FSM Based Traffic light controller

Introduction

Decision logic for negotiating traffic lights is a fundamental component of an automated driving application. The decision logic must react to inputs like the state of the traffic light and surrounding vehicles. The decision logic then provides the controller with the desired velocity and path. Since traffic light intersections are dangerous to test, simulating such driving scenarios can provide insight into the interactions of the decision logic and the controller.

This example shows how to design and test the decision logic for negotiating a traffic light. The decision logic in this example reacts to the state of the traffic light, distance to the traffic light, and distance to the closest vehicle ahead. In this example, you will:

1. **Explore the test bench model**: The model contains the traffic light sensors and environment, traffic light decision logic, controls, and vehicle dynamics.
2. **Model the traffic light decision logic**: The traffic light decision logic arbitrates between a lead vehicle and an upcoming traffic light. It also provides a reference path for the ego vehicle to follow at an intersection in the absence of lanes.
3. **Simulate a left turn with traffic light and a lead vehicle**: The model is configured to test the interactions between the traffic light decision logic and controls of the ego vehicle, while approaching an intersection in the presence of a lead vehicle.
4. **Simulate a left turn with traffic light and cross traffic**: The model is configured to test the interactions between the traffic light decision logic and controls of the ego vehicle when there is cross traffic at the intersection.

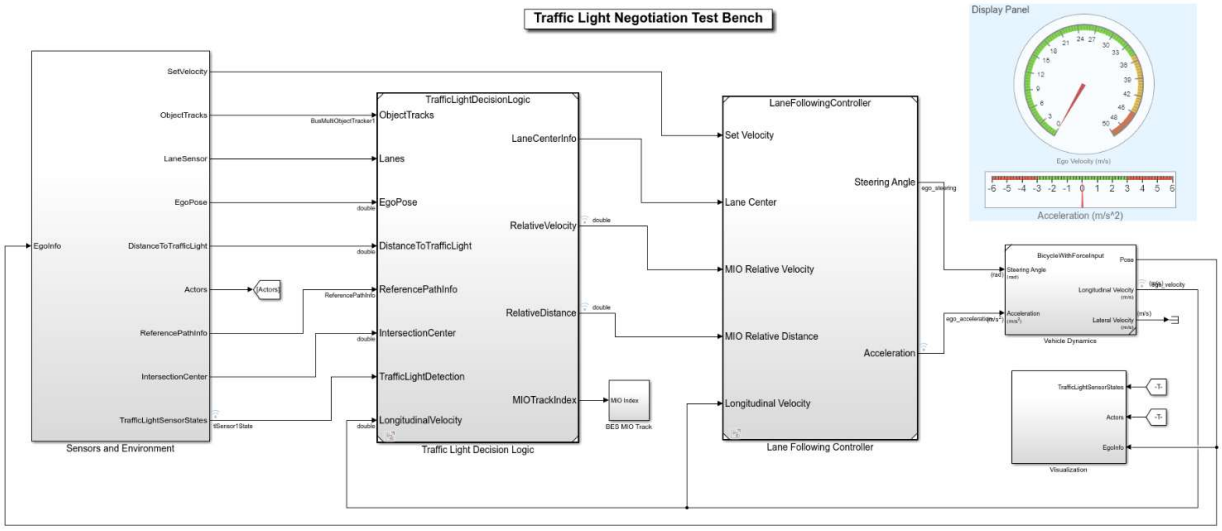
Explore Test Bench Model

To explore the test bench model, load the traffic light negotiation project.

```
openProject("TrafficLightNegotiation");
```

To explore the behavior of the traffic light negotiation system, open the simulation test bench model for the system.

```
open_system("TrafficLightNegotiationTestBench");
```



Opening this model runs the `helperSLTrafficLightNegotiationSetup` script that initializes the road scenario using the `drivingScenario` object in the base workspace. It runs the default test scenario, `scenario_02_TLN_left_turn_with_cross_over_vehicle`, that contains an ego vehicle and two other vehicles. This setup script also configures the controller design parameters, vehicle model parameters, and Simulink® bus signals required for defining the inputs and outputs for the `TrafficLightNegotiationTestBench` model.

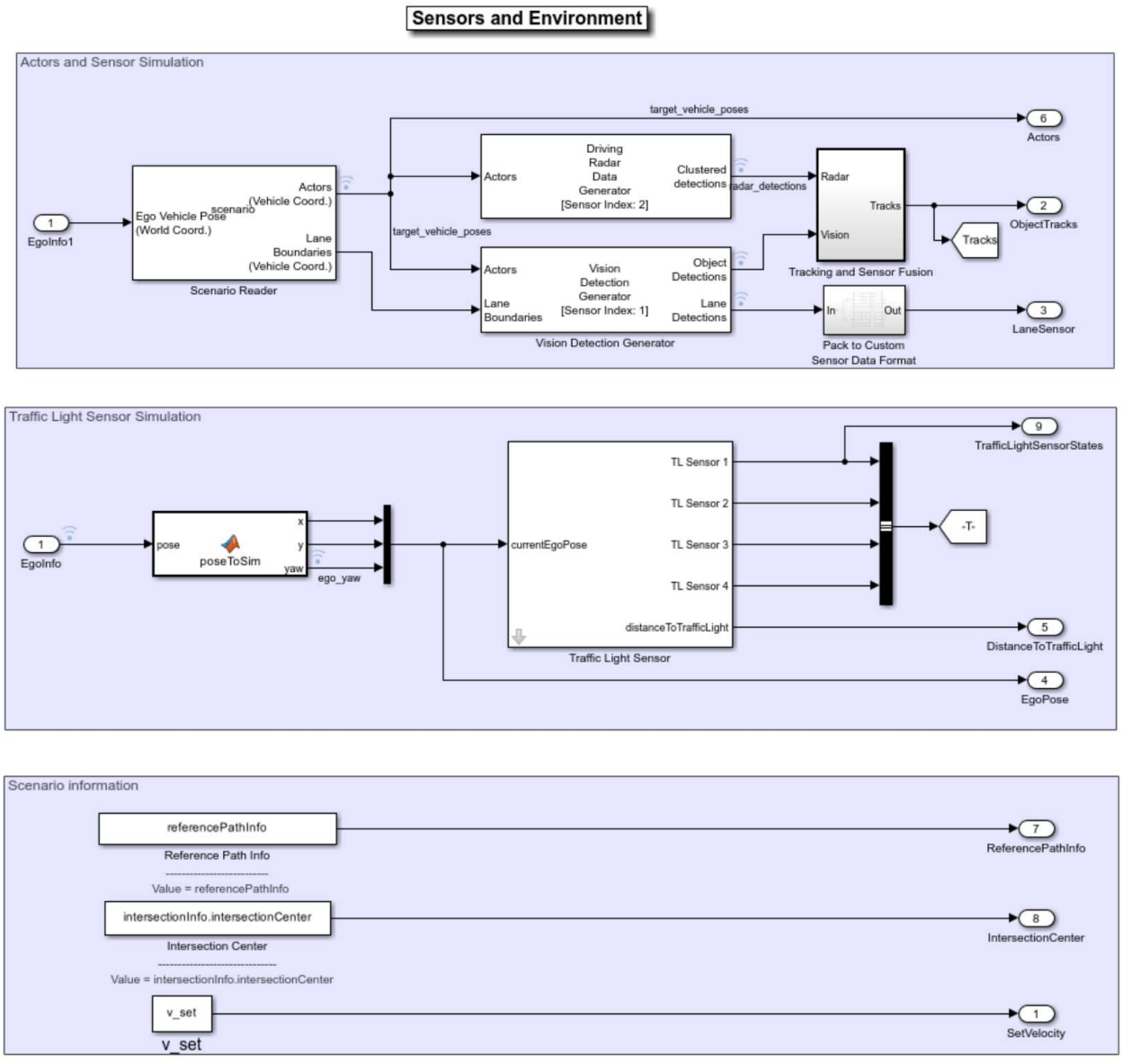
The test bench model contains the following subsystems:

