POLITECNICO DI TORINO

Master's Degree in MECHATRONIC ENGINEERING

Master's Degree Thesis

Scenario-Based Simulation for Advanced Driver Assistance Systems Validation

Supervisors

Prof. ANDREA TONOLI

PhD Candidate STEFANO FAVELLI

Candidate

YUWEI ZHANG

JULY 2025

Abstract

Advanced Driver Assistance Systems (ADAS) and intelligent vehicle control technologies require rigorous validation before deployment. While real-world testing offers realism, it remains expensive, time-consuming, and often limited by regulatory constraints or unsuitable for evaluating early-stage control strategies or edge-case behaviors. Virtual simulation provides a safe, configurable, and repeatable environment for functional testing, particularly for systems that rely on rule-based activation and real-time control response.

This thesis presents a co-simulation framework for function-level vehicle control validation. The platform integrates MATLAB/Simulink for control algorithm design with RoadRunner, which implements road scenario generation and traffic modeling. Real-time interaction is achieved through standardized interface blocks, allowing the control system to dynamically respond to lane geometry, lead vehicle behaviors, and other driving events defined in simulation.

To evaluate system performance, a range of representative test scenarios has been designed. These include typical highway and urban driving cases involving speed regulation, stopping behavior, and lateral tracking. System behavior is assessed based on control smoothness, rule compliance, and activation thresholds. In addition, the platform supports the integration of custom controllers and real-world GPS-based map data, allowing adaptation to tasks such as trajectory tracking, motion planning, and performance-oriented driving.

The proposed framework introduces a modular and simulation-driven strategy for testing ADAS functions and demonstrates its potential as a practical foundation for academic research, early-stage controller development, and regulation-oriented intelligent vehicle validation.

Keywords: Advanced Driver Assistance Systems (ADAS), co-simulation, function validation, MATLAB/Simulink, RoadRunner, scenario-based testing, virtual control evaluation, vehicle automation

Table of Contents

Chapter 1

Introduction

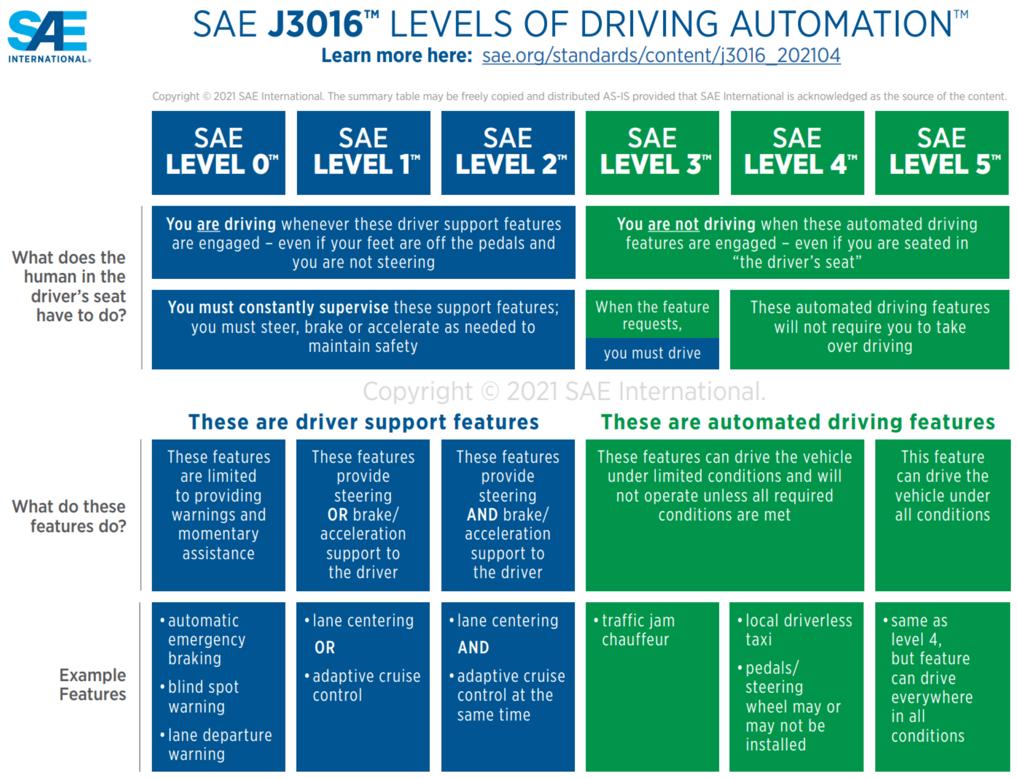
1.1 Background

As the automotive industry continues to advance towards higher level of driving automation, the integration of automated driving technologies becomes a critical research and development focus. Among these, Advanced Driver Assistance Systems (ADAS) play an increasingly important role in improving vehicle safety, comfort, eco-driving performance and automation. ADAS not only addresses real-world safety challenges but also serves as the foundation for higher levels of autonomous driving.

To standardize and classify the levels of driving automation, the Society of Automotive Engineers (SAE) introduced the widely accepted SAE J3016 taxonomy in 2014 and updated it in 2021, which defines six levels of driving automation from Level 0 (No Automation) to Level 5 (Full Automation)[1]. The six levels of automation defined in SAE J3016 are summarized in the figure below, which illustrates the increasing degree of driving automation and decreasing reliance on the human driver.

ADAS functionalities are primarily associated with Level 1 (Driver Assistance) and Level 2 (Partial Driving Automation), where the system provides limited assistance under driver supervision. At Level 1, ADAS functions focus more on single aspect of driving – either longitudinal or lateral control. Level 2 systems combine both, allowing for simultaneous control of speed and steering. However, in both cases or levels, the driver is expected or requested to monitor the environment and maintain full responsibility for current driving tasks.

As different systems and functions are increasingly integrated into production vehicles, ensuring the safety, reliability and regulatory compliance has become a



SAE J3016™ LEVELS OF DRIVING AUTOMATION™

Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			**These are automated driving features**		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.1: SAE Levels of Driving Automation

major concern for the automotive industry. ADAS functions must be evaluated through simulation under diverse conditions before deployment, because improper behaviors, even at low levels of automation, can also lead to dangerous situations. Therefore, a structured approach to function-level validation is essential to support the wider adoption of ADAS and build a solid foundation for the development of higher-level autonomous driving systems.

In this context, virtual testing environments have emerged as a valuable tool for evaluating ADAS behavior under diverse, repeatable and controlled conditions. Simulation platforms such as MATLAB/Simulink, RoadRunner, CarSim, SCANeR Studio enable developers to model vehicle dynamics and control systems, generate diverse scenarios, and evaluate system behaviors without the limitations of physical testing. Among these platforms, the integration of MATLAB/Simulink and RoadRunner provides a convenient, lightweight, and flexible solution that is suitable for early-stage function verification and academic research.

1.2 Problems and Motivations

As interest in automated driving technologies grows, regulatory authorities around the world have increasingly introduced stringent requirements for the design, testing, and validation of Advanced Driver Assistance Systems (ADAS) and higher-level autonomous functions. However, the diversity of regional policies, safety regulations, and infrastructure constraints poses significant challenges to the standardized development and deployment of such systems.

In the European Union, the United Nations Economic Commission for Europe (UNECE) has issued the R157 regulation, which governs the approval of Level 3 Automated Lane Keeping Systems (ALKS)[2]. The regulation requires not only real-world tests but also scenario-based virtual validation, including edge cases that are difficult to replicate on real-world or open roads. Compliance with UNECE R157 thus necessitates simulation frameworks that can accurately recreate or realistically simulate complex traffic environments in a repeatable and traceable manner.

In the United States, the National Highway Traffic Safety Administration (NHTSA) has adopted a more flexible, voluntary safety assessment framework[3]. While NHTSA promotes innovation and public trust through consumer information and federal guidelines, the actual enforcement of testing regulations varies by state. Several jurisdictions may require strict road-testing permits, safety driver protocols and incident reporting for autonomous vehicle trials. Furthermore, due to the lack of unified federal legislation for Level 2 automation, developers often face a fragmented compliance landscape that complicates nationwide deployment and validation.

Notably, NHTSA recognizes the critical role of driver assistance technologies in improving road safety. As stated on its official website, “We now know that driver assistance technologies are the right path toward safer roads. We will work diligently to bring you updated information whenever there are breakthroughs with new driver assistance technologies” (National Highway Traffic Safety Administration, n.d.)[3]. This highlights the agency’s supportive stance toward ADAS development, while also reinforcing the need for robust validation procedures to ensure system safety and public acceptance.

In China, where the development of Intelligent and Connected Vehicles (ICVs) is regarded as a national strategic priority, the Ministry of Industry and Information Technology (MIIT), along with other government bodies, has actively promoted pilot programs and closed test zones for intelligent driving. In 2024, MIIT launched a large-scale “Vehicle-Road-Cloud Integration” pilot program across 20 major cities including Beijing and Shanghai[4]. This initiative aims to establish unified standards, develop intelligent road infrastructure, and support scenario-driven testing environments through cloud-based control platforms. In parallel, China also approved nine major Original Equipment Manufacturers (OEMs), including Changan, BYD, GAC, and SAIC, to participate in national pilot projects for L3-level automated driving access and road testing, marking a significant step toward commercializing higher-level autonomous driving[5].

These real-world limitations — high testing costs, regulatory fragmentation, and restricted access — highlight the need for flexible and reproducible alternatives during early ADAS development. In particular, many Level 1-2 ADAS features, such as ACC, AEB and LKA, rely heavily on control logic and rule-based responses that can be effectively evaluated in structured virtual environments. Simulation-based testing provides a safe, scalable, and repeatable approach for verifying such functionality without the risks or logistical burdens of physical road testing.

The main motivation of this thesis work is to narrow the gap between theoretical control design and functional scenario testing by employing a co-simulation approach that integrates MATLAB/Simulink for control modeling with RoadRunner for the scenario generation. Rather than targeting high-level autonomy or perception-driven behavior, the focus is on function-level validation of ADAS modules under simplified but representative driving situations. By leveraging the accessibility, modularity, and extensibility of these tools, the thesis work aims to support academic research, early-stage development, and educational applications aligned with the SAE Level 1–2 automation scope.

