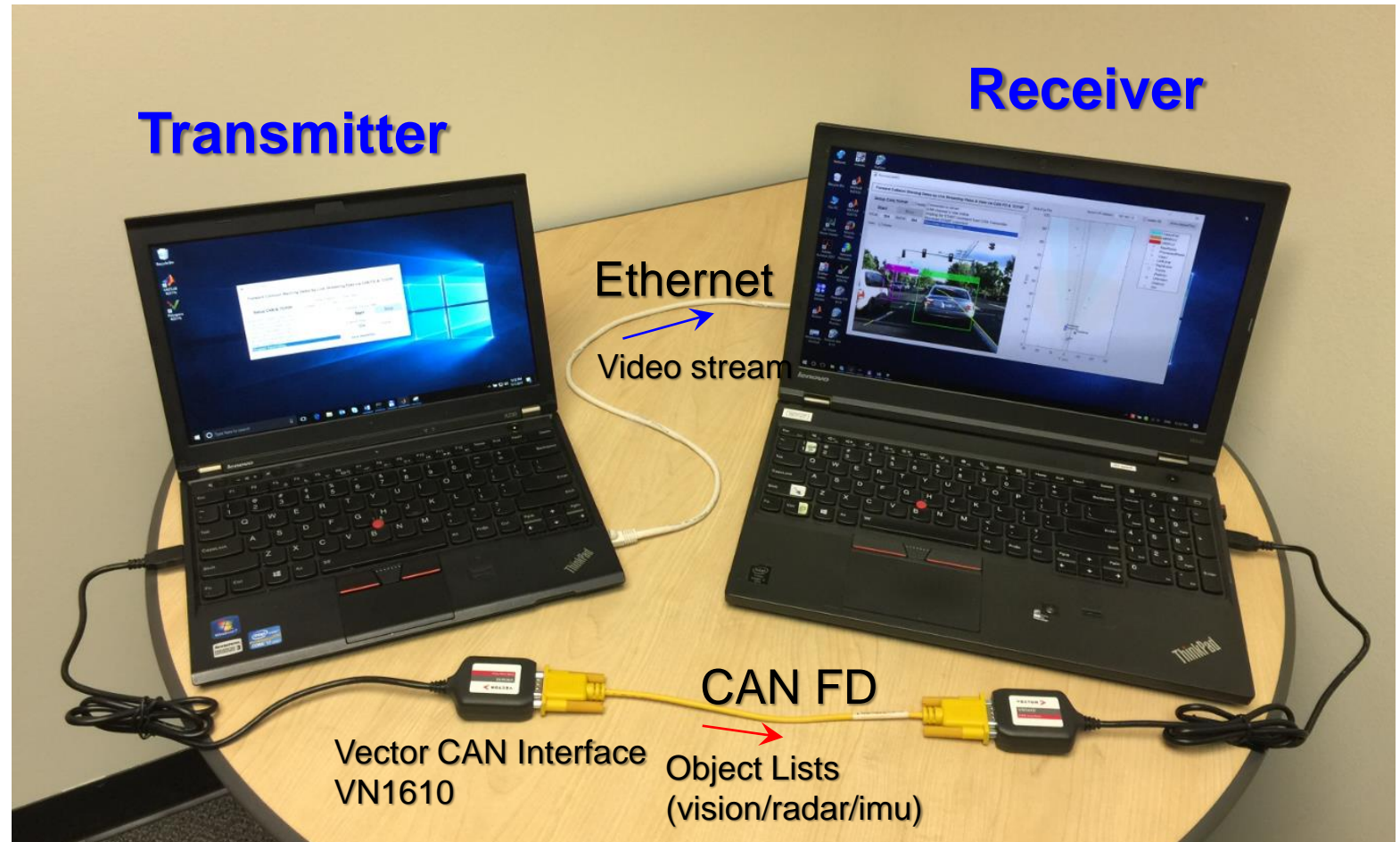
Stream live CAN FD and TCP/IP data into compiled algorithm code



Learn about developing sensor fusion algorithms with live data using this example



- Stream CAN FD data to prototype algorithms on your laptop
Vehicle Network Toolbox™



How can you use MATLAB and Simulink to develop control algorithms?

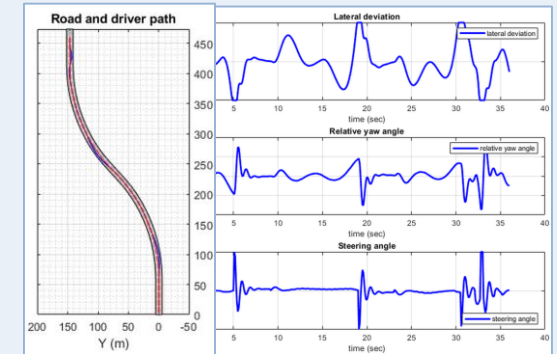
Deep learning



Perception

Control

Sensor models & model predictive control

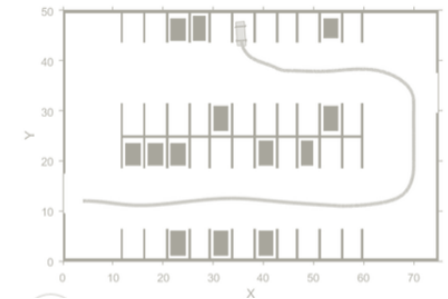


Sensor fusion with live data

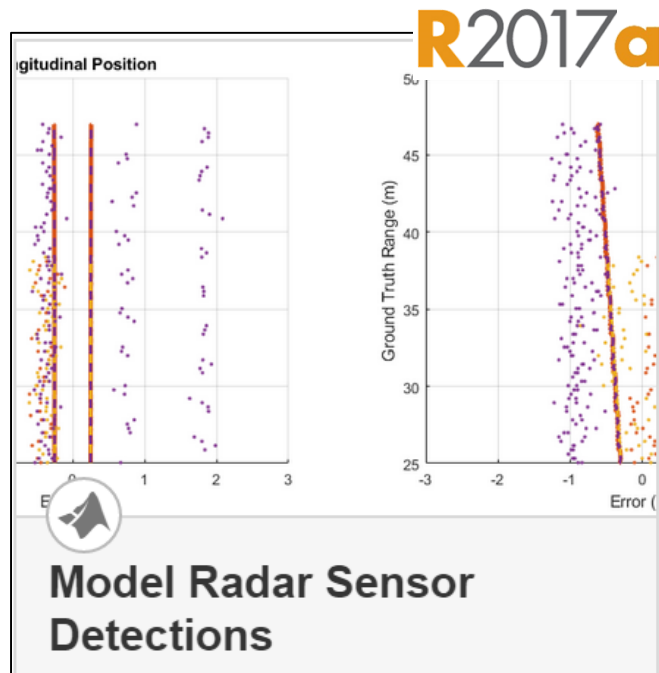


Planning

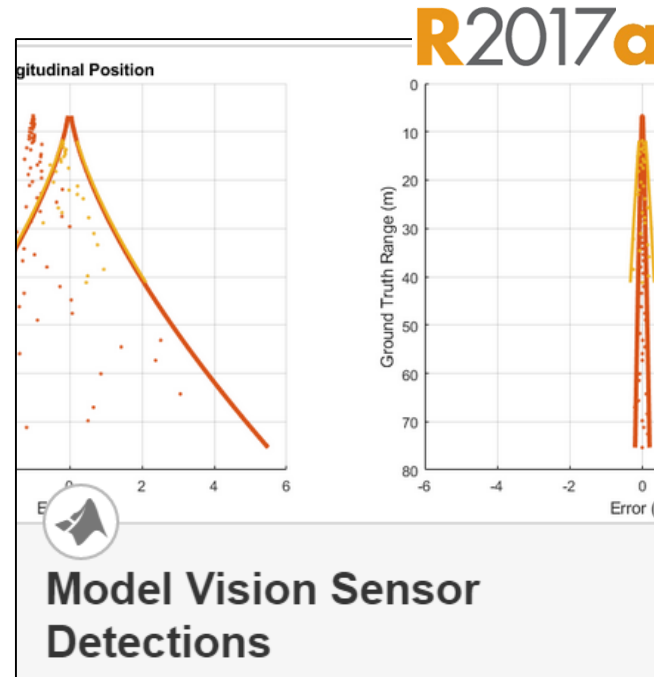
Path planning



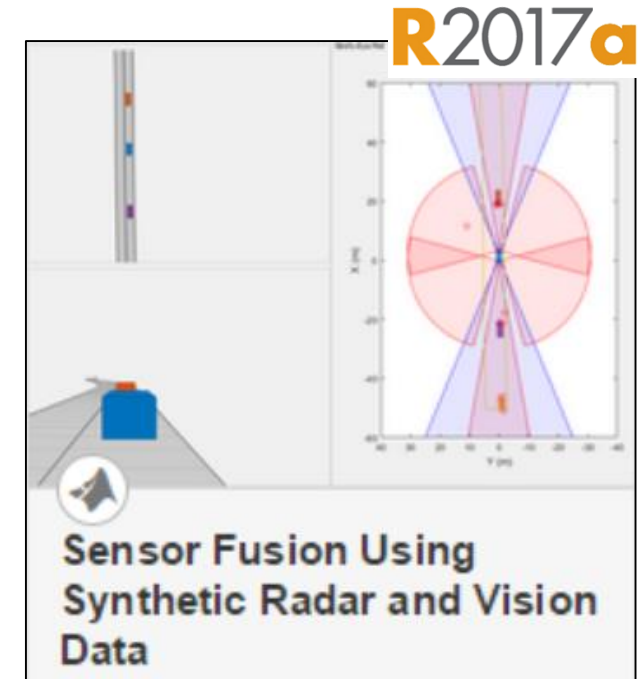
Automated Driving System Toolbox introduced examples to: Synthesize detections to test sensor fusion algorithms



- **Synthesize radar** detections with probabilistic impairments



- **Synthesize vision** detections with probabilistic impairments



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

Automated Driving System Toolbox introduced: Radar and vision detections for closed loop simulation

Release	Fuse	Synthesize	Visualize
R2017a	<code>multiObjectTracker</code>	<code>radarDetectionGenerator</code> <code>visionDetectionGenerator</code>	<code>birdsEyePlot</code>
R2017b	<p>A block diagram for the Multi-Object Tracker. It has two input ports on the left: 'Detections' and 'Prediction Time'. Inside the block, the text 'Multi Object Tracker' is centered. It has one output port on the right labeled 'Confirmed Tracks'.</p>	<p>Two block diagrams for detection generators. The top one is the Radar Detection Generator, with an input 'Actors', an internal label 'Radar Detection Generator [Sensor Index: 2]', and an output 'Detections'. The bottom one is the Vision Detection Generator, with an input 'Actors', an internal label 'Vision Detection Generator [Sensor Index: 1]', and an output 'Detections'.</p>	<p>A block diagram for the Bird's-Eye Plot. It has six input ports on the left: 'Actors', 'Vision', 'Radar', 'Tracks', 'MIO', and 'Roads'. Inside the block, the text 'Bird's-Eye Plot' is centered.</p>

Voyage Develops Longitudinal Controls for Self-Driving Taxis

Challenge

Develop a controller that enables a self-driving car to maintain a target velocity and keep a safe distance from obstacles.