1. **Sensors and Environment**: Models the traffic light sensor, road network, vehicles, and the camera and radar sensors used for simulation.
2. **Traffic Light Decision Logic**: Arbitrates between the traffic light and other lead vehicles or cross-over vehicles at the intersection.
3. **Lane-Following Controller**: Generates longitudinal and lateral controls.
4. **Vehicle Dynamics**: Models the ego vehicle using a Bicycle Model block and updates its state using commands received from the **Lane Following Controller** subsystem.
5. **Visualization**: Plots the world coordinate view of the road network, vehicles, and the traffic light state during simulation.

The **Lane Following Controller** reference model and the **Vehicle Dynamics** subsystem are reused from the Highway Lane Following example. This example focuses on the **Sensors and Environment** and **Traffic Light Decision Logic** subsystems.

The **Sensors and Environment** subsystem configures the road network, defines target vehicle trajectories, and synthesizes sensors. Open the **Sensors and Environment** subsystem.

```
open_system("TrafficLightNegotiationTestBench/Sensors and Environment");
```

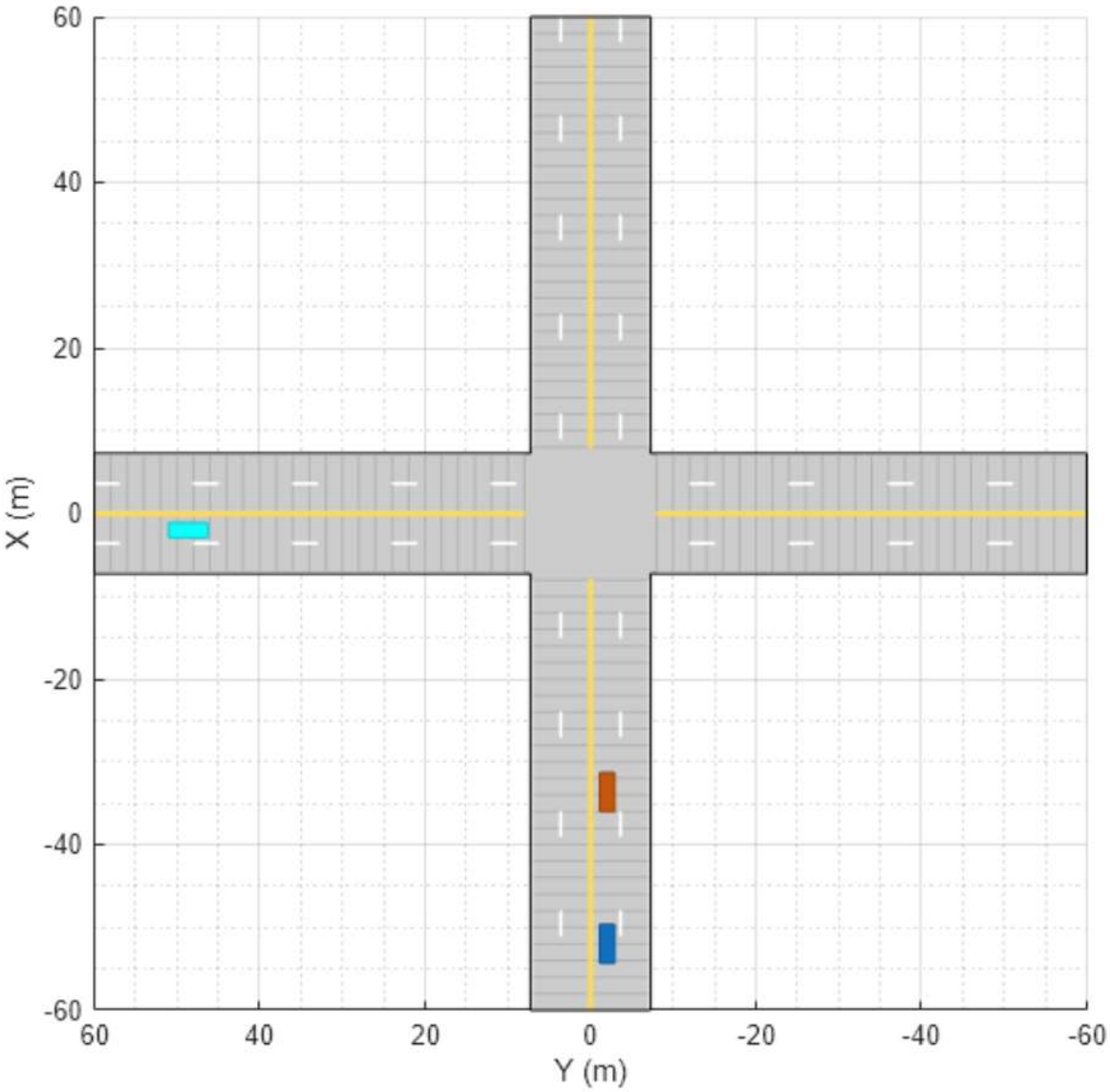


The scenario and sensors on the ego vehicle are specified by the following parts of the subsystem:

- The Scenario Reader block is configured to take in ego vehicle information to perform a closed-loop simulation. It outputs ground truth information of lanes and actors in ego vehicle coordinates. This block reads the `drivingScenario` object variable, `scenario`, from the base workspace, which contains a road network compatible with the `TrafficLightNegotiationTestBench` model.