1.3 Thesis Outline

The structure of this thesis is as follows:

- **Chapter 2** reviews the theoretical background and simulation tools relevant to this study. It introduces the core principles of key ADAS functions—such as Adaptive Cruise Control (ACC), Autonomous Emergency Braking (AEB), and Lane Keeping Assist (LKA)—and provides an overview of simulation platforms including MATLAB/Simulink and RoadRunner. Their respective roles in control modeling and scenario generation are discussed in the context of co-simulation.

- **Chapter 3** describes the setup of the simulation framework and the implementation of ADAS functions. It outlines the vehicle dynamics model, the control logic for each ADAS module, and the integration method between

Simulink and RoadRunner. This chapter also introduces a series of simplified yet representative test scenarios designed to evaluate each function individually or in combination.

- **Chapter 4** presents the simulation results across different scenarios. For each scenario—such as straight-line following, sudden stop response, lane departure, or curve tracking—the system's behavior is analyzed in terms of response correctness, rule compliance, and control smoothness. Visual outputs and behavioral evaluations are provided to demonstrate system performance.
- **Chapter 5** concludes the thesis by summarizing the research findings and reflecting on the effectiveness of the proposed co-simulation approach. Limitations of the current implementation are discussed, and possible directions for future work are suggested, including the integration of perception models and expansion to higher-level autonomous driving functions.

Chapter 2

Background and Tools

Before introducing the proposed simulation framework, this chapter provides the theoretical background and technical tools relevant to this thesis. While the goal of this project is not to achieve full autonomous driving, it addresses the functional verification of ADAS modules that form the core of SAE Level 1–2 automation. These systems—such as Adaptive Cruise Control (ACC), Autonomous Emergency Braking (AEB), and Lane Keeping Assist (LKA)—rely heavily on control logic and structured environmental interaction.

The chapter begins with a brief overview of the operational principles and control objectives of the selected ADAS functions. It then introduces the simulation tools used in this study—MATLAB/Simulink for control system modeling, and RoadRunner for generating realistic traffic scenarios. Together, these platforms enable modular and reproducible validation of control-layer behavior. The concepts and tools presented here serve as the foundation for the simulation setup and co-simulation platform described in the following chapter.

2.1 ADAS Overview

While Chapter 1 has introduced the basic concept, importance, and industrial context of Advanced Driver Assistance Systems (ADAS)—including a graphical summary of the SAE automation levels (see Figure 1.1)—this section provides a more detailed technical analysis of ADAS functional architecture, system classification, operational pipelines, and their relationship with simulation-based validation. The aim is to establish a comprehensive theoretical foundation for the subsequent discussion on simulation tools, co-simulation environments, and ADAS controller evaluation.

2.1.1 Functional Scope and Classification of ADAS

ADAS are engineered to augment driver awareness, support vehicle control, and automate specific driving tasks under defined conditions. Their core functions can be grouped as follows:

- **Longitudinal Assistance:**Managing speed and headway, e.g., Adaptive Cruise Control (ACC), Autonomous Emergency Braking (AEB).
- **Lateral Assistance:** Maintaining lane position or assisting in lane changes, e.g., Lane Keeping Assist (LKA), Lane Departure Warning (LDW), Lane Centering.
- **Surround Awareness:** Monitoring blind spots, cross-traffic, or other potential hazards, e.g., Blind Spot Detection (BSD), Rear Cross-Traffic Alert.
- **Information and Compliance:** Identifying traffic signs, navigation aids, or road conditions, e.g., Traffic Sign Recognition (TSR), Traffic Light Detection.

The SAE J3016 standard provides the widely recognized taxonomy for classifying driving automation. As summarized in Figure 1.1 (see Chapter 1), SAE Levels 0–2 define the scope of ADAS, with Level 1 corresponding to single-function driver assistance (either lateral or longitudinal), and Level 2 enabling combined assistance under constant human supervision. Table 2.1 summarizes the main ADAS functionalities and their typical sensor suites[6]:

Function	**Control Objective**	**Key Sensors**
Adaptive Cruise Control (ACC)	Maintain time gap to lead vehicle	Radar, Camera
Lane Keeping Assist (LKA)	Keep vehicle within lane boundaries	Camera
Autonomous Emergency Braking (AEB)	Prevent or mitigate frontal collisions	Radar, Camera, LiDAR
Traffic Sign Recognition (TSR)	Detect and display road signs	Camera
Blind Spot Detection (BSD)	Warn of vehicles in adjacent lanes	Radar, Ultrasonic

Table 2.1: Common ADAS Functions and Typical Sensor Suites

2.1.2 Simulation Pipeline Structure

A typical ADAS simulation workflow is generally structured as a closed-loop pipeline that connects perception, control module, and vehicle dynamics. This setup supports systematic testing of control strategies in a virtual environment and is compatible with a range of simulation tools such as Simulink and RoadRunner.

(1) Perception

- **Scenario Configuration:** Virtual environments, including roads, lane geometry, and dynamic actors such as other vehicles or pedestrians, are defined using scenario editors such as RoadRunner and imported into modeling environments such as Simulink.
- **Virtual Sensor Models:** Simulated sensor models such as radar, camera and LiDAR capture environmental information, such as lane positions and nearby objects, based on the predefined scenario setup.
- **Detection and Processing:** Dedicated components process sensor outputs to extract lane geometry, identify surrounding vehicles, and estimate their motion characteristics.

(2) Control Module

- **Decision Logic:** The control module begins with decision-making logic that interprets perception outputs and determines the appropriate behavioral response. This includes identifying the most relevant target, such as the lead vehicle, computing the relative distance d_{rel} and velocity v_{rel}, and determining whether to accelerate, decelerate, or maintain speed. A common metric used in Adaptive Cruise Control (ACC) is the time gap t_{gap}, defined as:

$$t_{\text{gap}} = \frac{d_{\text{rel}}}{v_{\text{ego}}}, \tag{2.1}$$

  where v_{ego} is the ego vehicle's speed.

- **Controller Design:** Based on decision outcomes, the controller generates actuation commands for longitudinal and lateral movement. These are often implemented using Proportional–Integral–Derivative (PID) control or Model Predictive Control (MPC)[7]. A typical longitudinal control law may take the form:

$$a_{\text{ego}} = K_p(d_{\text{rel}} - d_{\text{des}}) + K_v(v_{\text{lead}} - v_{\text{ego}}), \tag{2.2}$$

where d_{des} is the desired following distance, v_{lead} is the velocity of the lead vehicle, and K_p, K_v are controller gains.

- **Mode Management:** Control logic may also include supervisory mechanisms for handling operational mode transitions, such as switching from regular following to emergency braking. These transitions are typically triggered by threshold conditions derived from sensor inputs and vehicle dynamics.

(3) Vehicle Dynamics

- **Dynamics Modeling:** Once the control module generates control outputs such as acceleration and steering angle, the vehicle dynamics model simulates the corresponding motion of the vehicle. Many simulation setups adopt simplified models (e.g., the bicycle model) for real-time performance and explainability. These models approximate both longitudinal and lateral behavior while capturing essential physical constraints[8][9][10].
- **State Update:** The vehicle's motion is updated based on its current state and control inputs. In the longitudinal direction, the update equations are typically given by:

  $$\dot{v}_{\text{ego}} = a_{\text{ego}}, \quad \dot{x}_{\text{ego}} = v_{\text{ego}} \tag{2.3}$$

  where v_{ego} is the ego vehicle's velocity, a_{ego} is the applied acceleration, and x_{ego} is the vehicle's longitudinal position.

  Lateral motion is often modeled using yaw angle and curvature-based updates in more advanced configurations.
- **Feedback Integration:** The updated vehicle states—position, velocity, orientation—are continuously fed back into the simulation environment to maintain a closed-loop system. This enables dynamic interaction with surrounding traffic and road features, allowing scenario evolution to be sensitive to vehicle behavior. Such feedback is critical for evaluating the robustness of control logic under diverse and changing driving conditions.

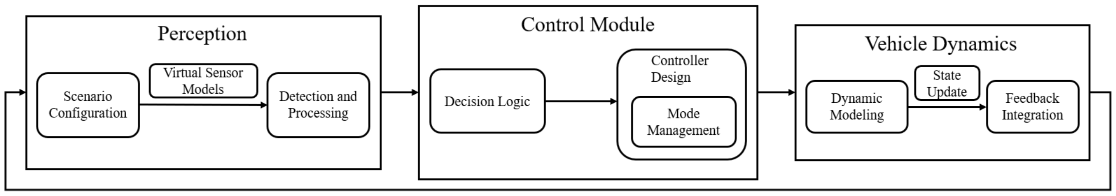


Figure 2.1: Overview of Simulation Pipeline Structure

2.2 Simulation Tools

As Advanced Driver Assistance Systems (ADAS) continue to evolve, virtual simulation has become a critical part of the development and validation pipeline. Compared to physical prototyping and on-road testing, simulation tools provide a safe, repeatable, and cost-efficient environment for testing complex driving scenarios, edge cases, and control logic in early development stages.

This section introduces three software tools used in this thesis: MATLAB, Simulink, and RoadRunner. Each plays an important role in the simulation and validation of ADAS functions.

2.2.1 MATLAB

MATLAB is a high-level programming language and an integrated development environment (IDE). It plays an important role in the simulation framework for ADAS development, particularly through its Driving Scenario Designer app and the Automated Driving Toolbox. These tools provide a dedicated environment for building, visualizing, and simulating driving scenarios, as well as for configuring virtual sensors and actors for perception and decision-making studies.

(a) Driving Scenario Designer app

The Driving Scenario Designer app, included in MATLAB's Automated Driving Toolbox, provides a graphical interface to build and simulate driving scenarios. It is commonly used in the early stages of ADAS development, where traffic environments and sensor setups need to be quickly defined and tested. Its main functions include:

- **Interactive Scenario Creation:** Users can design multilane roads, intersections, and actor trajectories using drag-and-drop operations. The app uses a grid layout and supports intersections with traffic lights.
- **Sensor Configuration:** The ego vehicle can be equipped with virtual sensors such as camera, radar, lidar, ultrasonic, and inertial sensors. These provide simulated outputs for detection, localization, and control testing. The app can generate MATLAB code or Simulink models of sensors for further simulations and testing.
- **Map and Geometry Import:** Road data can be imported from OpenDRIVE (v1.4–1.6), OpenStreetMap, and HERE HD Live Map, allowing realistic or standards-compliant layouts to be used in simulation.