Solution

Use Simulink[®] to design a longitudinal model predictive controller. Tune parameters based on experimental data imported into MATLAB[®]. Deploy the controller as an ROS node using Robotics System Toolbox[™]. Generate source code with Simulink Coder[™], and package it as a Docker container.

Results

- Development speed tripled
- Easy integration with open-source software
- Simulink algorithms delivered as production software

[Technical Article](#)



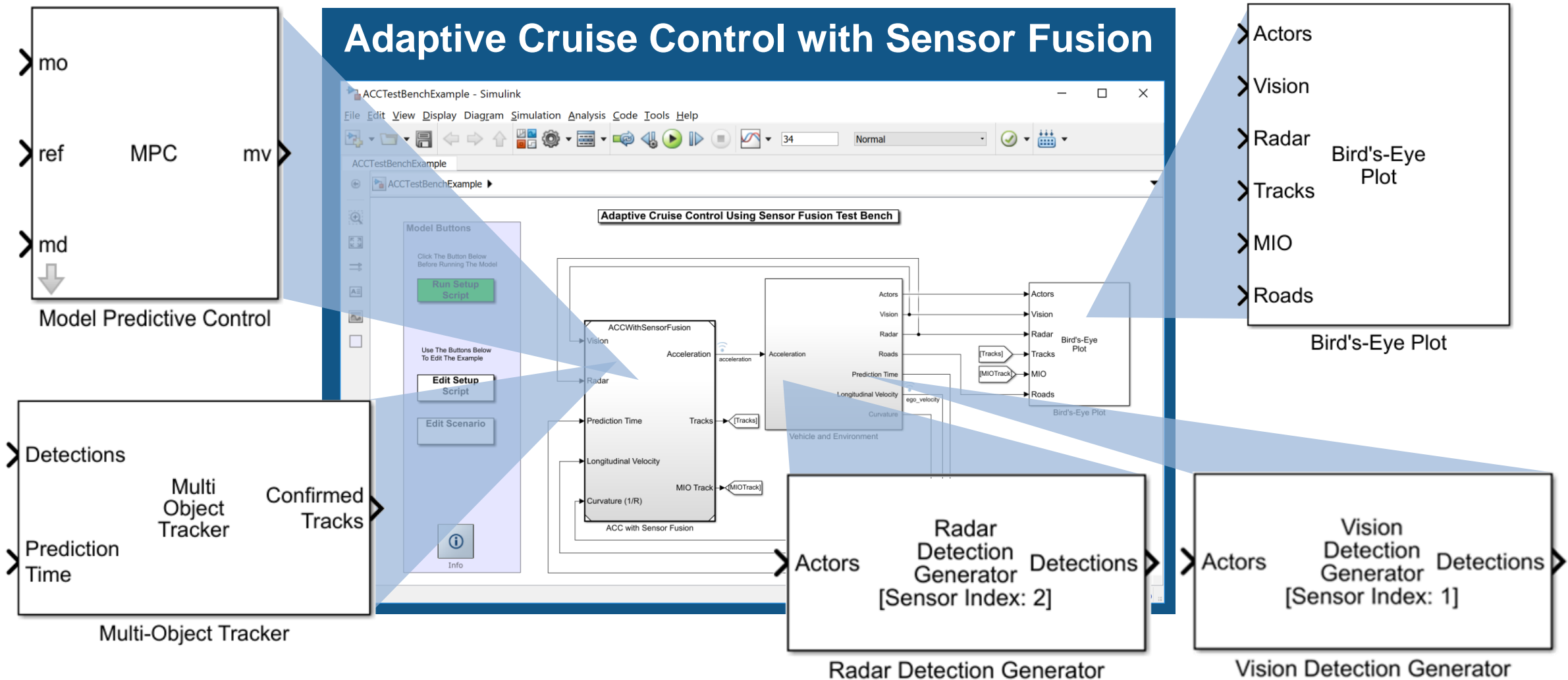
Voyage's self-driving car in San Jose, California.

"We were searching for a prototyping solution that was fast for development and robust for production. We decided to go with Simulink for controller development and code generation, while using MATLAB to automate development tasks."

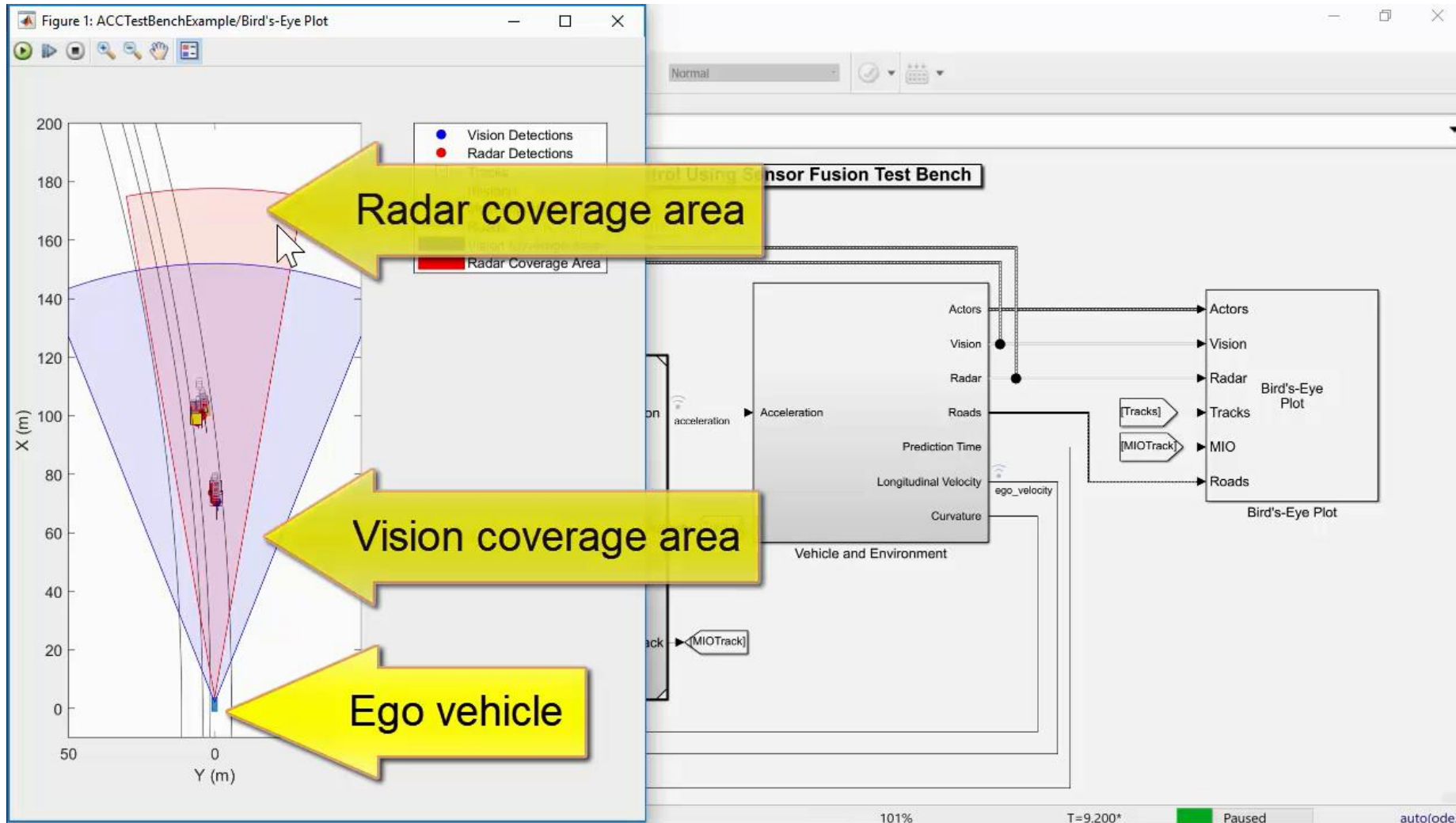
- Alan Mond, Voyage

Simulate closed loop system with radar/vision detections, sensor fusion, and model-predictive control