Plot the road network provided by the scenario.

```
hFigScenario = figure('Position', [1 1 800 600]);
plot(scenario, 'Parent', axes(hFigScenario));
```



This default scenario has one intersection with an ego vehicle, one lead vehicle, and one cross-traffic vehicle.

Close the figure.

```
close(hFigScenario);
```

The **Tracking and Sensor Fusion** subsystem fuses vehicle detections from Driving Radar Data Generator and Vision Detection Generator blocks by using a Multi-Object Tracker block to provide object tracks surrounding the ego vehicle.

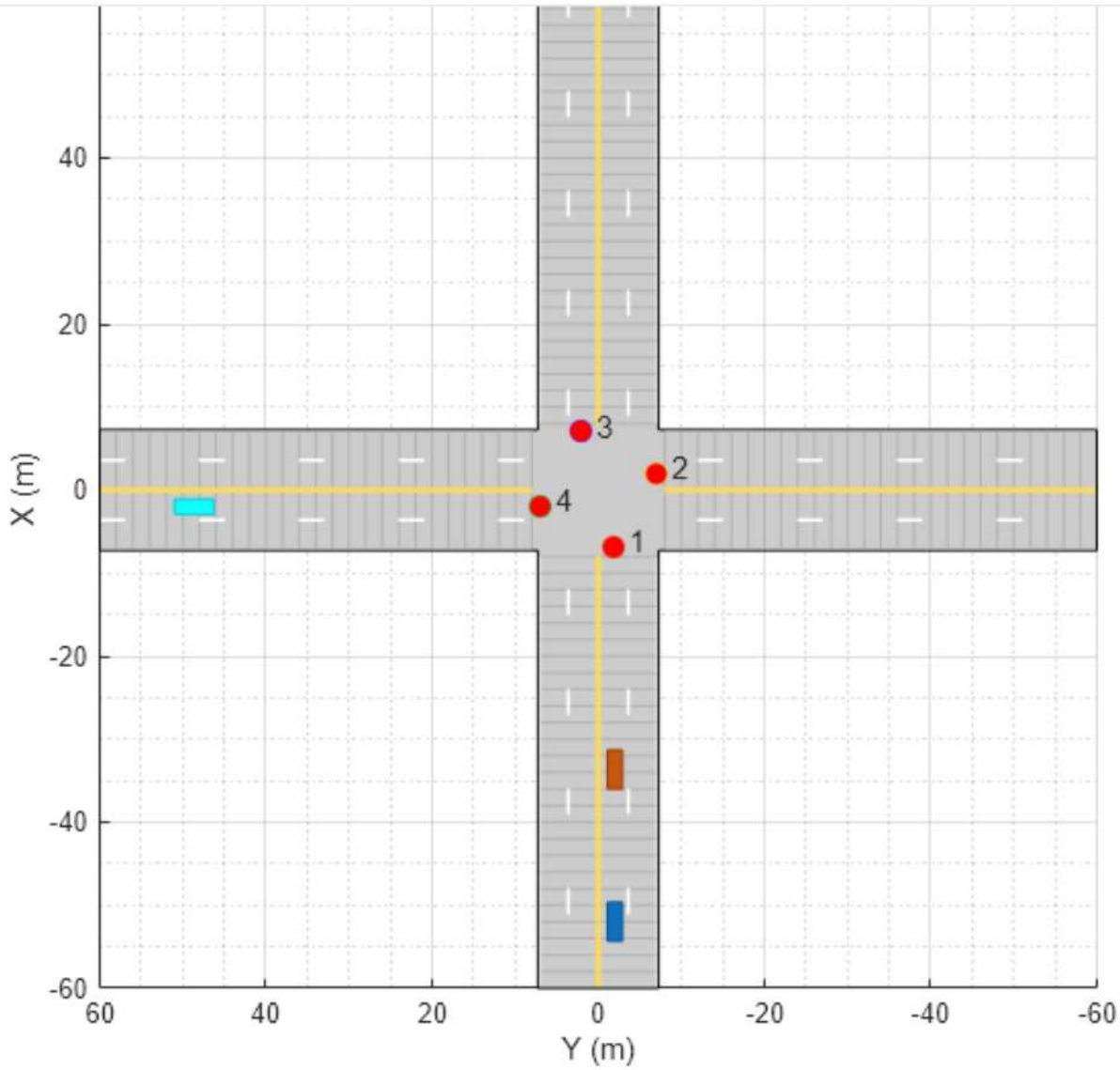
The Vision Detection Generator block also provides lane detections with respect to the ego vehicle that helps in identifying vehicles present in the ego lane.

The **Traffic Light Sensor** subsystem simulates the traffic lights. It is configured to support four traffic light sensors at the intersection, **TL Sensor 1**, **TL Sensor 2**, **TL Sensor 3**, and **TL Sensor 4**.

Plot the scenario with traffic light sensors.

```
hFigScenario = helperPlotScenarioWithTrafficLights();
```

nples Functions Blocks Apps Scenes Videos Answers



Observe that this is the same scenario as before, only with traffic light sensors added. These sensors are represented by red circles at the intersection, indicating red traffic lights. The labels for the traffic lights **1**, **2**, **3**, **4** correspond to **TL Sensor 1**, **TL Sensor 2**, **TL Sensor 3**, and **TL Sensor 4**, respectively.