- **Scenario Export and Integration:** Scenarios can be exported to OpenSCENARIO, RoadRunner HD Map, or automatically translated into MATLAB scripts and Simulink models for further development and testing.

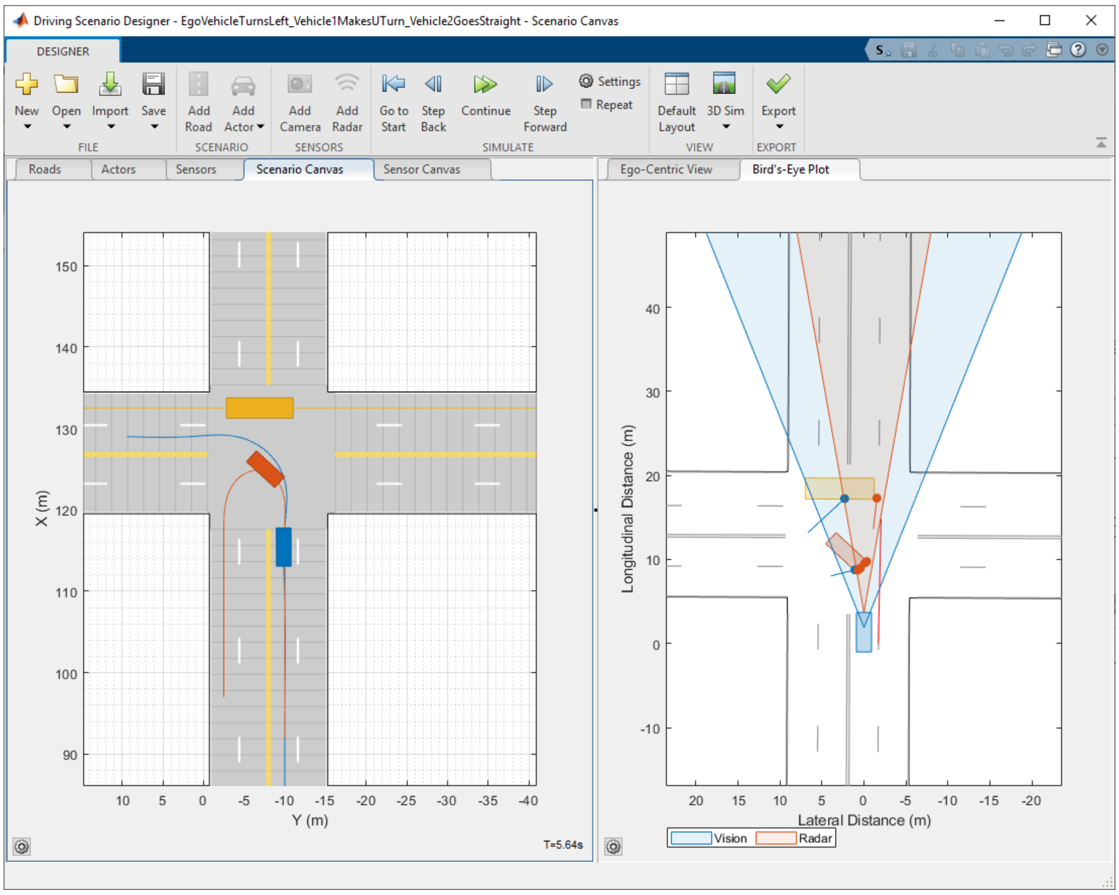


Figure 2.2: Driving Scenario Designer app of MATLAB

(b) Automated Driving Toolbox

The Automated Driving Toolbox provides functions for modeling and testing perception, planning, and control components used in advanced driver assistance systems. It supports simulations of camera, radar, and lidar sensors, and includes tools for visualizing sensor outputs, tracking objects, and analyzing dynamic scenes. Functions are available for tasks such as lane detection, object tracking, and trajectory planning.

Driving scenarios can be generated through scripting and actor behaviors can be programmatically adjusted under different test conditions. Road network data can be imported from formats such as OpenDRIVE and HERE HD Live Map, allowing scenes to reflect real-world layouts.

The toolbox also supports integration with Simulink for closed-loop simulation and enables C/C++ code generation for hardware-in-the-loop testing.

By combining scenario modeling, sensor simulation, and code-level integration, the toolbox can work as a link between scene-level testing and the implementation of control algorithms.

2.2.2 Simulink

Simulink is a graphical modeling and simulation platform. In ADAS function development and simulation, Simulink is commonly used to build controller models, vehicle dynamics models, and to perform closed-loop simulations together with other software tools with other software tools. Its main roles include:

- **Controller modeling and Strategy Implementation:** The control architecture is built using block diagrams, enabling modular design of acceleration commands, time gap policy execution, and driver response logic. This approach allows the controller to be tested independently from the plant model and facilitates systematic tuning of decision layers.
- **Vehicle Dynamics Modeling:** A high-fidelity vehicle dynamics model is implemented within Simulink, incorporating components such as electric drivetrain behavior, resistive forces, and tire-road interaction. These elements provide realistic feedback to the controller, ensuring that performance evaluation—especially in terms of energy consumption—is grounded in physical constraints.
- **Integration of Sensor Models and Perception Feedback:** Simulated sensors, including camera, radar, and lidar models, are connected to the control loop to emulate perception of the lead vehicle's motion. These inputs supply real-time distance, speed, and relative position data, which the control algorithm uses to adjust vehicle behavior in response to traffic flow.
- **Scenario Co-simulation with External Platforms:** Simulink supports co-simulation with various external platforms that provide traffic scenarios or perception environments. In this thesis, Simulink interfaces with RoadRunner through dedicated blocks for importing scenario data and exporting ego vehicle data. During simulation, other actors are governed by predefined behaviors, while the ego vehicle is controlled by the Simulink model in a closed-loop setup. This co-simulation architecture can also be extended to other tools designed for more complex, high-fidelity, and physically realistic simulation environments.

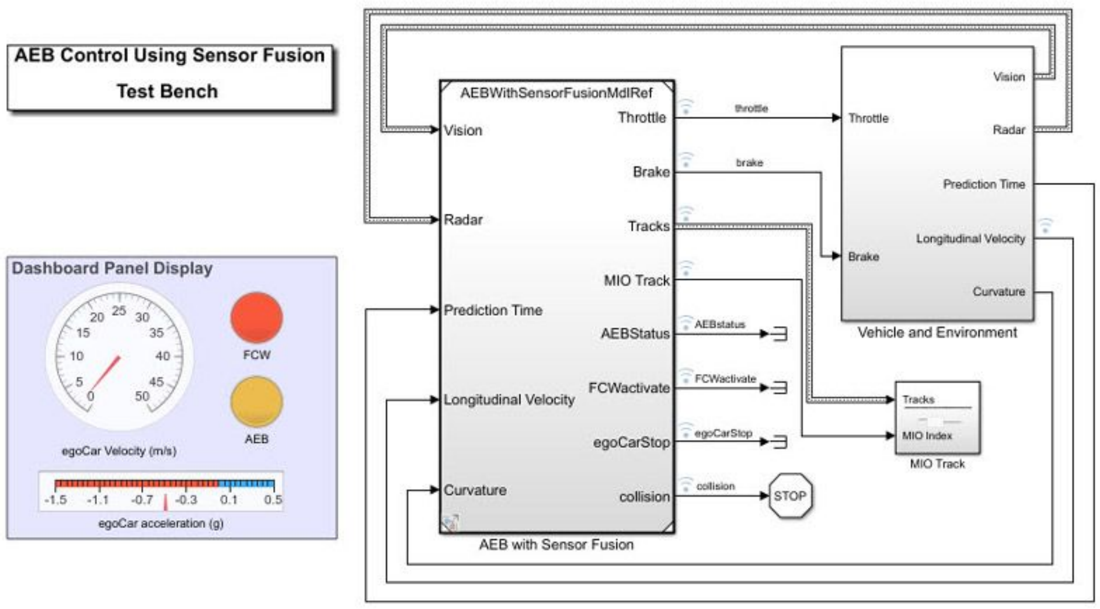


Figure 2.3: Automatic Emergency Braking (AEB) Test Bench with Sensor Fusion in Simulink

2.2.3 RoadRunner

(a) RoadRunner Core Capabilities

RoadRunner is a 3D scene design tool used to create driving environments for the simulation and testing of automated driving systems. It allows for the construction of detailed road networks, customization of traffic infrastructure, and integration with various geographic and simulation data. Its core capabilities include:

- **3D Road and Environment Modeling:** Users can create complex layouts involving intersections, tunnels, bridges, ramps, and elevation changes. The tool supports the addition of guardrails, road surface damage, vegetation, buildings, and other 3D assets to reflect realistic environments.

- **Traffic Signs and Signal Control:** RoadRunner enables the placement of region-specific or customized road signs and markings. It also provides tools for configuring traffic light logic, including phase timing and transitions at intersections.