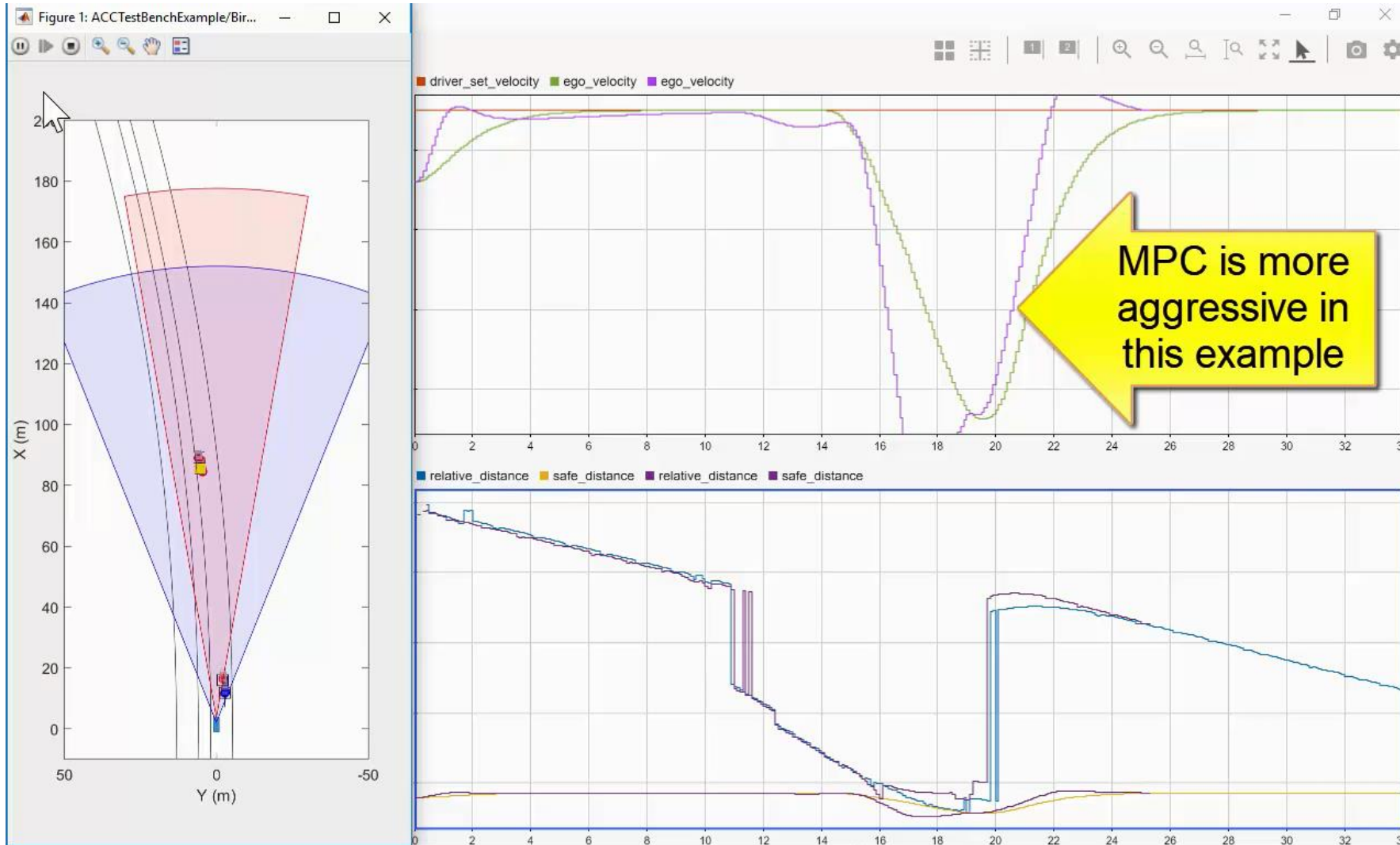
R2017b



Synthesize detections to test sensor fusion and model-predictive controller

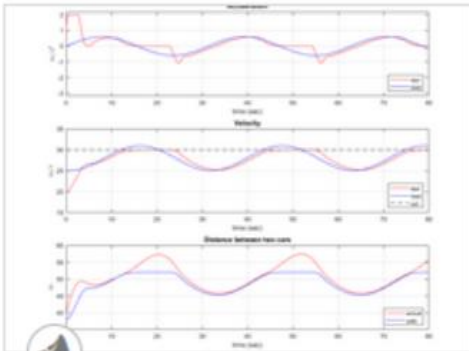


Compare classical and model predictive control algorithms



Automated Driving Applications with Model Predictive Controls

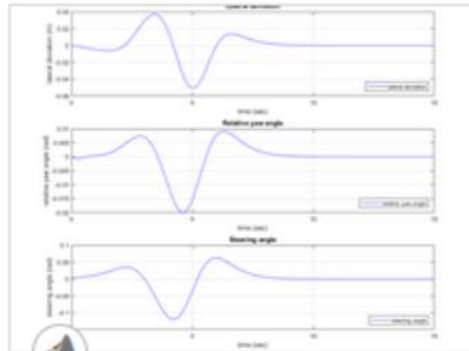
Automated Driving Applications



Adaptive Cruise Control System Using Model Predictive Control

Use the block in Simulink® and demonstrates the control objectives and constraints of this block.

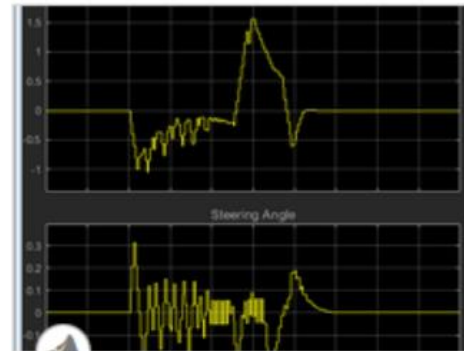
[Open Script](#)



Lane Keeping Assist System Using Model Predictive Control

Use the block in Simulink® and demonstrates the control objectives and constraints of this block.

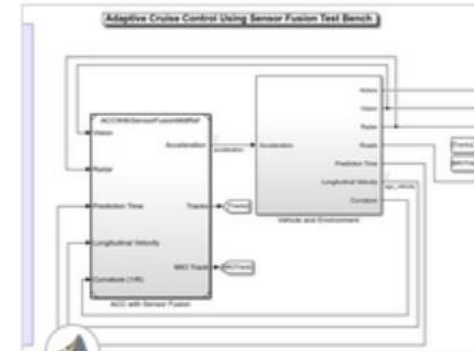
[Open Script](#)



Obstacle Avoidance Using Adaptive Model Predictive Control

Make a vehicle (ego car) follow a reference velocity and avoid obstacles in the lane using adaptive MPC. To do so, you update the

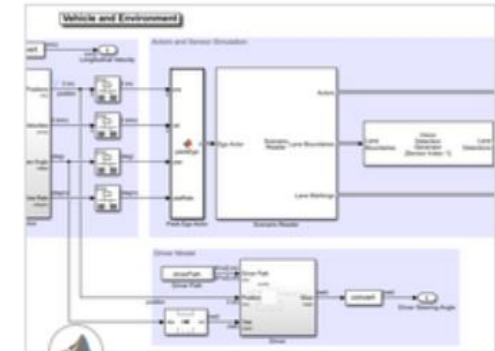
[Open Script](#)



Adaptive Cruise Control with Sensor Fusion

Implement a sensor fusion based automotive adaptive cruise controller for a vehicle traveling on a curved road using sensor fusion.

[Open Model](#)



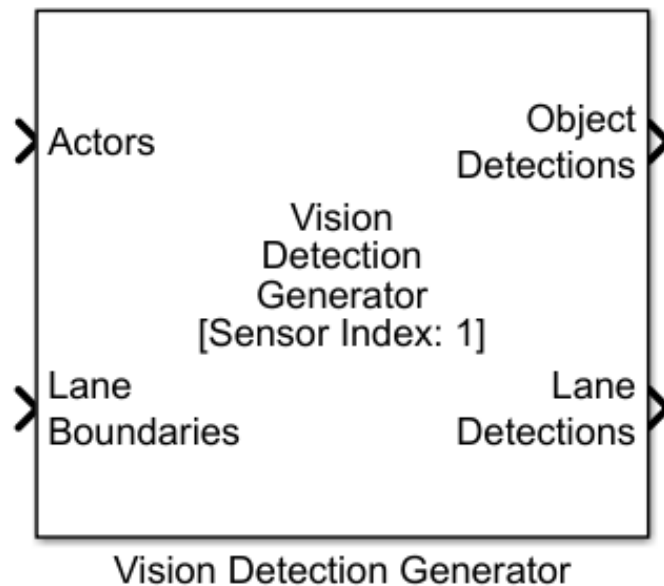
Lane Keeping Assist with Lane Detection

Simulate and generate code for an automotive lane keeping assist (LKA) controller.

[Open Model](#)

Vision Detection Generator models lane detection sensor

R2018a



Block Parameters: Vision Detection Generator

Vision Detection Generator

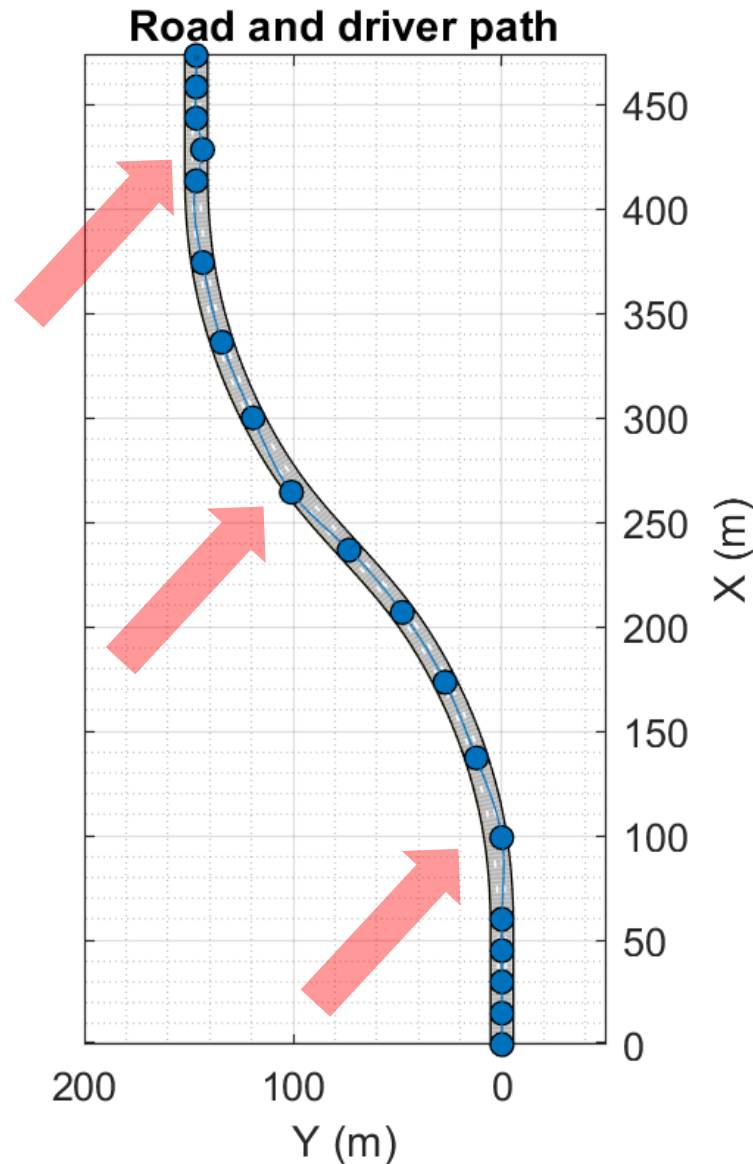
Sensor simulation block used to generate vision detections from simulated actor poses. Detections are generated at intervals of the sensor's update interval. A statistical model generates measurement noise, true detections, and false positives. The random numbers used by the statistical model are controlled by the random number generator settings on the Measurements tab.