Close the figure.

```
close(hFigScenario);
```

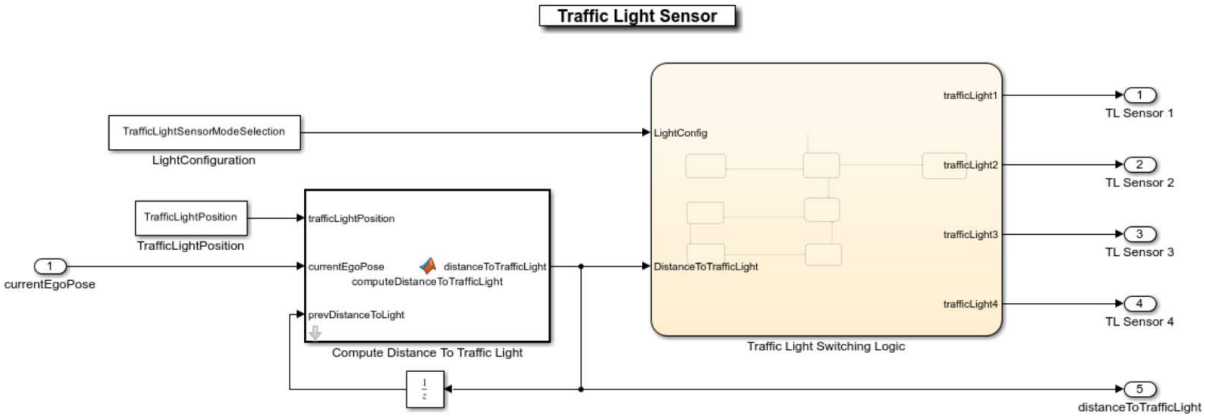
The test scenarios in `TrafficLightNegotiationTestBench` are configured such that the ego vehicle negotiates with **TL Sensor 1**. There are three modes in which you can configure this **Traffic Light Sensor** subsystem:

1. **Steady Red**: **TL Sensor 1** and **TL Sensor 3** are always in a red state. The other two traffic lights are always in a green state.
2. **Steady Green**: **TL Sensor 1** and **TL Sensor 3** are always in a green state. The other two traffic lights are always in a red state.
3. **Cycle** [Default]: **TL Sensor 1** and **TL Sensor 3** follow a cyclic pattern: green-yellow-red with predefined timings. Other traffic lights also follow a cyclic pattern: red-green-yellow with predefined timings to complement the **TL Sensor 1** and **TL Sensor 3**.

You can configure this subsystem in one of these modes by using the `Traffic Light Sensor Mode` mask parameter.

Open the **Traffic Light Sensor** subsystem.

```
open_system(['TrafficLightNegotiationTestBench/' ...
    'Sensors and Environment/Traffic Light Sensor'], 'force');
```



The **Traffic Light Switching Logic** Stateflow® chart implements the traffic light state change logic for the four traffic light sensors. The initial state for all the traffic lights is set to red. Transition to a different mode is based on a trigger condition defined by distance of the ego vehicle to the **TL Sensor 1** traffic light. This distance is defined by the variable `distanceToTrafficLight`. Traffic light transition is triggered if this distance is less than `trafficLightStateTriggerThreshold`. This threshold is currently set to 60 meters and can be changed in the `helperSLTrafficLightNegotiationSetup` script.

The Compute Distance To Traffic Light block calculates `distanceToTrafficLight` using the traffic light position of **TL Sensor 1**, defined by the variable `trafficLightPosition`. This is obtained from the `Traffic Light Position` mask parameter of the **Traffic Light Sensor** subsystem. The value of the mask parameter is set to `intersectionInfo.tlSensor1Position`, a variable set in the base workspace by the `helperSLTrafficLightNegotiationSetup` script. `intersectionInfo` structure is an output from the `helperGetTrafficLightScene` function. This function is used to create the test scenarios that are compatible with the `TrafficLightNegotiationTestBench` model.

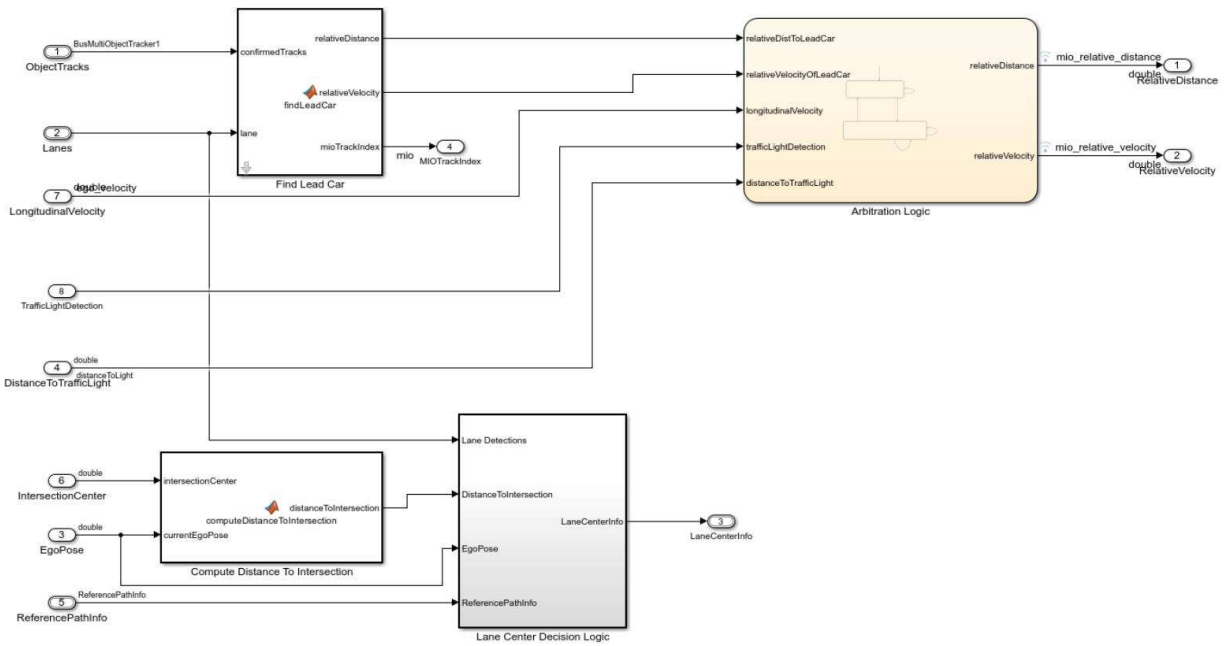
The following inputs are needed by the traffic light decision logic and controller to implement their functionalities:

- **ReferencePathInfo** provides a predefined reference trajectory that can be used by the ego vehicle for navigation in absence of lane information. The ego vehicle can go straight, take a left turn, or a right turn at the intersection based on the reference path. This reference path is obtained using `referencePathInfo`, an output from `helperGetTrafficLightScene`. This function takes an input argument to specify the direction of travel at the intersection. The possible values are: `Straight`, `Left`, and `Right`.
- **IntersectionCenter** provides the position of the intersection center of the road network in the scenario. This is obtained using the `intersectionInfo`, an output from `helperGetTrafficLightScene`.
- **Set Velocity** defines the user-set velocity for the controller.