- **Map Data and GIS Integration:** Aerial imagery, LiDAR point clouds, and digital elevation models can be imported to accurately replicate real-world

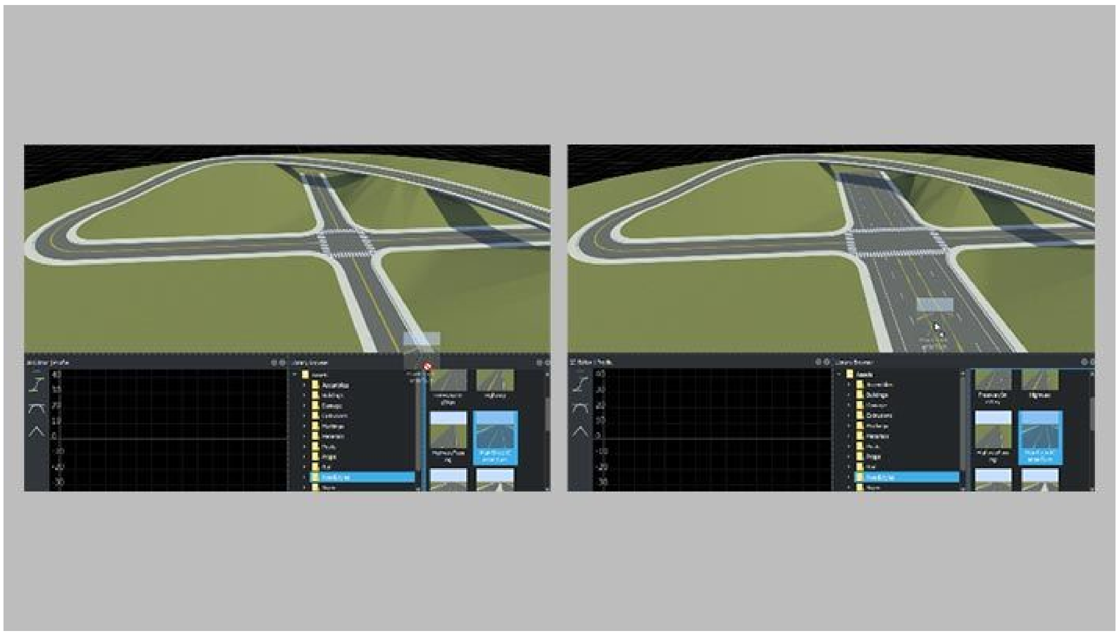


Figure 2.4: Road and 3D Scene Modeling in RoadRunner

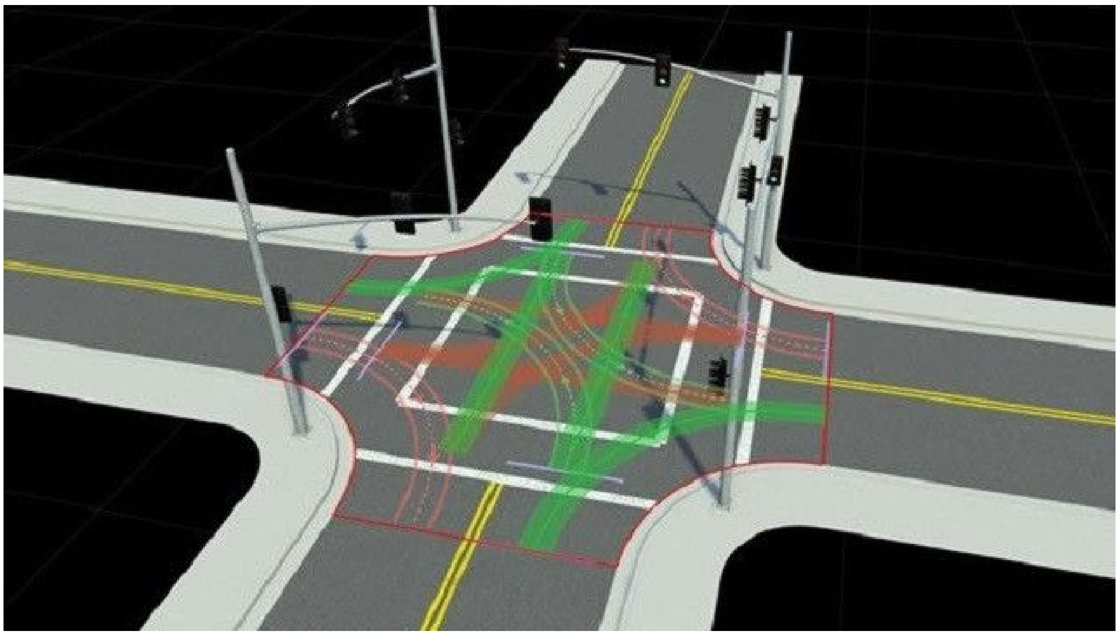


Figure 2.5: Functional Road Networks and Traffic Signals Design in RoadRunner

road networks. Support is provided for OpenDRIVE-based road layouts and external SD/HD maps, including those from OpenStreetMap and commercial

Application Programming Interfaces (APIs).

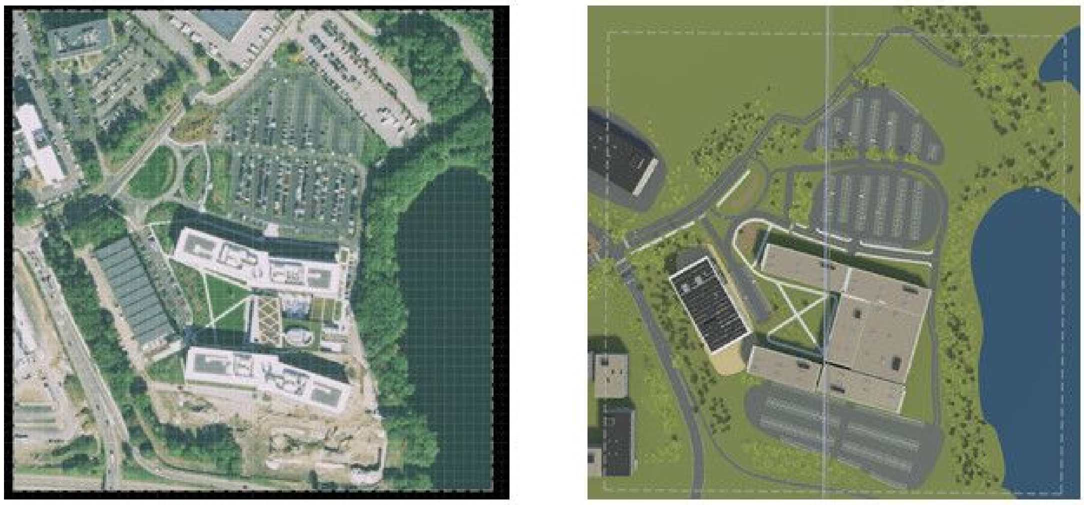


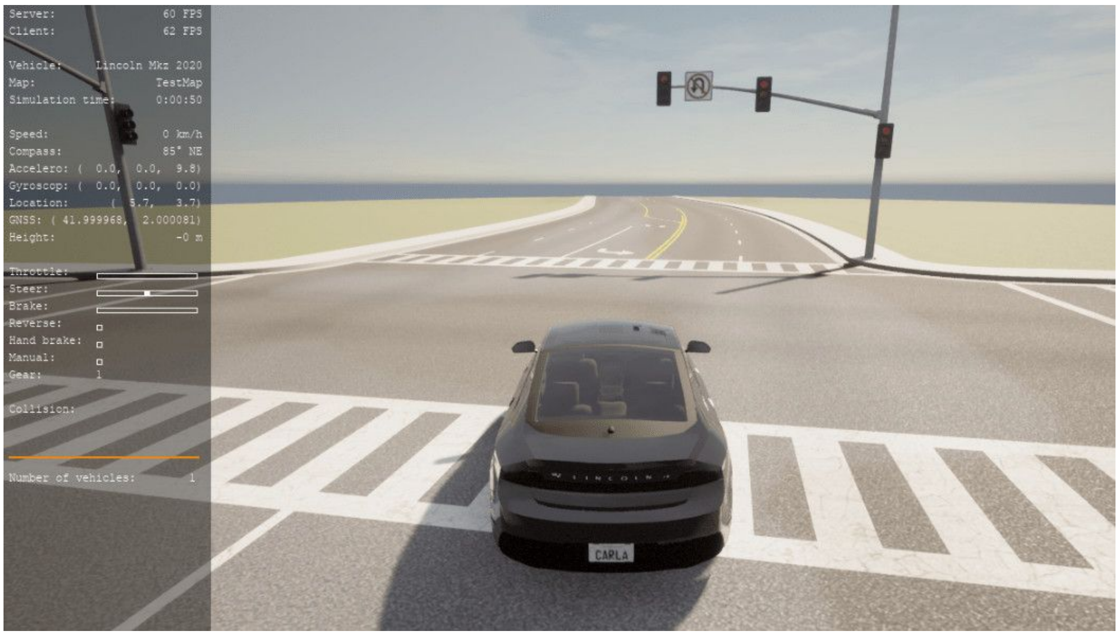
Figure 2.6: Map Import in RoadRunner

- **Cross-Platform Scene Export:** RoadRunner scenes can be exported to industry-standard formats (OpenDRIVE®, FBX®, glTF™, USD, etc.), allowing compatibility with platforms such as CARLA, dSPACE ASM, IPG CarMaker, and Unreal Engine.

- **Programmatic API and Automation:** Scenes can be generated or modified programmatically using MATLAB functions or a gRPC interface. This facilitates the batch creation of test environments and automated scenario variation.

- **Asset Library and Custom Models:** With the RoadRunner Asset Library, users can populate scenes using a broad set of built-in and user-defined 3D assets, including barriers, signs, signal poles, and roadside objects.

- **Surface Feature Modeling:** The tool supports OpenCRG formats for high-fidelity road surface modeling. Surface anomalies such as potholes, rumble strips, and speed bumps can be added to evaluate vehicle response in realistic terrain.



```
Server:                 60 FPS
Client:                 62 FPS

Vehicle:       Lincoln Mkz 2020
Map:                   TestMap
Simulation time:       0:00:50

Speed:                  0 km/h
Compass:                85° NE
Accelero: (  0.0,   0.0,   9.8)
Gyroscop: (  0.0,   0.0,   0.0)
Location:        (   5.7,   3.7)
GNSS: ( 41.999968,   2.000081)
Height:                   -0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:         1

Collision:

Number of vehicles:          1
```

Figure 2.7: RoadRunner Scenes Export to Other Simulators

(b) RoadRunner Product Family

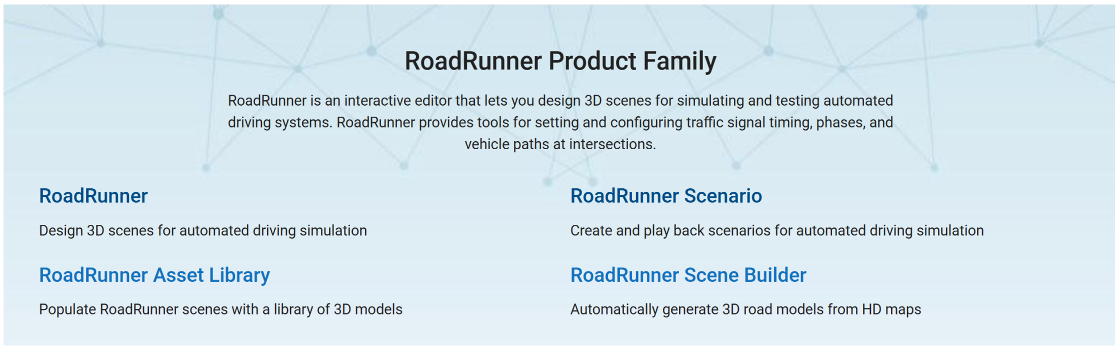
RoadRunner is not just a single application, but a group of related tools designed to support different stages of scene creation and simulation. Each product serves a specific purpose and can be used individually or together, depending on the needs of the simulation setup.