[Source code](#)

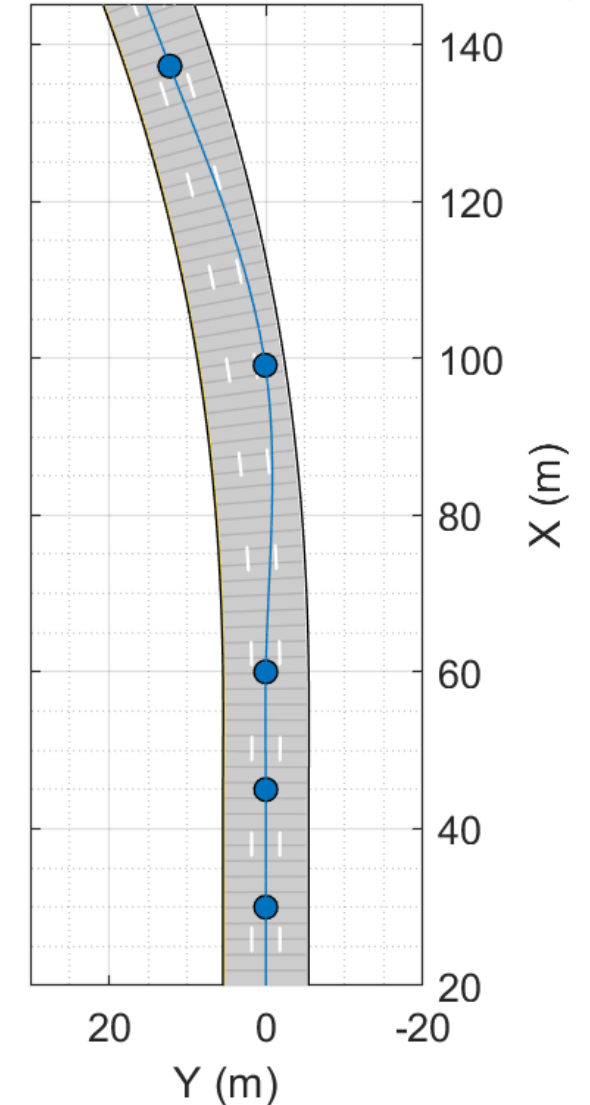
Parameters	Measurements	Actor Profiles	Camera Intrinsic
Sensor Identification			
Unique identifier of sensor:	<input type="text" value="1"/>		
Types of detections generated by sensor:	<ul style="list-style-type: none"> Lanes and objects Objects only Lanes only Lanes with occlusion Lanes and objects 		
Required interval between sensor updates (s):	<input type="text"/>		
Required interval between lane detection updates (s):	<input type="text"/>		
Sensor Extrinsic			
Sensor's (x,y) position (m):	<input type="text" value="[1.9, 0]"/>		
Sensor's height (m):	<input type="text" value="1.1"/>		
Yaw angle of sensor mounted on ego vehicle (deg):	<input type="text" value="0"/>		
Pitch angle of sensor mounted on ego vehicle (deg):	<input type="text" value="1"/>		
Roll angle of sensor mounted on ego vehicle (deg):	<input type="text" value="0"/>		

Create highway double curve with drivingScenario

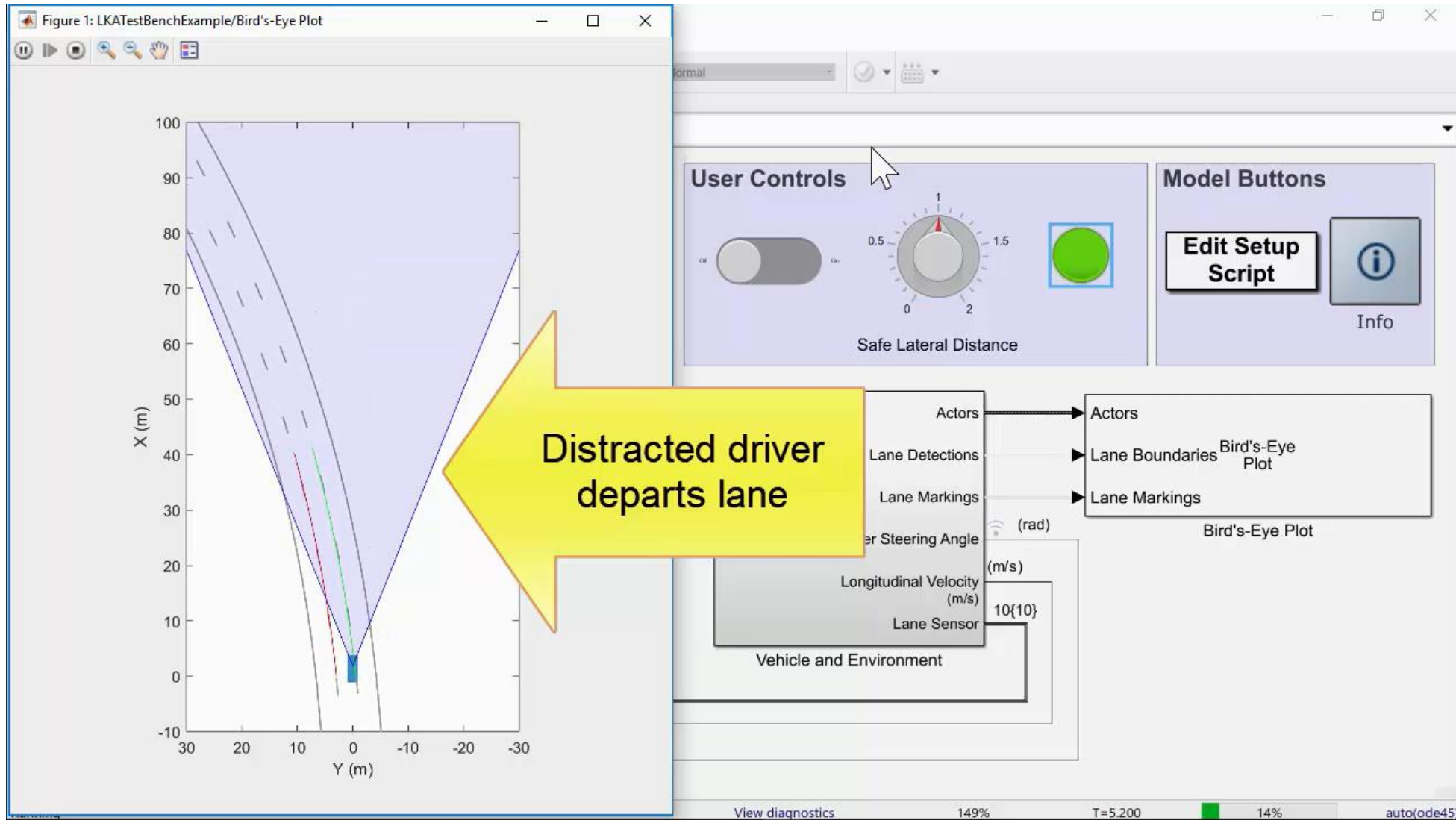
- Driver waypoints simulate distraction at curvature changes