Model Traffic Light Decision Logic

The **Traffic Light Decision Logic** reference model arbitrates between the lead car and the traffic light. It also calculates the lane center information as required by the controller either using the detected lanes or a predefined path. Open the **Traffic Light Decision Logic** reference model.

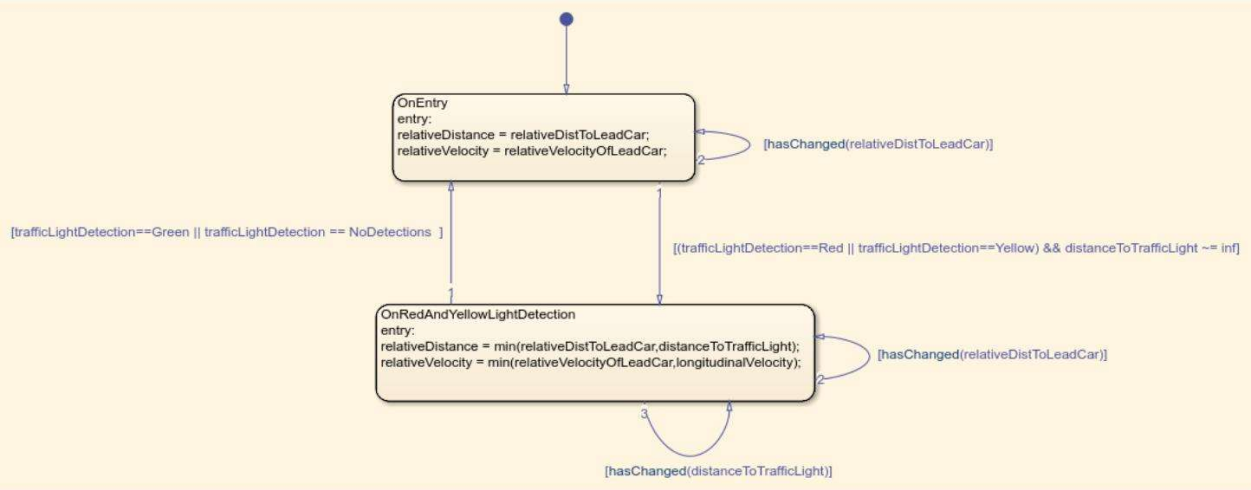
```
open_system("TrafficLightDecisionLogic");
```



The **Find Lead Car** subsystem finds the lead car in the current lane from input object tracks. It provides relative distance, **relativeDistToLeadCar**, and relative velocity, **relativeVelocityOfLeadCar**, with respect to the lead vehicle. If there is no lead vehicle, then this block considers the lead vehicle to be present at infinite distance.

The **Arbitration Logic** Stateflow chart uses the lead car information and implements the logic required to arbitrate between the traffic light and the lead vehicle at the intersection. Open the **Arbitration Logic** Stateflow chart.

```
open_system("TrafficLightDecisionLogic/Arbitration Logic");
```



The **Arbitration Logic** Stateflow chart consists of two states, `OnEntry` and `OnRedAndYellowLightDetection`. If the traffic light state is green or if there are no traffic light detections, the state remains in the `OnEntry` state. If the traffic light state is red or yellow, then the state transitions to the `OnRedAndYellowLightDetection` state. The control flow switches between these states based on `trafficLightDetection` and `distanceToTrafficLight` variables. In each state, relative distance and relative velocity with respect to the most important object (MIO) are calculated. The lead vehicle and the red traffic light are considered as MIOs.

OnEntry:

```
relativeDistance = relativeDistToLeadCar;
```

```
relativeVelocity = relativeVelocityOfLeadCar;
```

OnRedAndYellowLightDetection:

```
relativeDistance = min(relativeDistToLeadCar,distanceToTrafficLight);
```

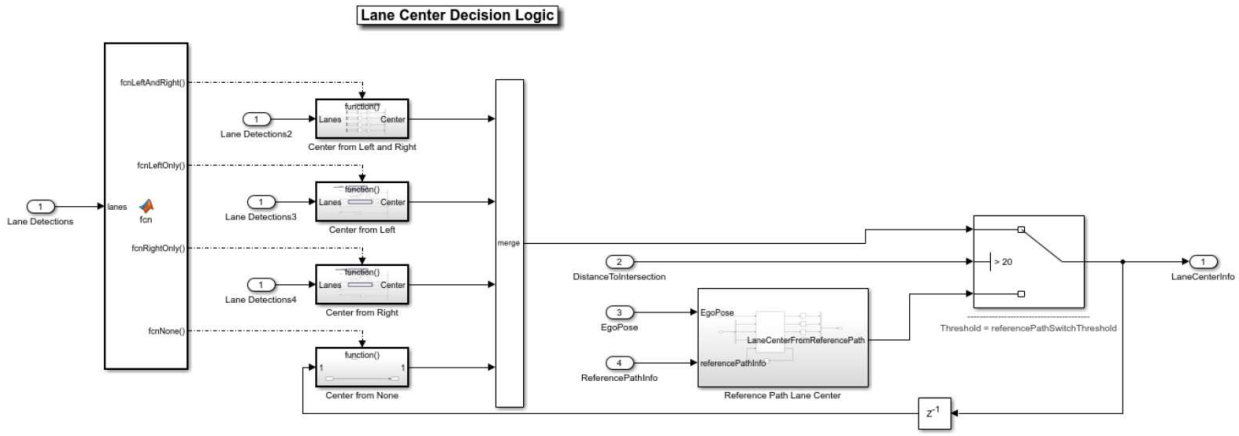
```
relativeVelocity = min(relativeVelocityOfLeadCar,longitudinalVelocity);
```

The **longitudinalVelocity** represents the longitudinal velocity of the ego vehicle.

The Compute Distance To Intersection block computes the distance to the intersection center from the current ego position. Because the intersection has no lanes, the ego vehicle uses this distance to fall back to the predefined reference path at the intersection.