- **RoadRunner:** The main application used to build 3D road networks, edit lane structures, place traffic signs and signals, and add visual elements such as buildings, sidewalks, or roadside objects.

- **RoadRunner Scenario:** This tool is used to create and play back driving scenarios within RoadRunner scenes. It supports the definition of vehicle paths, actor behavior, and traffic control logic for simulation testing.

- **RoadRunner Asset Library:** This tool provides a library of 3D models—such as signs, barriers, and poles—that can be used to populate RoadRunner scenes with region-specific and realistic roadside elements.

- **RoadRunner Scene Builder:** This tool is used to automatically generate 3D road scenes from HD map data, streamlining the process of building accurate road geometry and network structure.


RoadRunner Product Family

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems. RoadRunner provides tools for setting and configuring traffic signal timing, phases, and vehicle paths at intersections.

RoadRunner

Design 3D scenes for automated driving simulation

RoadRunner Scenario

Create and play back scenarios for automated driving simulation

RoadRunner Asset Library

Populate RoadRunner scenes with a library of 3D models

RoadRunner Scene Builder

Automatically generate 3D road models from HD maps

Figure 2.8: RoadRunner Product Family

2.2.4 Summary

In summary, MATLAB, Simulink, and RoadRunner together can form a flexible and powerful toolchain for simulating and validating advanced driving systems.

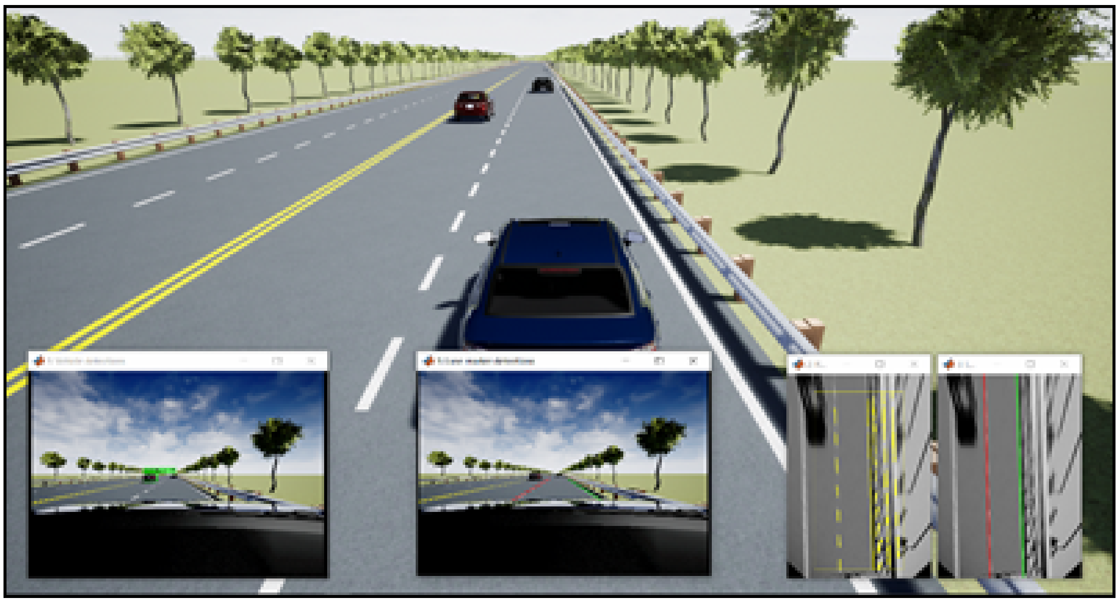


Figure 2.9: Automated driving simulation within a RoadRunner scene

Chapter 3

Scene Construction and Simulation Configuration

Following the introduction of the simulation pipeline structure and software tools in Chapter 2, this chapter focuses on the configuration process of a co-simulation environment for scenario-based testing. The objective is not to develop new algorithms or models, but to establish a functional simulation setup that integrates key components—such as virtual environments, sensors, vehicle dynamics, and control logic—into a unified workflow.

The configuration is centered around RoadRunner, which provides the test scenario, including the virtual environment, road layouts, path definitions, and the behavior or action configuration of actors. Simulink functions as the ego vehicle's behavior, where sensor models, controllers, and vehicle dynamics blocks are connected to form a closed-loop system.

The controllers and vehicle dynamics model used in the simulation are from the Simulink library or are prebuilt modules that have been integrated into the system for validation purposes. Similarly, sensor blocks such as radar and camera modules are from the Automated Driving Toolbox or the Simulink library. The focus here is on how these components are organized or customized, and how they can work together to support our testing under various scenarios.

Other parts, such as coordinate system transformation and metrics logging, are also included, as they are essential for generating simulation outputs and evaluating performance.

3.1 Scene Construction

The first step in scenario-based simulation is to establish the scene that serves as the foundation for the dynamic testing. Most scenarios are built on a straight or curved

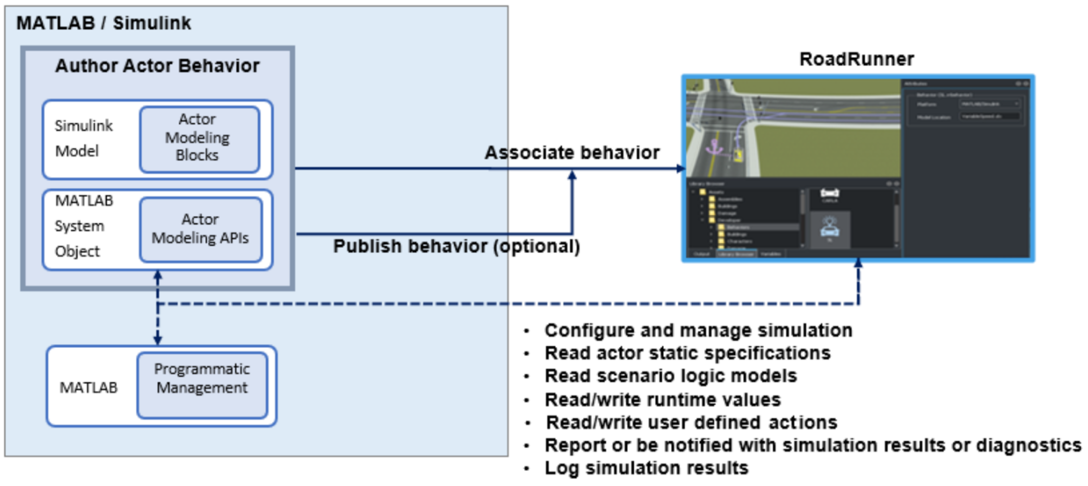

MATLAB / Simulink
Author Actor Behavior
Simulink Model
Actor Modeling Blocks
MATLAB System Object
Actor Modeling APIs
MATLAB
Programmatic Management
Associate behavior
Publish behavior (optional)
RoadRunner
Configure and manage simulation
Read actor static specifications
Read scenario logic models
Read/write runtime values
Read/write user defined actions
Report or be notified with simulation results or diagnostics
Log simulation results

Figure 3.1: Overview of Co-Simulation Workflow

multi-lane highway, created using the basic road editing tools in RoadRunner or the official example provided by RoadRunner. The layout typically features one lane or two lanes in each direction, with standard lane widths and markings for clarity.

The road network for each simulation is constructed using the basic and advanced road editing tools available in RoadRunner. Typically, I start by laying out straight road segments and multi-lane highways, but the software also supports more complex features like curves, roundabouts, and intersections if needed. Road geometry can be adjusted by setting curvature, slope and cross-sectional details such as banking and crowning to better reflect real-world road conditions.

Figure 3.2: RoadRunner Toolbar

Lane attributes—including the number of lanes, their widths, and markings—are easily modified with dedicated lane tools. Detailed surfaces can be created using standard lane markings, stencils, and road paint. For scenarios that require realistic or site-specific environments, RoadRunner also allows importing map data in formats such as OpenDRIVE, enabling rapid generation of road networks based on actual GPS or HD/SD map sources. This flexibility makes it possible to adapt the test scene to a wide range of research or validation requirements.