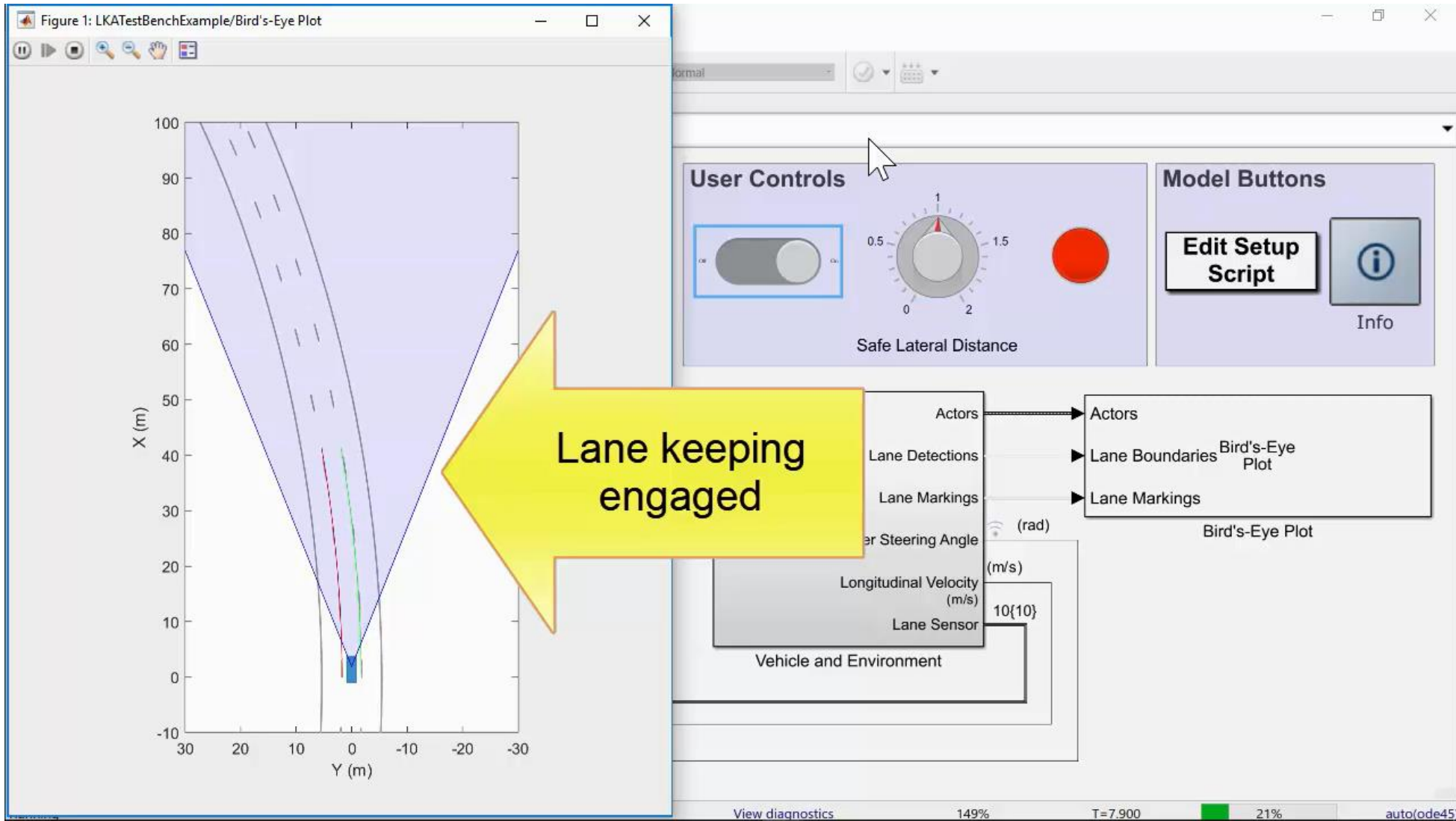
Driver distracted at curvature change



Simulate distracted driver

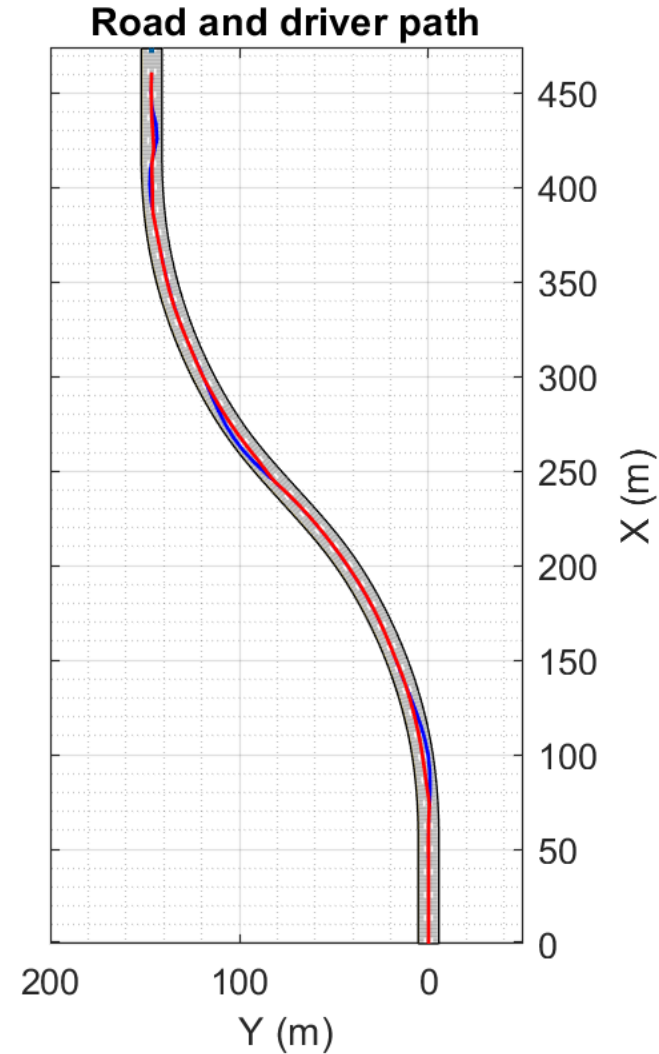


Simulate lane keep assist at distraction events

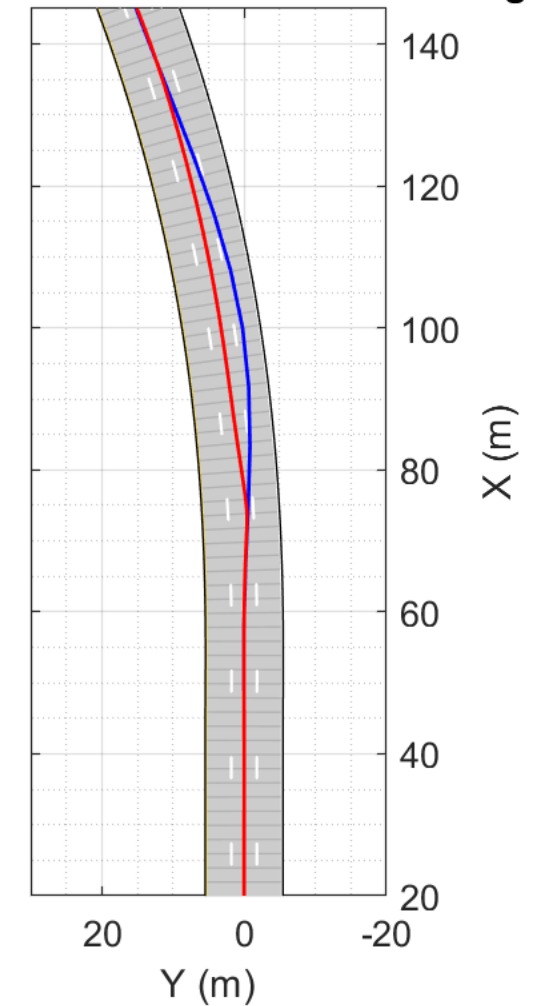


Compare distracted and assisted results

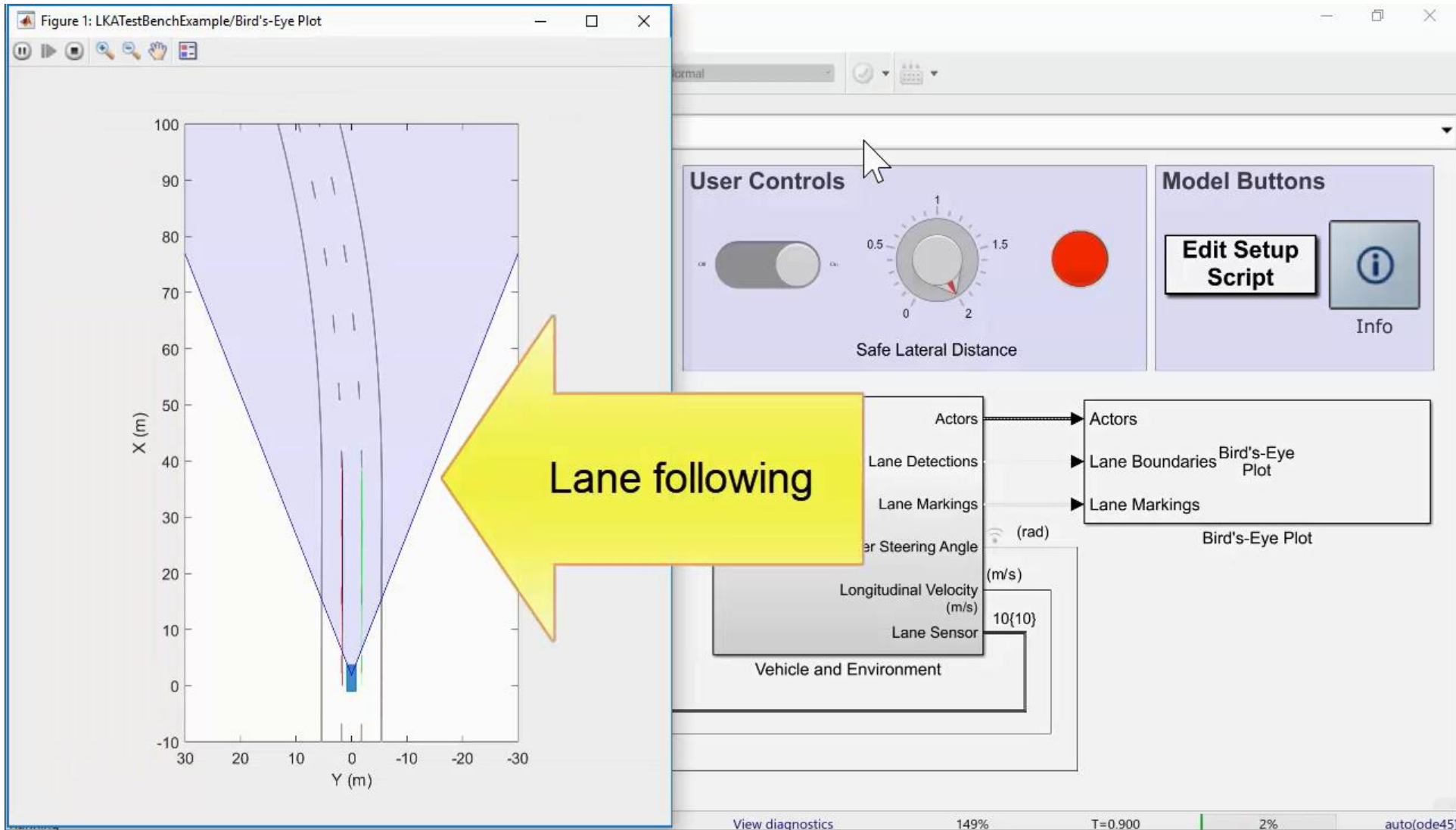
- Detect lane departure and maintain lane during distraction