The **Lane Center Decision Logic** subsystem calculates the lane center information as required by the Path Following Control System (Model Predictive Control Toolbox). Open the **Lane Center Decision Logic** subsystem.

```
open_system("TrafficLightDecisionLogic/Lane Center Decision Logic");
```



The **Lane Center Decision Logic** subsystem primarily relies on the lane detections from the Vision Detection Generator block to estimate lane center information like curvature, curvature derivative, lateral offset, and heading angle. However, there are no lane markings to detect at the intersection. In such cases, the lane center information can be estimated from a predefined reference path.

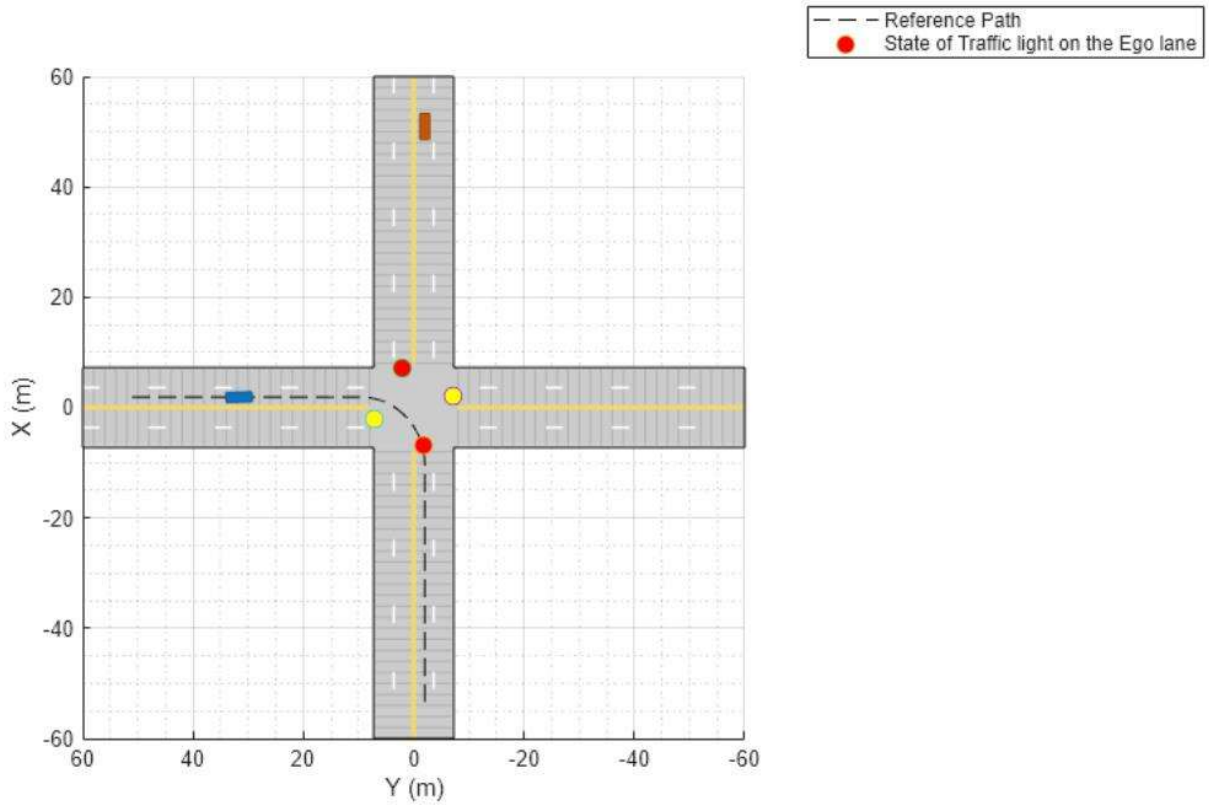
The **Reference Path Lane Center** subsystem computes lane center information based on the current ego pose and predefined reference path. A switch is configured to use **LaneCenterFromReferencePath** when **DistanceToIntersection** is less than `referencePathSwitchThreshold`. This threshold is currently set to 20 meters and can be changed in the `helperSLTrafficLightNegotiationSetup` script.

Simulate Left Turn with Traffic Light and Lead Vehicle

In this test scenario, a lead vehicle travels in the ego lane and crosses the intersection. The traffic light state keeps green for the lead vehicle and turns red for the ego vehicle. The ego vehicle is expected to follow the lead vehicle, negotiate the traffic light, and make a left turn.

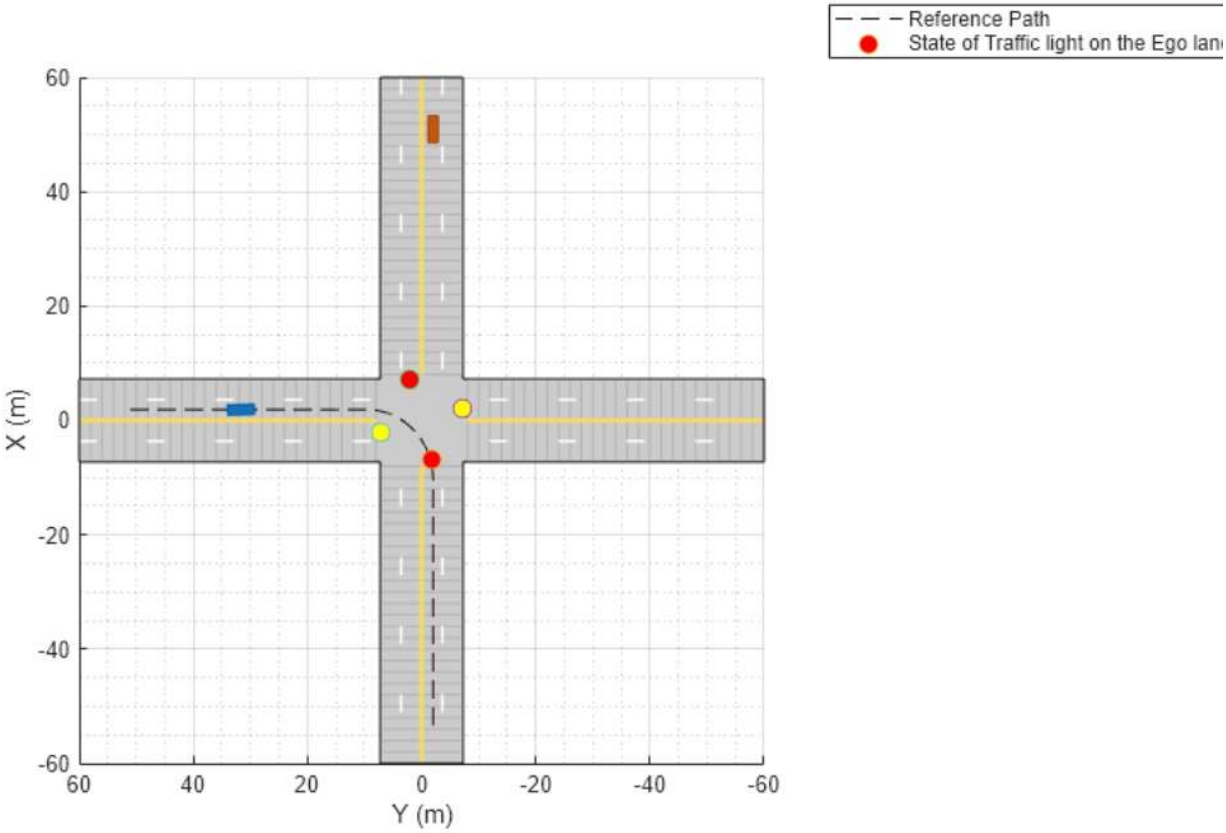
Configure the `TrafficLightNegotiationTestBench` model to use the `scenario_03_TLN_left_turn_with_lead_vehicle` scenario.

```
helperSLTrafficLightNegotiationSetup( ...
    "scenario_03_TLN_left_turn_with_lead_vehicle");
% To reduce command-window output, first turn off the MPC update messages.
mpcverbosity('off');
% Simulate the model.
sim("TrafficLightNegotiationTestBench");
```