Although the scenes used in this study are kept deliberately simple, RoadRunner provides a range of features that make it straightforward to expand the simulation

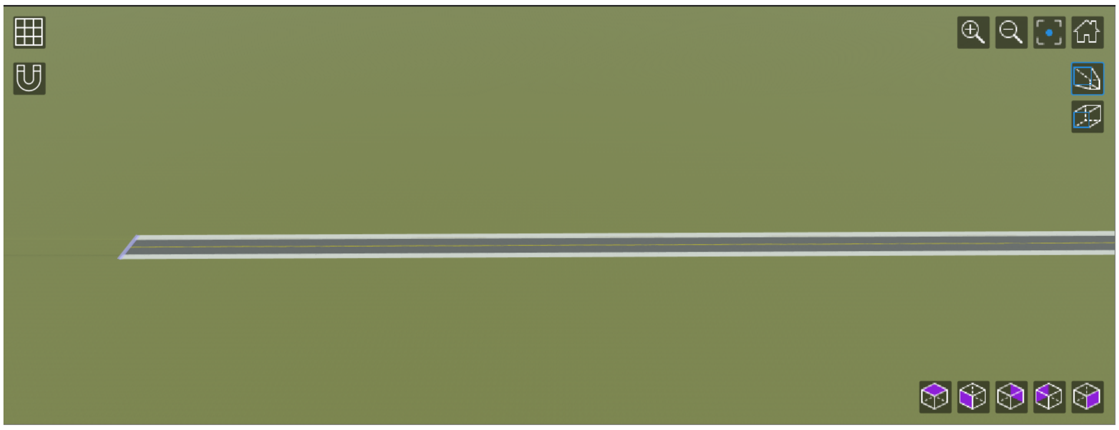


Figure 3.3: Straight Road Created in RoadRunner

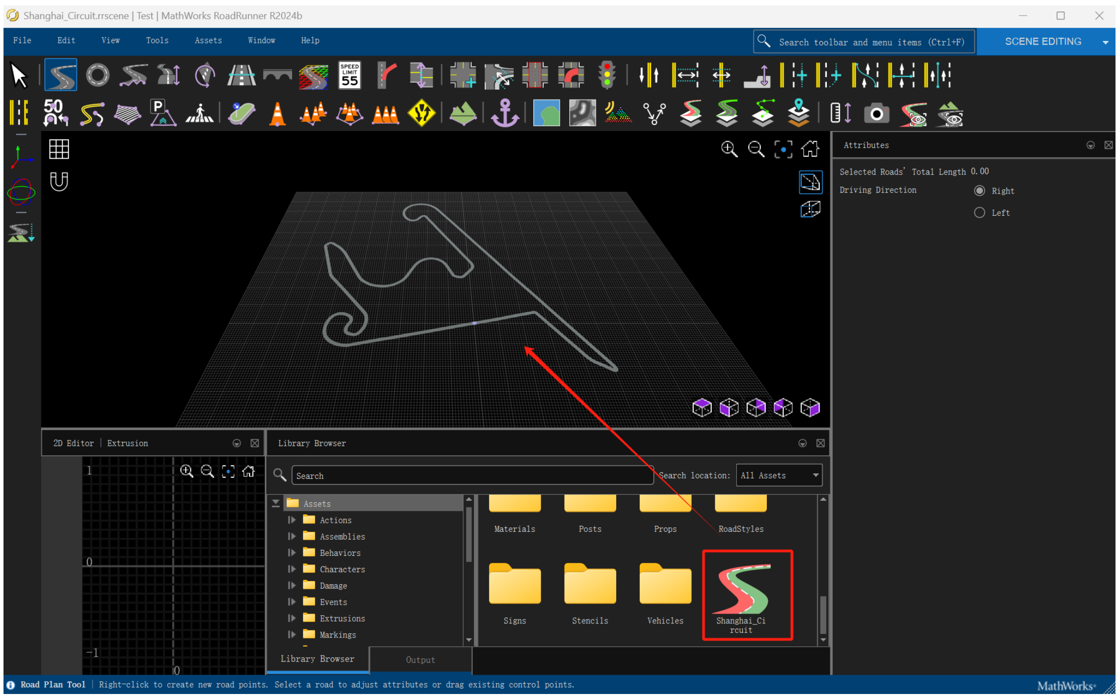


Figure 3.4: Insertion of Shanghai International Circuit OpenDRIVE Data

environment in future work. For example, surface and terrain properties can be adjusted to create custom driving surfaces or to introduce roughness and elevation changes for more advanced dynamics testing. This makes it possible to recreate urban areas, rural roads, or any other real environment with a high degree of fidelity.

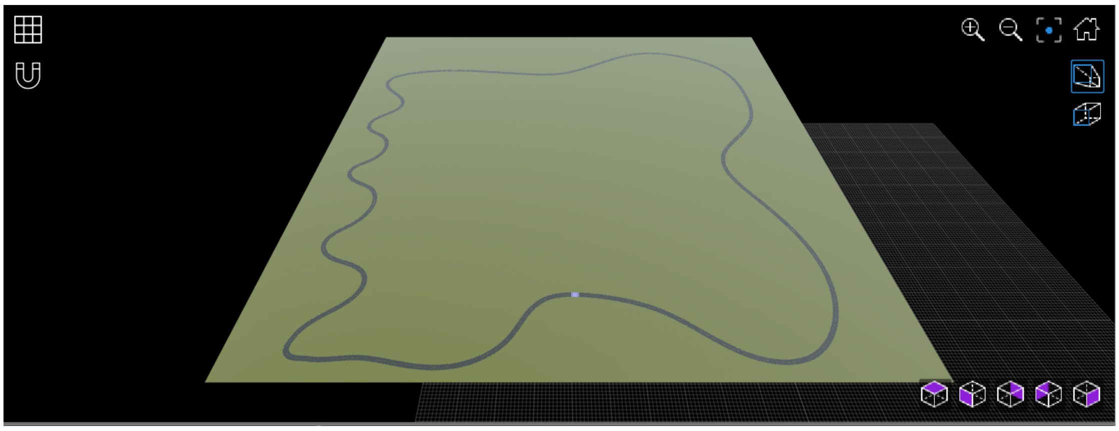


Figure 3.5: Main Scene Used for Simulation Testing: Curved Road

3.2 Scenario Configuration

With the static road environment in place, the next step is to configure each scenario with the required vehicles and behaviors. In RoadRunner, this involves placing the necessary actors, defining their motion logic, and adjusting relevant parameters to reflect the intended test conditions[11].

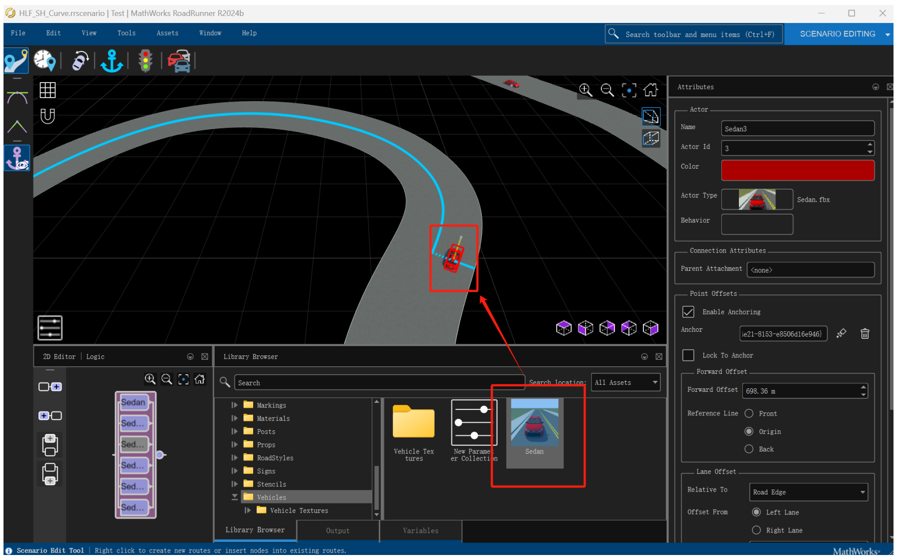
3.2.1 Scenario and Actor Setup in RoadRunner

- **Actor Placement:** Actors are placed directly onto the road network using the scenario editor. Each actor (such as a lead vehicle or cut-in vehicle) is assigned a specific starting lane and initial position.

- **Vehicle Customization:** For clarity during testing, each actor is given a unique name (e.g., “Lead Vehicle”, “Cut-In Vehicle”) and, if needed, its visual model or color can be selected from the RoadRunner asset library or the actor panel, and even can be imported from external resources to distinguish between vehicle types in multi-actor scenarios.

- **Behavior and Path Definition:** Actor’s behavior can be defined by assigning a sequence of actions. In most test cases, this means setting the initial speed, using “follow lane” as the default movement, and then adding action types such as lane changes, stops, or accelerations at specific times or positions during the simulation. The Path can be created and customized by selecting the actor and right-clicking on the road to set waypoints.

- **Simulation Properties Editing:** Simulation properties such as simulation time, step size, fail conditions and other conditions can be edited or customized in the attributes panel or the simulation panel.



Attributes

Actor
Name: Sedan3
Actor Id: 3
Color: 🟥
Actor Type: Sedan.fbx
Behavior:

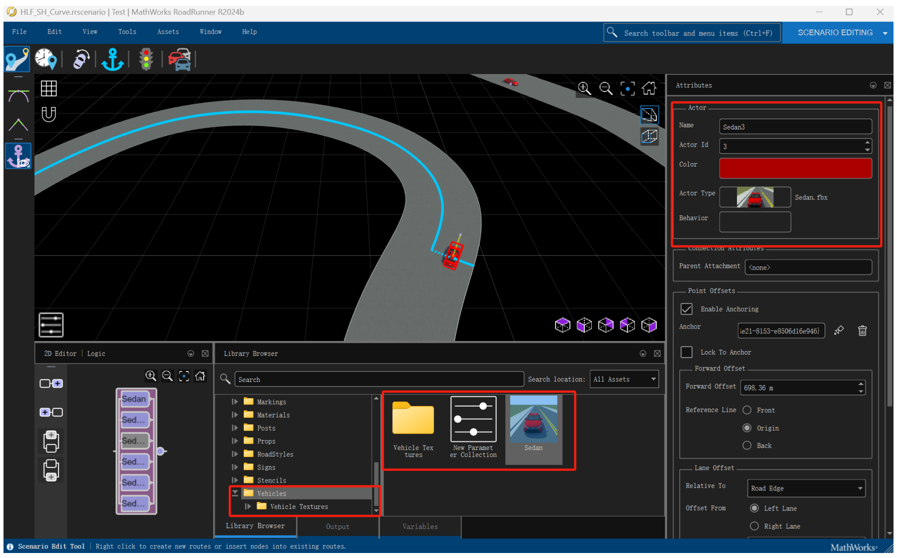
Connection Attributes
Parent Attachment: <none>

Point Offsets
[✓] Enable Anchoring
Anchor: {e21-8153-e8506d16e946}
[ ] Lock To Anchor

Forward Offset
Forward Offset: 698.36 m
Reference Line: Front [✓ Origin] Back

Lane Offset
Relative To: Road Edge
Offset From: [✓ Left Lane] Right Lane

Figure 3.6: Actor Vehicle Placement



Attributes

Actor
Name: Sedan3
Actor Id: 3
Color: 🟥
Actor Type: Sedan.fbx
Behavior:

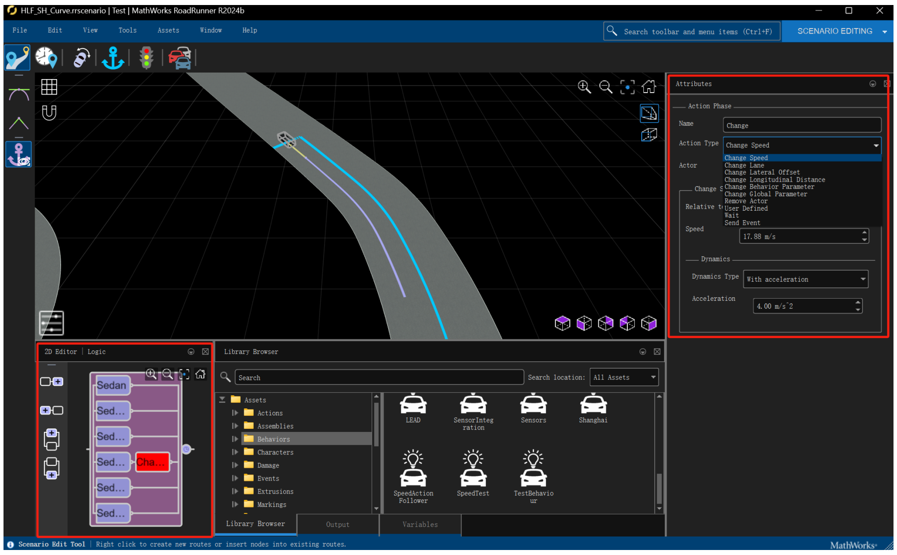
Connection Attributes
Parent Attachment: <none>

Point Offsets
[✓] Enable Anchoring
Anchor: {e21-8153-e8506d16e946}
[ ] Lock To Anchor

Forward Offset
Forward Offset: 698.36 m
Reference Line: Front [✓ Origin] Back

Lane Offset
Relative To: Road Edge
Offset From: [✓ Left Lane] Right Lane

Figure 3.7: Actor Vehicle Customization



Attributes

Action Phase

Name: Change

Action Type: Change Speed

Change Speed
Change Lane
Change Lateral Offset
Change Longitudinal Distance
Change Behavior Parameter
Change Global Parameter
Remove Actor
User Defined
Wait
Send Event

Actor

Relative to

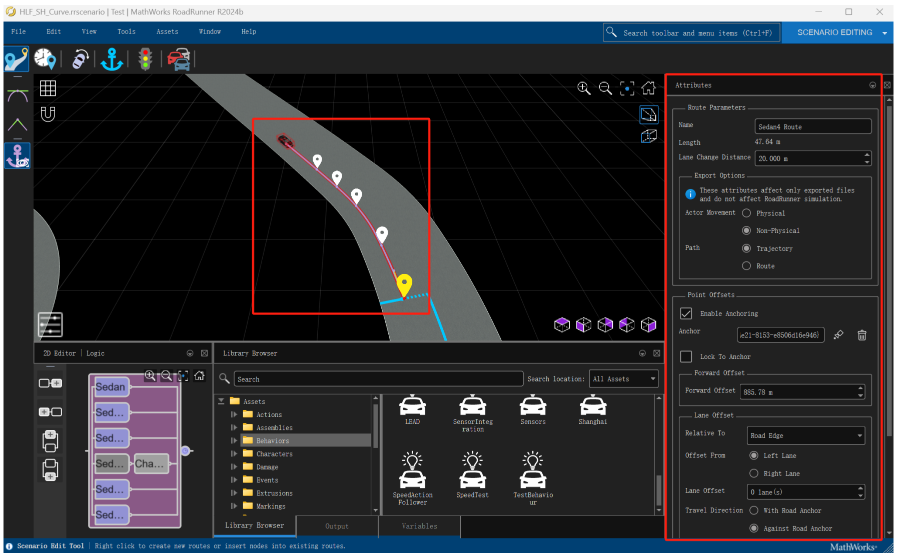
Speed: 17.88 m/s

Dynamics

Dynamics Type: With acceleration

Acceleration: 4.00 m/s^2

Figure 3.8: Actor Vehicle Behavior Definition



Attributes

Route Parameters

Name: Sedan4 Route

Length: 47.64 m

Lane Change Distance: 20.000 m

Export Options

These attributes affect only exported files and do not affect RoadRunner simulation.

Actor Movement: Physical [✓ Non-Physical]

Path: [✓ Trajectory] Route

Point Offsets

[✓] Enable Anchoring

Anchor: e21-8153-e8506d16e946}

[ ] Lock To Anchor

Forward Offset

Forward Offset: 885.78 m

Lane Offset

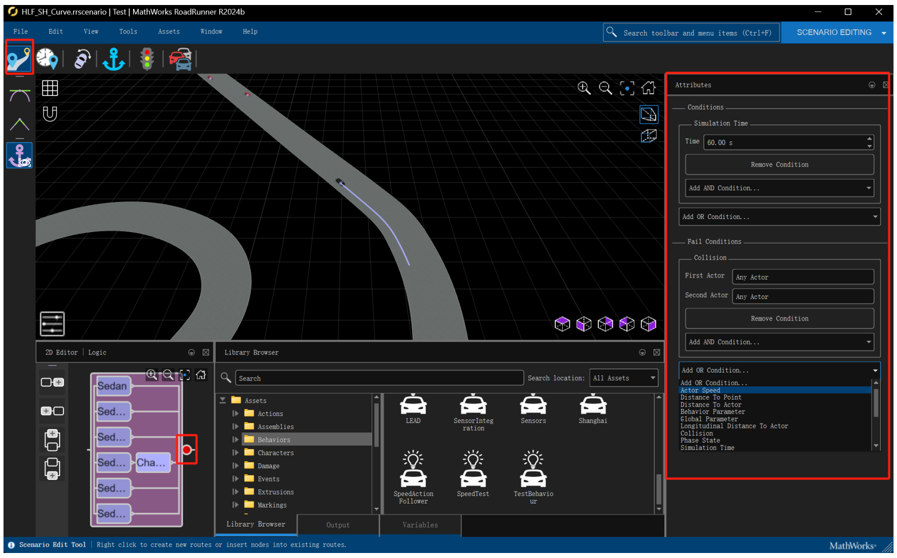
Relative To: Road Edge

Offset From: [✓ Left Lane] Right Lane

Lane Offset: 0 lane(s)

Travel Direction: With Road Anchor [✓ Against Road Anchor]

Figure 3.9: Actor Vehicle Path Definition


HLF_SH_Curve.rrscenario | Test | MathWorks RoadRunner R2024b
File Edit View Tools Assets Window Help
SCENARIO EDITING
Attributes
Conditions
Simulation Time
Remove Condition
Add AND Condition...
Add OR Condition...
Fail Conditions
Collision
Library Browser
Scenario Edit Tool | Right click to create new routes or insert nodes into existing routes.

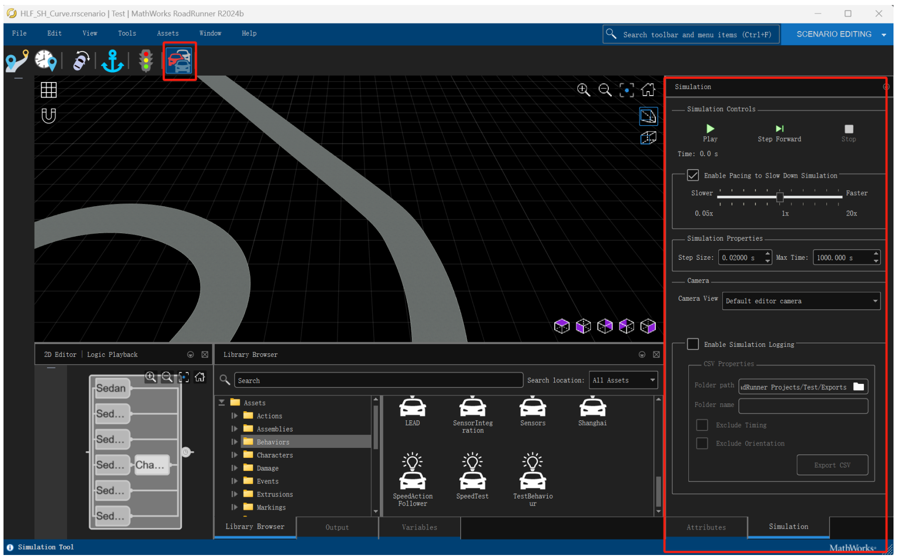
Time 60.00 s

First Actor Any Actor

Second Actor Any Actor

Add OR Condition...: Actor Speed, Distance To Point, Distance To Actor, Behavior Parameter, Global Parameter, Longitudinal Distance To Actor, Collision, Phase State, Simulation Time

Figure 3.10: Simulation Properties Editing(1)



Time: 0.0 s

[✓] Enable Pacing to Slow Down Simulation

Step Size: 0.02000 s Max Time: 1000.000 s

Camera View Default editor camera

[ ] Enable Simulation Logging

Folder path ...dRunner Projects/Test/Exports

Folder name

[ ] Exclude Timing

[ ] Exclude Orientation

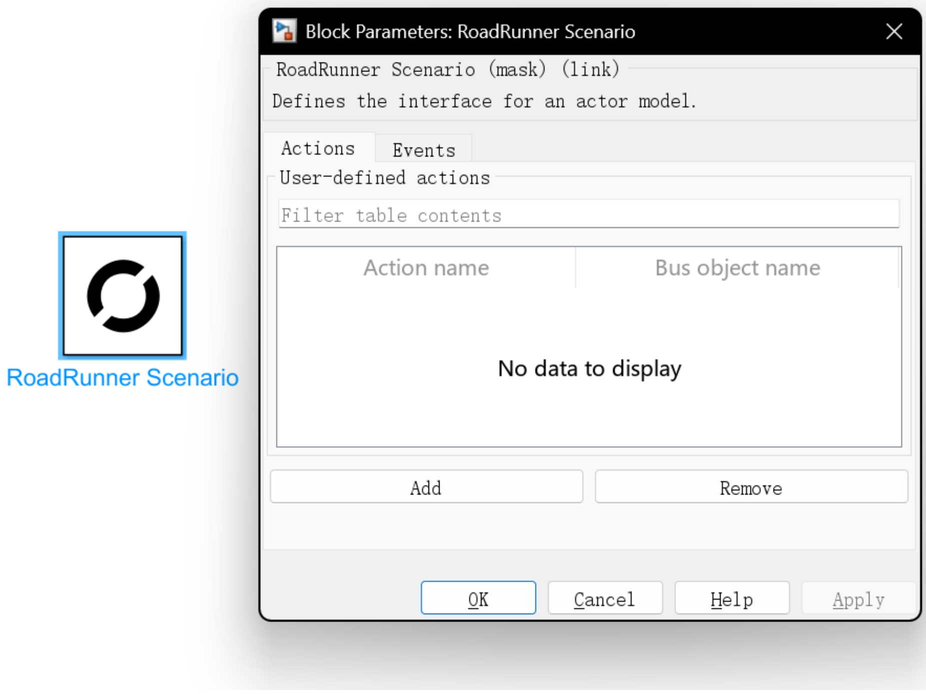
Figure 3.11: Simulation Properties Editing(2)

3.2.2 Ego Vehicle Behavior and Simulink Integration

The ego vehicle's behavior in each scenario is managed externally using Simulink, rather than being assigned a fixed action sequence in RoadRunner. During scenario setup, the ego vehicle is placed in the desired lane and position, but its subsequent motion is determined by the real-time output of the connected Simulink model.