Driver assisted at curvature change

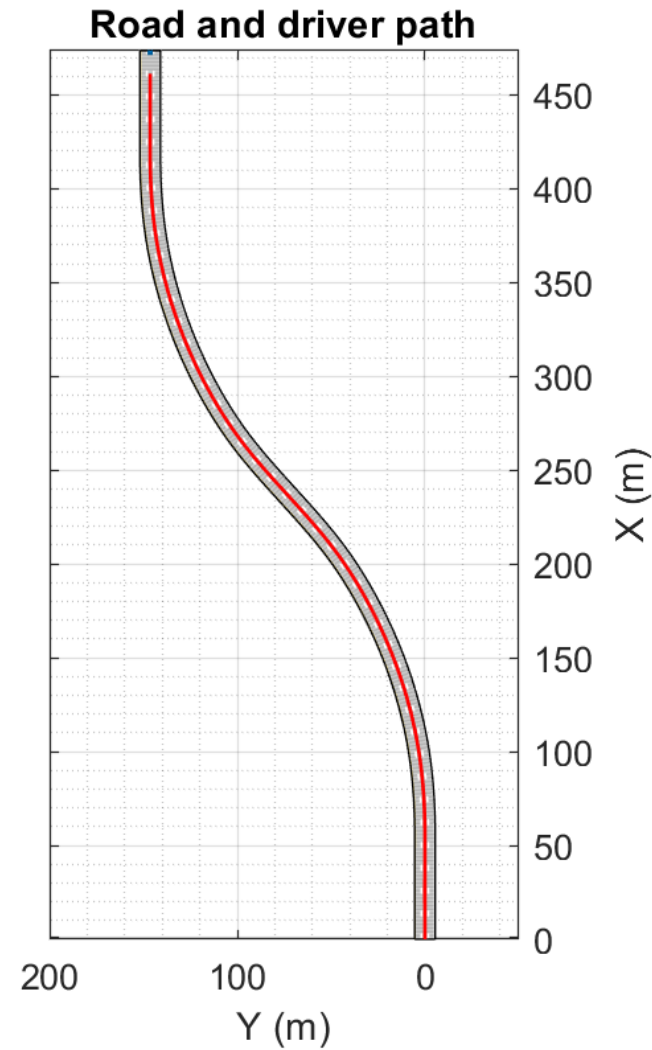


Simulate lane following by increasing minimum safe distance

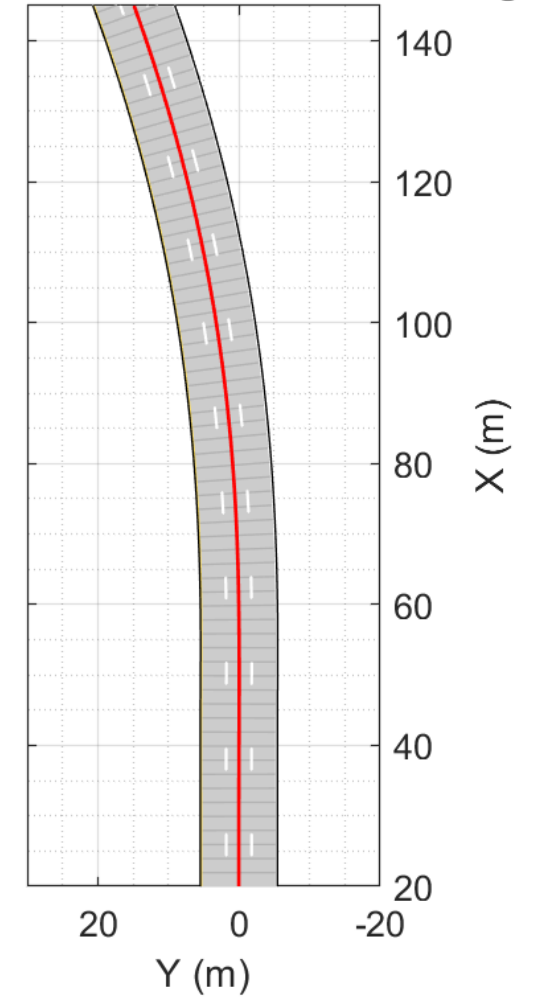


Explore lane following results

- Vehicle stays within lane boundaries



Driver assisted at curvature change



Graphically edit scenarios with Driving Scenario Designer

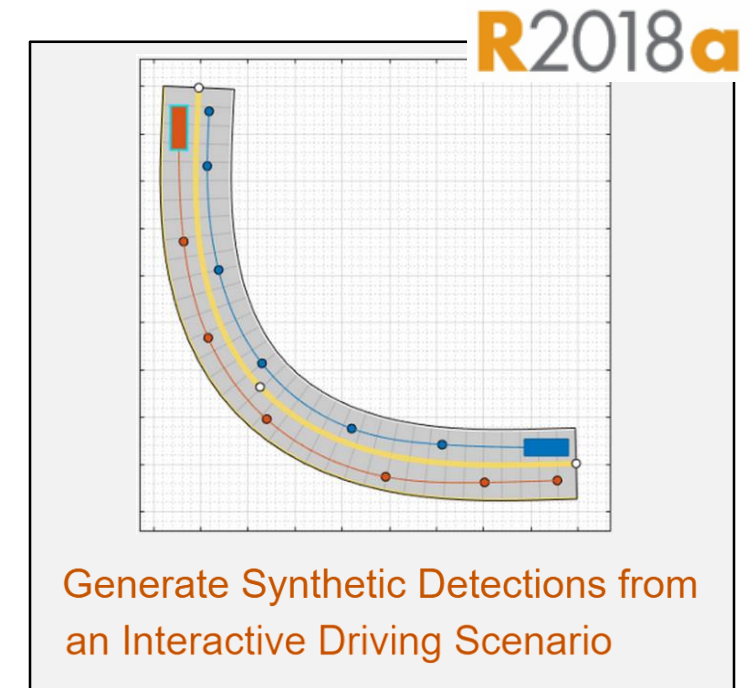
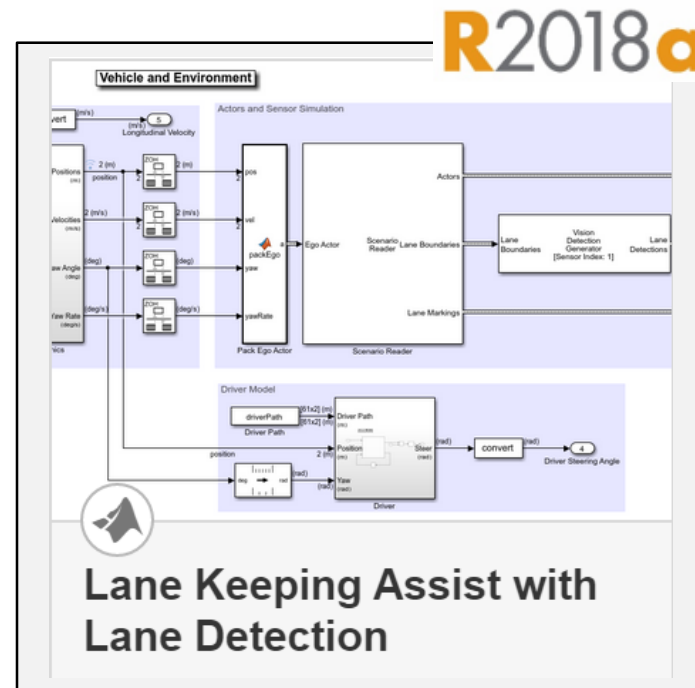
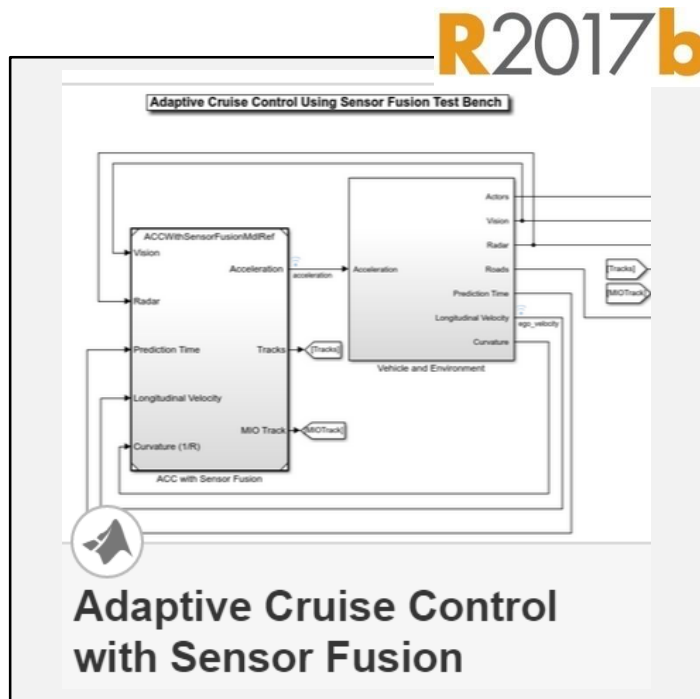
The screenshot displays the 'Driving Scenario Designer' software interface. The main window is titled 'Driving Scenario Designer - untitled* - Scenario Canvas'. The interface is divided into several sections:

- DESIGNER Toolbar:** Contains icons for New, Open, Save, Add Road, Add Actor, Add Camera, Add Radar, Go to Start, Step Back, Run, Step Forward, Settings, Default Layout, and Export.
- Left Panel (Properties):** Shows settings for the selected road, including 'Road: 1', 'Name:', 'Width (m): 6', and 'Bank Angle (deg): 0'. Below this is a table of 'Road Centers'.
- Road Centers Table:**

	x (m)	y (m)	z (m)
1	0	0	0
2	15	0	0
3	30	0	0
4	45	0	0
5	60	0	0
6	99.10...	3.0779	0
7	131.2...	19.80...	0
8	173.4...	27.24...	0
9	206.9...	47.74...	0
10	236.7...	73.22...	0
11	266.6...	98.70...	0
12	300.0...	119.1...	0
13	336.2...	134.2...	0
14	374.4...	143.3...	0
15	413.5...	146.4...	0
16	428.5...	146.4...	0
17	443.5...	146.4...	0
18	458.5...	146.4...	0
19	473.5...	146.4...	0
- Scenario Canvas:** A 2D plot showing a road curve. The vertical axis is labeled 'Y (m)' and ranges from 100 to -80. The horizontal axis is labeled 'Y (m)' and ranges from 200 to 0. A blue road curve is shown, with a mouse cursor hovering over one of the points on the curve.
- Ego Centric View:** A 3D perspective view of the road, showing a blue car model positioned on the road.

A large yellow arrow points from the 'Road Centers' table towards the 'Scenario Canvas' plot, with the text 'Graphically edit scenario' written inside it.

Learn about synthesizing sensor detections to develop control algorithms with these examples

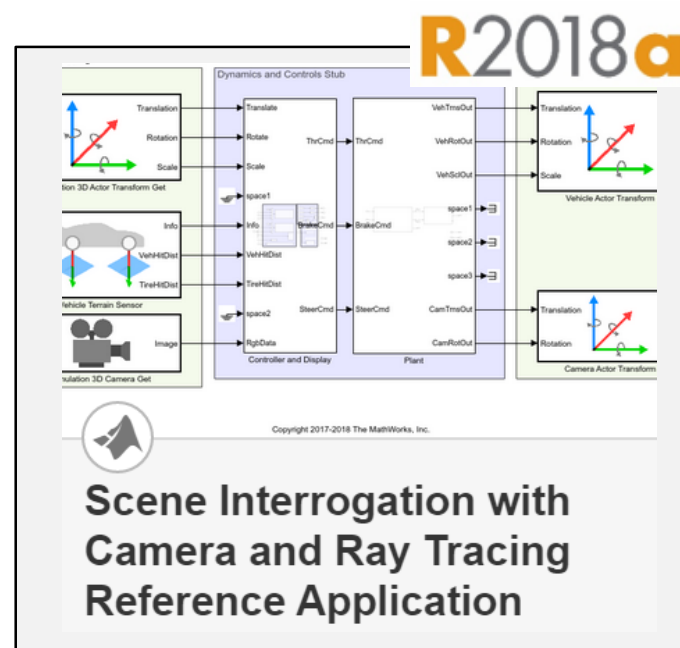
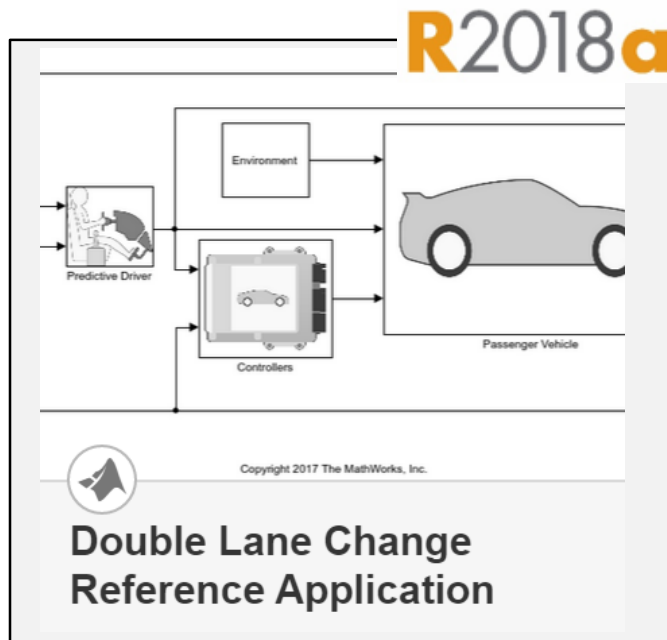


- **Simulate and generate C++** for model-predictive control and sensor fusion algorithms

- **Simulate and generate C++** for model-predictive control with lane detections

- **Edit roads, cuboid actors, and sensors** with Driving Scenario Designer App `drivingScenarioDesigner`

Learn about modeling vehicle dynamics to develop control algorithms with these examples



- **Simulate vehicle dynamics** for closed loop design
Vehicle Dynamics Blockset™

- **Co-simulate with Unreal Engine** and to set actor positions get camera image
Vehicle Dynamics Blockset™

Controls and Embedded Systems

14:30

Full Vehicle Simulation for Electrification and Automated Driving Applications

Prasanna Deshpande, MathWorks

R Vijayalayan, MathWorks

How can you use MATLAB and Simulink to develop planning algorithms?