Plot the simulation results.

```
hFigResults = helperPlotTrafficLightNegotiationResults(logsout);
```



Examine the results.

- The **Traffic light state - TL Sensor 1** plot shows the traffic light sensor states of **TL Sensor 1**. It changes from green to yellow, then from yellow to red, and then repeats in **Cycle** mode.
- The **Relative longitudinal distance** plot shows the relative distance between the ego vehicle and the MIO. Notice that the ego vehicle follows the lead vehicle from 0 to 4.2 seconds by maintaining a safe distance from it. You can also observe that from 4.2 to 9 seconds, this distance reduces because the red traffic light is detected as an MIO. Also notice the gaps representing infinite distance when there is no MIO after the lead vehicle exceeds the maximum distance allowed for an MIO.
- The **Ego acceleration** plot shows the acceleration profile from the **Lane Following Controller**. Notice the negative acceleration from 4.2 to 4.7 seconds, in reaction to the detection of the red traffic light as an MIO. You can also observe the increase in acceleration after 9 seconds, in response to the green traffic light.
- The **Ego yaw angle** plot shows the yaw angle profile of the ego vehicle. Notice the variation in this profile after 12 seconds, in response to the ego vehicle taking a left turn.

Close the figure.

```
close(hFigResults);
```

Simulate Left Turn with Traffic Light and Cross Traffic

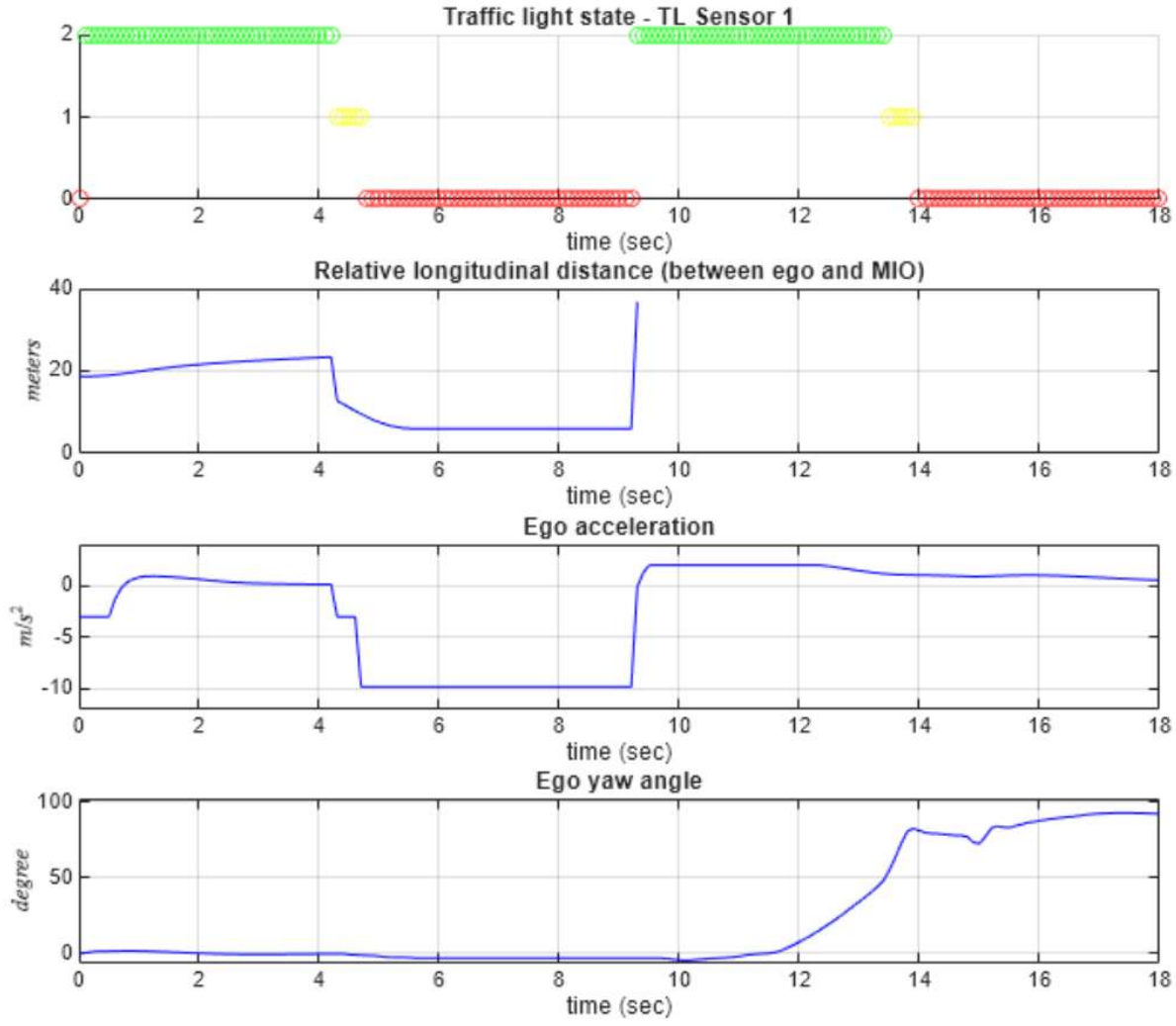
This test scenario is an extension to the previous scenario. In addition to the previous conditions, in this scenario, a slow-moving cross-traffic vehicle is in the intersection when the traffic light is green for the ego

vehicle. The ego vehicle is expected to wait for the cross-traffic vehicle to pass the intersection before taking the left turn.