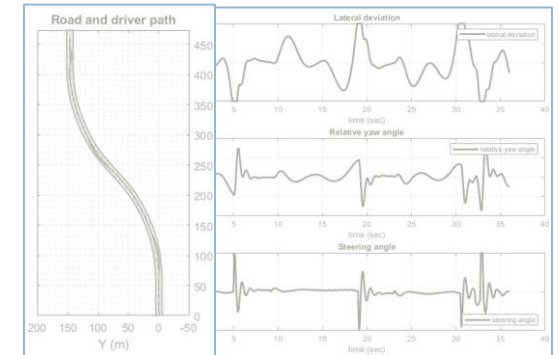
Deep learning



Perception

Control

Sensor models & model predictive control

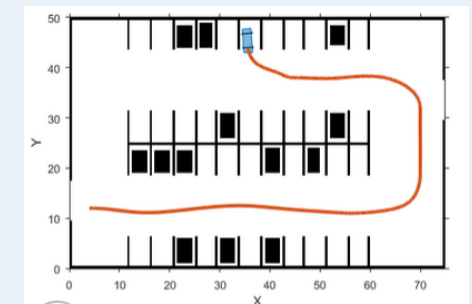


Sensor fusion with live data

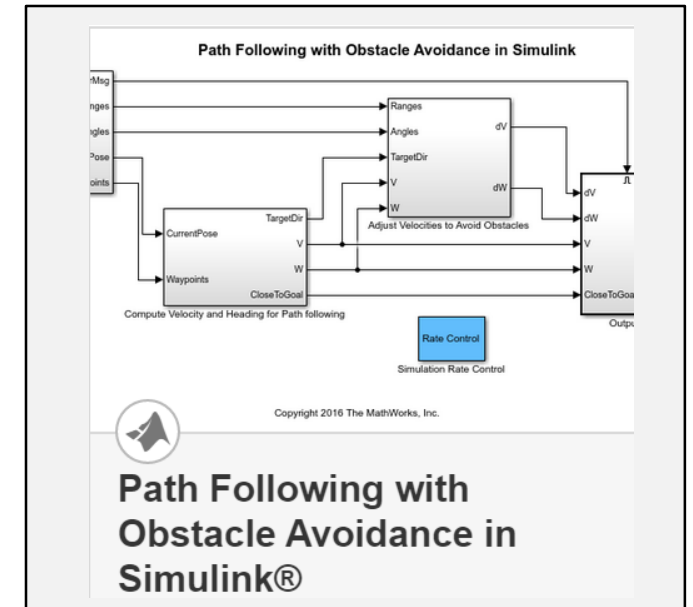
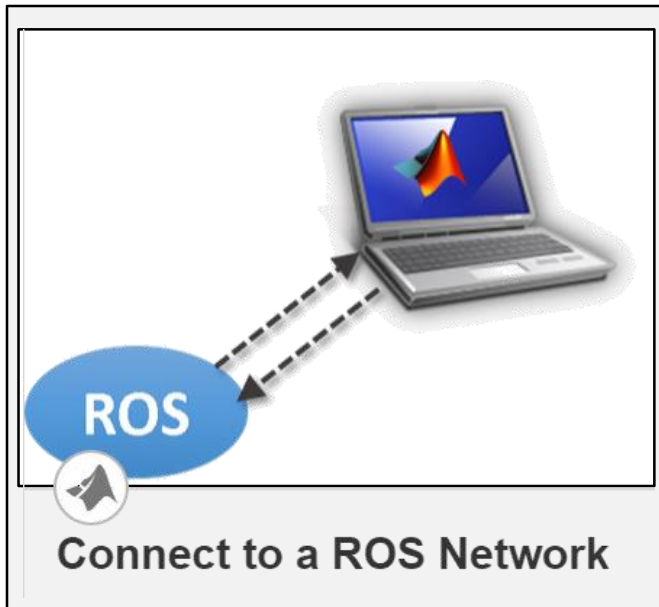


Planning

Path planning

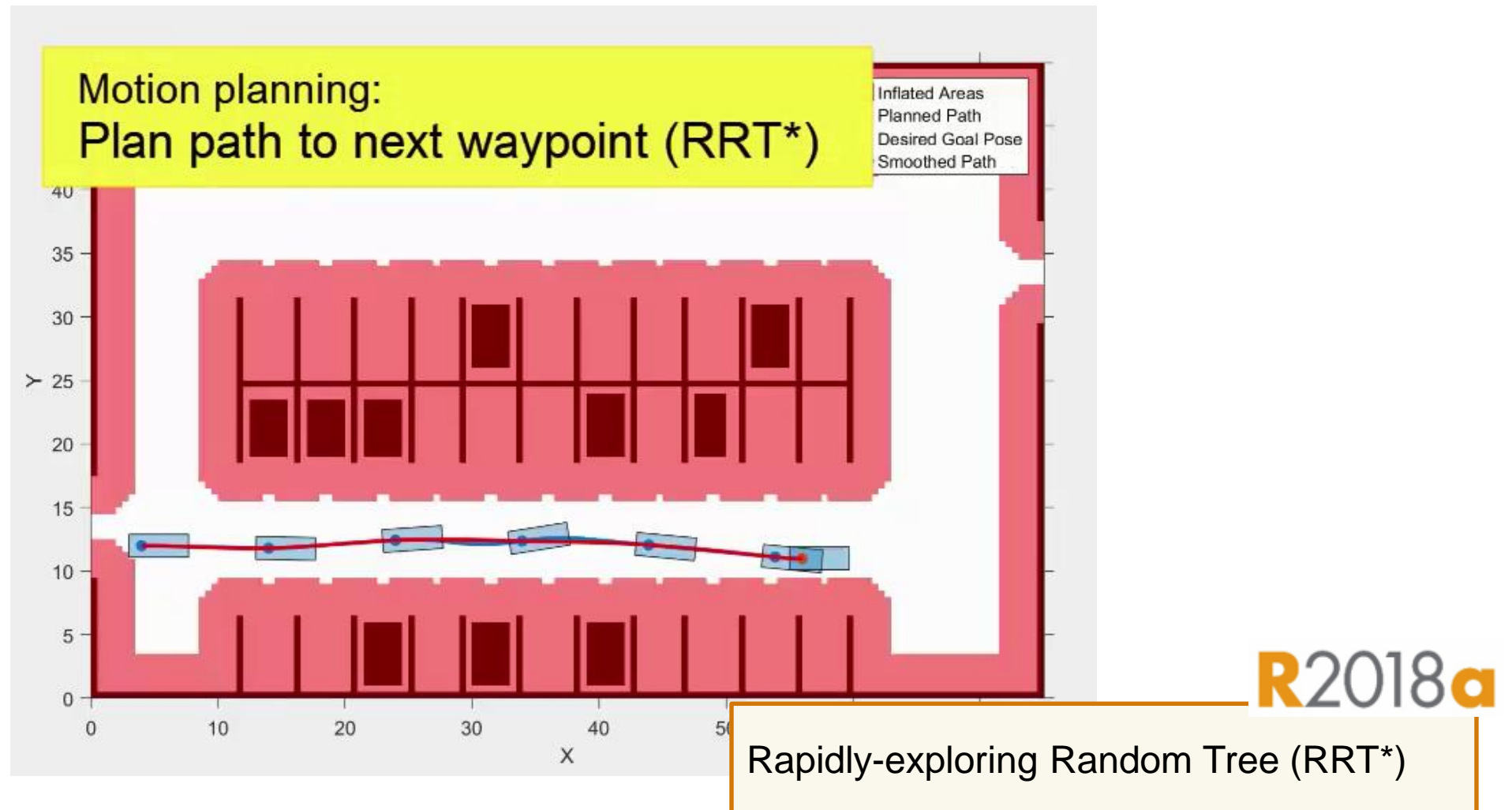


Robotics System Toolbox introduced: Connectivity with the ROS ecosystem

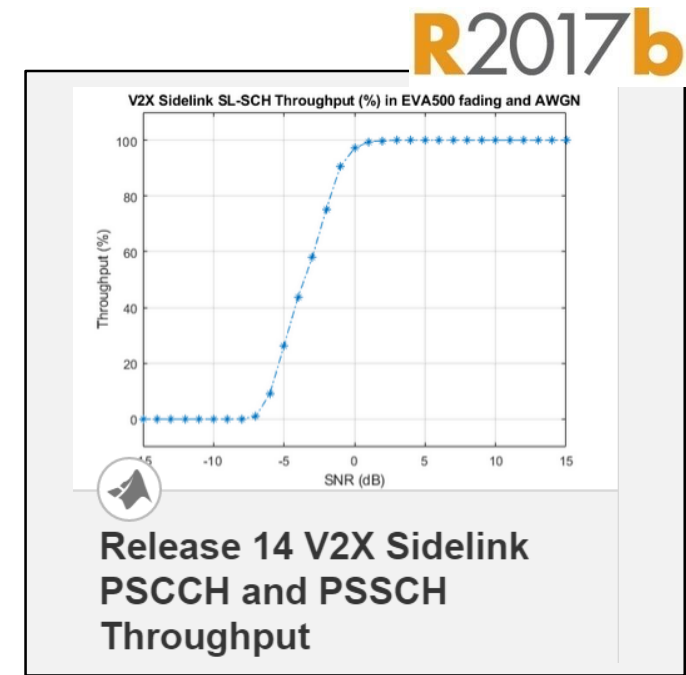
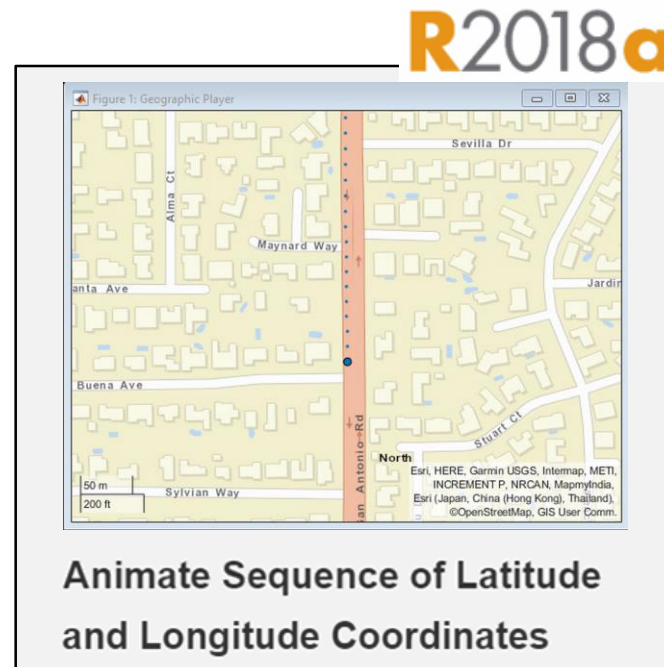
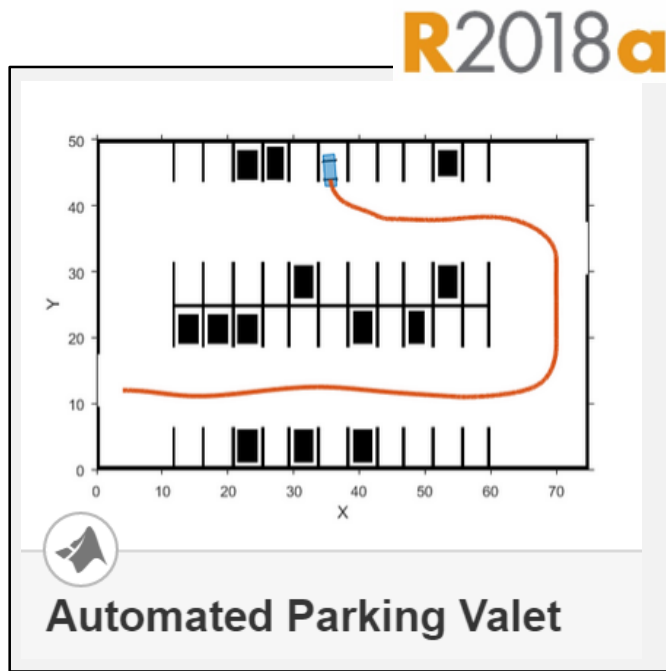


- **Communicate via ROS** to integrate with externally authored ROS components
- **Communication with Gazebo** to visualize and simulated system
- **Follow path** for differential drive robot with ROS based simulator

We are investing in design and simulation of path planning for automobiles



Learn about developing path planning algorithms with these examples



- **Plan path** for automobile given pre-defined map
Automated Driving System Toolbox™
- **Plot map tiles** using World Street Map (Esri)
Automated Driving System Toolbox™
- **Simulate V2X communication** to assess channel throughput
LTE System Toolbox™

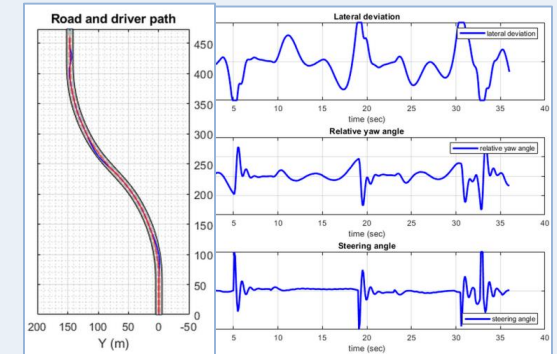
Examples of how you can use MATLAB and Simulink to develop automated driving algorithms

Deep learning



Perception

Sensor models & model predictive control



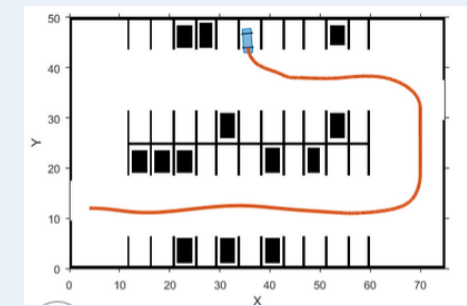
Control

Sensor fusion with live data

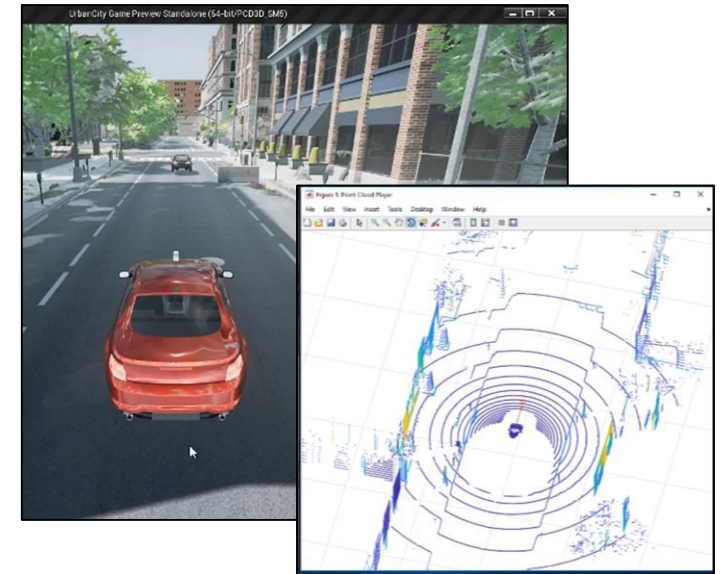
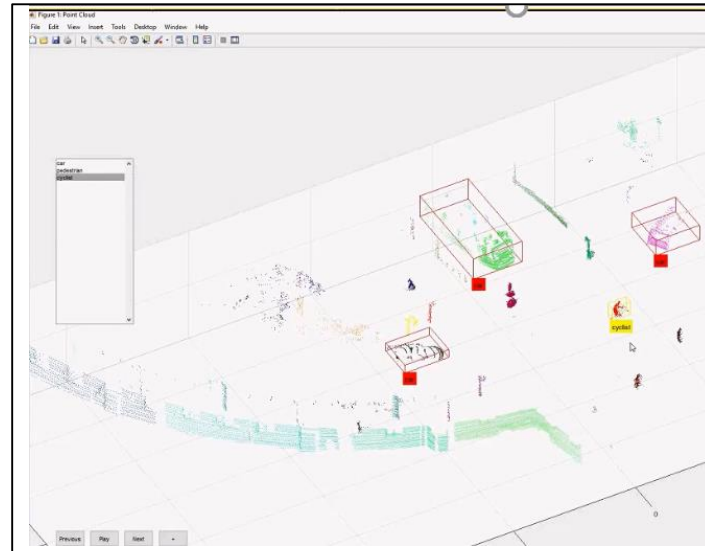


Planning

Path planning



MathWorks can help you customize MATLAB and Simulink for your automated driving application



- **Web based ground truth labeling**

- Consulting project with Caterpillar
- [2017 MathWorks Automotive Conference](#)

- **Lidar ground truth labeling**

- Joint presentation with Autoliv
- 2018 MathWorks Automotive Conference (May 2nd, Plymouth MI)

- **Lidar sensor model for Unreal Engine**

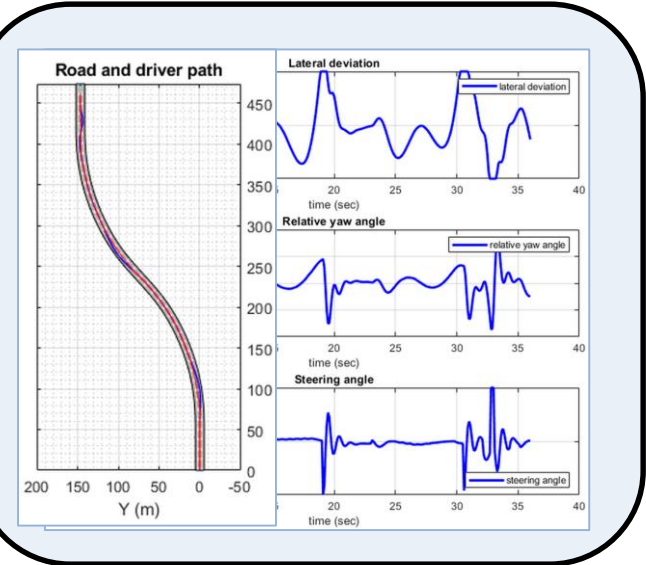
- Joint paper with Ford
- SAE Paper 2017-01-0107

How can we help you can use MATLAB and Simulink to develop automated driving algorithms?

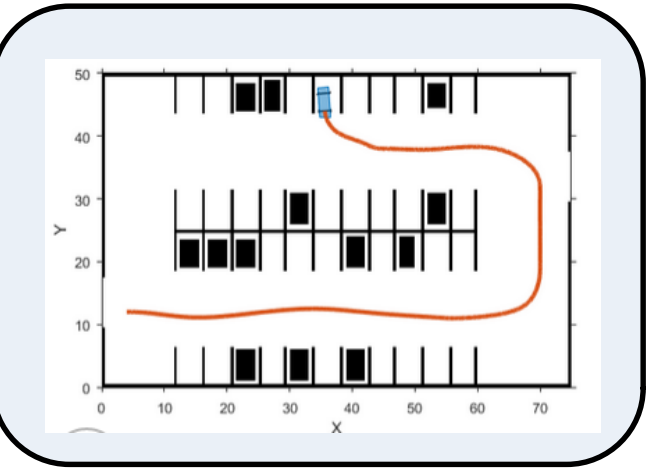


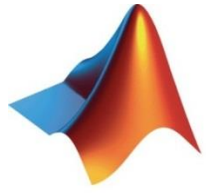
Perception

Control



Planning





MathWorks®

Accelerating the pace of engineering and science

Speaker Details

MANOHAR REDDY M

Email: Manohar.Reddy@mathworks.in

LinkedIn: <https://www.linkedin.com/in/manoharreddym>

Contact MathWorks India

Products/Training Enquiry Booth

Call: 080-6632-6000

Email: info@mathworks.in

Your feedback is valued.

Please complete the feedback form provided to you.