Configure the `TrafficLightNegotiationTestBench` model to use the `scenario_02_TLN_left_turn_with_cross_over_vehicle` scenario.

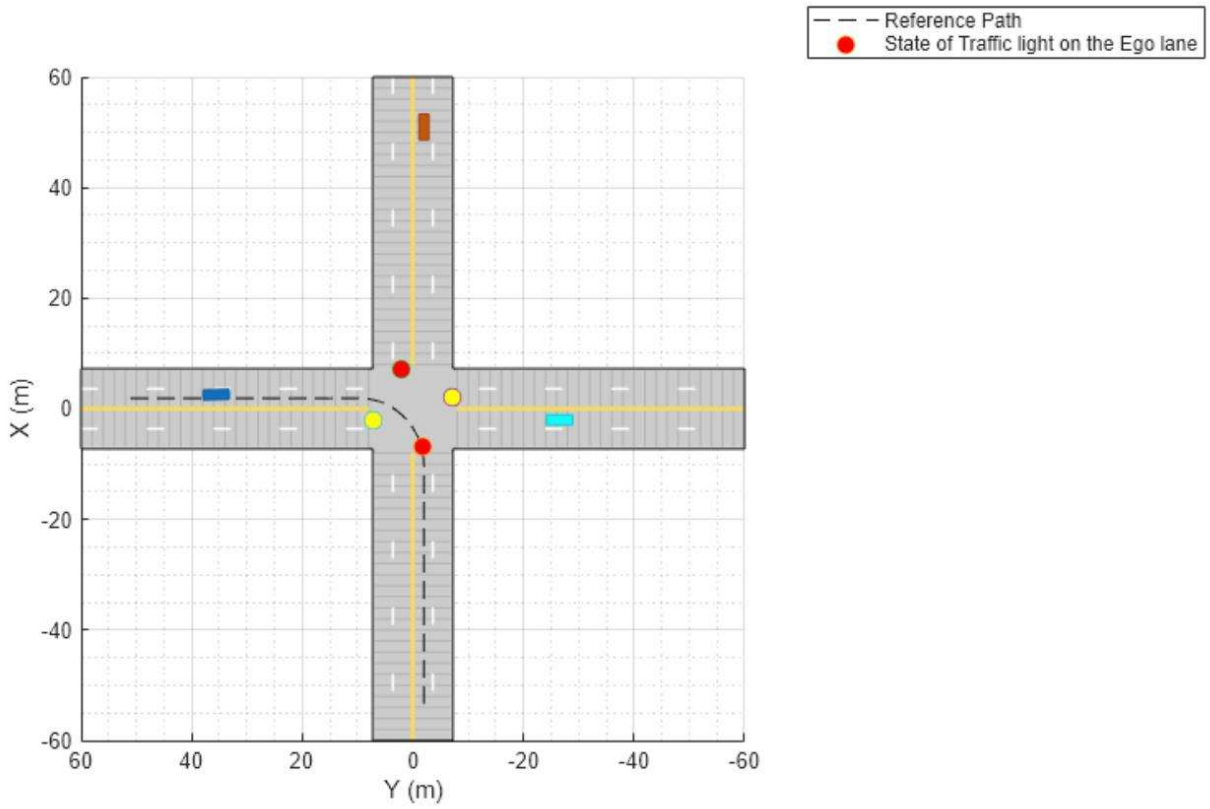
```
helperSLTrafficLightNegotiationSetup( ...
    "scenario_02_TLN_left_turn_with_cross_over_vehicle");

% Simulate the model.
sim("TrafficLightNegotiationTestBench");
```



Plot the simulation results.

```
hFigResults = helperPlotTrafficLightNegotiationResults(logsout);
```



Examine the results.

- The **Traffic light state - TL Sensor 1** plot is same as the one from the previous simulation.
- The **Relative longitudinal distance** plot diverges from the previous simulation run from 10.5 seconds onward. Notice the detection of the cross-traffic vehicle as the MIO at 10 seconds at around 10 meters.
- The **Ego acceleration** plot also quickly responds to the cross-traffic vehicle at 10.6. You can notice a hard-braking profile in response to the cross-traffic vehicle at the intersection.
- The **Ego yaw angle** plot shows that the ego vehicle initiates a left turn after 14 seconds, in response to the cross-traffic vehicle leaving the intersection.

Close the figure.

```
close(hFigResults);
```

Explore Other Scenarios

In the previous sections, you explored the system behavior for the `scenario_03_TLN_left_turn_with_lead_vehicle` and `scenario_02_TLN_left_turn_with_cross_over_vehicle` scenarios. Below is a list of scenarios that are compatible with `TrafficLightNegotiationTestBench`.

```
scenario_01_TLN_left_turn
 scenario_02_TLN_left_turn_with_cross_over_vehicle [Default]
 scenario_03_TLN_left_turn_with_lead_vehicle
```

```
scenario_04_TLN_straight
scenario_05_TLN_straight_with_lead_vehicle
```

Use these additional scenarios to analyze `TrafficLightNegotiationTestBench` under different conditions. For example, while learning about the interactions between the traffic light decision logic and controls, it can be helpful to begin with a scenario that has an intersection with a traffic light but no vehicles. To configure the model and workspace for such a scenario, use this code:

```
helperSLTrafficLightNegotiationSetup( ...
    "scenario_04_TLN_straight");
```

Enable the MPC update messages.

```
mpcverbosity('on');
```

Conclusion

In this example, you implemented decision logic for traffic light negotiation and tested it with a lane following controller in a closed-loop Simulink model.