To establish this link, we have to configure both Simulink and RoadRunner. In Simulink, we have to use three blocks in the Automated Driving Toolbox:

- **RoadRunner Scenario block:** The RoadRunner Scenario block defines the interface of a Simulink model and must be present at the root level of the model. In the Block Parameters dialog box, inputs are grouped under Actions, Events, and Configuration.


Block Parameters: RoadRunner Scenario

RoadRunner Scenario (mask) (link)

Defines the interface for an actor model.

Actions | Events

User-defined actions

Filter table contents

Action name	Bus object name

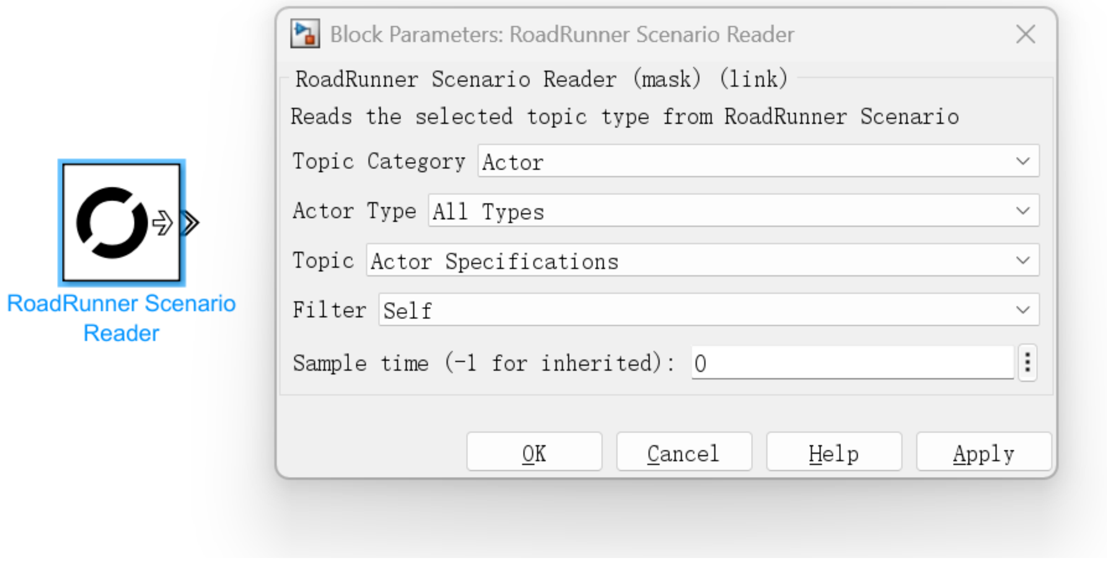
No data to display

Figure 3.12: RoadRunner Scenario block

- **RoadRunner Scenario Reader block:** The RoadRunner Scenario Reader block is used to extract different types of scenario data during simulation. By adjusting the Topic Category parameter, it is possible to access information related to actors, actions, sensors, or custom events as needed for the test.

For example, when configured to the "Actor" topic, the block provides details such as actor specifications and real-time pose. If set to "Action," it outputs information on state changes like speed or lane shifts for each actor. Data is read one time step behind the simulation, and the block can be filtered to focus on a specific actor within the Simulink model.

When the topic is set to "Sensor," the block returns sensor-specific information, such as target positions or lane boundaries, based on the sensor ID assigned in the scenario. Selecting the "Event" topic allows the block to deliver outputs related to user-defined events specified by name.


RoadRunner Scenario Reader
Block Parameters: RoadRunner Scenario Reader
OK
Cancel
Help
Apply

RoadRunner Scenario Reader (mask) (link)
Reads the selected topic type from RoadRunner Scenario

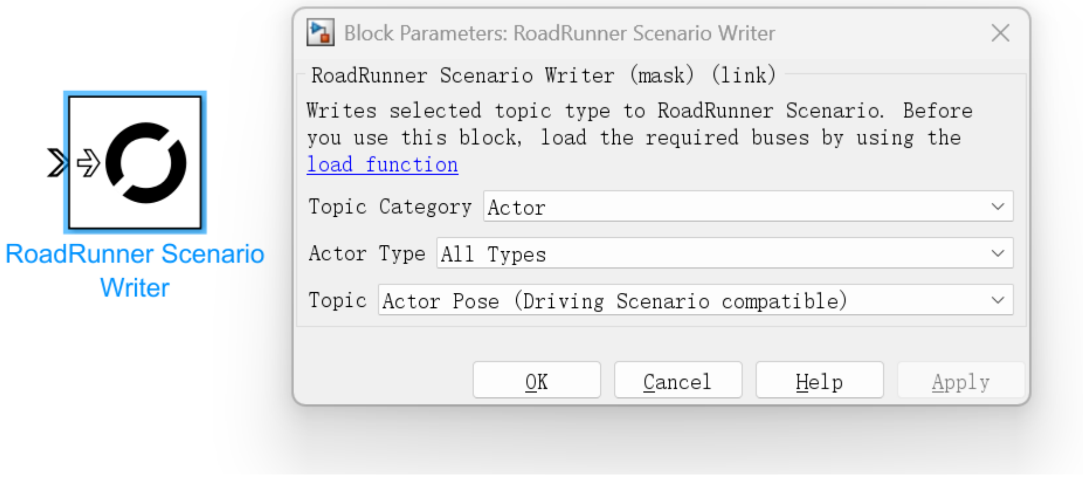
Topic Category: Actor
Actor Type: All Types
Topic: Actor Specifications
Filter: Self
Sample time (-1 for inherited): 0

Figure 3.13: RoadRunner Scenario Reader block

- **RoadRunner Scenario Writer block:** The RoadRunner Scenario Writer block is used to send dynamic information—such as actor states, events, or diagnostics—from Simulink back into a running RoadRunner scenario. Typically, this involves passing real-time data generated by a Simulink behavior model, packaged as Simulink messages.

  By setting the Topic Category to "Actor," the block can update attributes like actor ID, velocity, and angular velocity within the scenario. The Writer block operates using bus objects, which define the structure and data types of the information being sent. To ensure compatibility, the required bus types must first be loaded into the MATLAB workspace, and a Bus Creator block is used to build the corresponding signals in the Simulink model.

When controlling a group of actors, the block supports different usage patterns: you can use a single Writer block and combine outputs with a Message Merge block, or assign separate Writer blocks to each actor, specifying their ActorID as needed. This setup enables flexible, step-by-step control over the behavior of any child actor in the scenario, all from within the Simulink environment.


RoadRunner Scenario Writer
Block Parameters: RoadRunner Scenario Writer
OK
Cancel
Help
Apply

RoadRunner Scenario Writer (mask) (link)

Writes selected topic type to RoadRunner Scenario. Before you use this block, load the required buses by using the load function

Topic Category: Actor

Actor Type: All Types

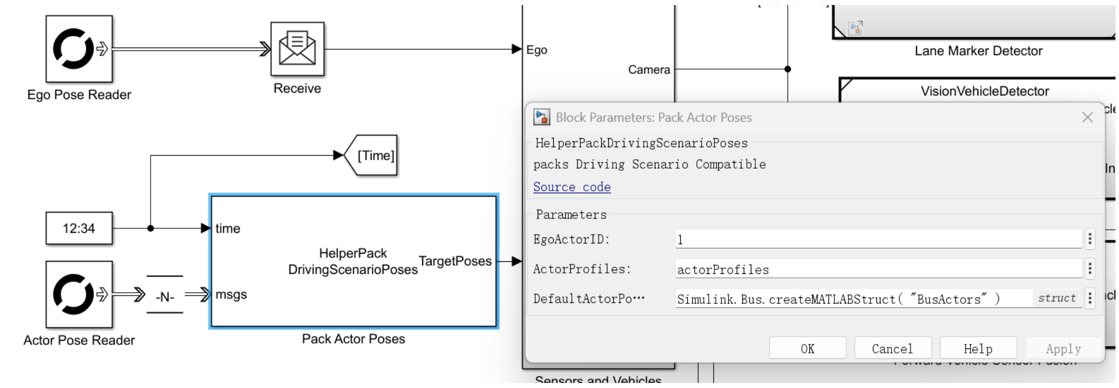
Topic: Actor Pose (Driving Scenario compatible)

Figure 3.14: RoadRunner Scenario Writer block

3.2.3 Ego and Actor Pose

In the integrated simulation framework, the poses of the ego vehicle and other actors are exchanged between RoadRunner and Simulink at each time step to enable accurate closed-loop control. The key data flow starts with the RoadRunner Scenario Reader block with Topic 'Actor Pose (Driving Scenario Compatible)', which receives current positions and states of all vehicles from the RoadRunner scenario.

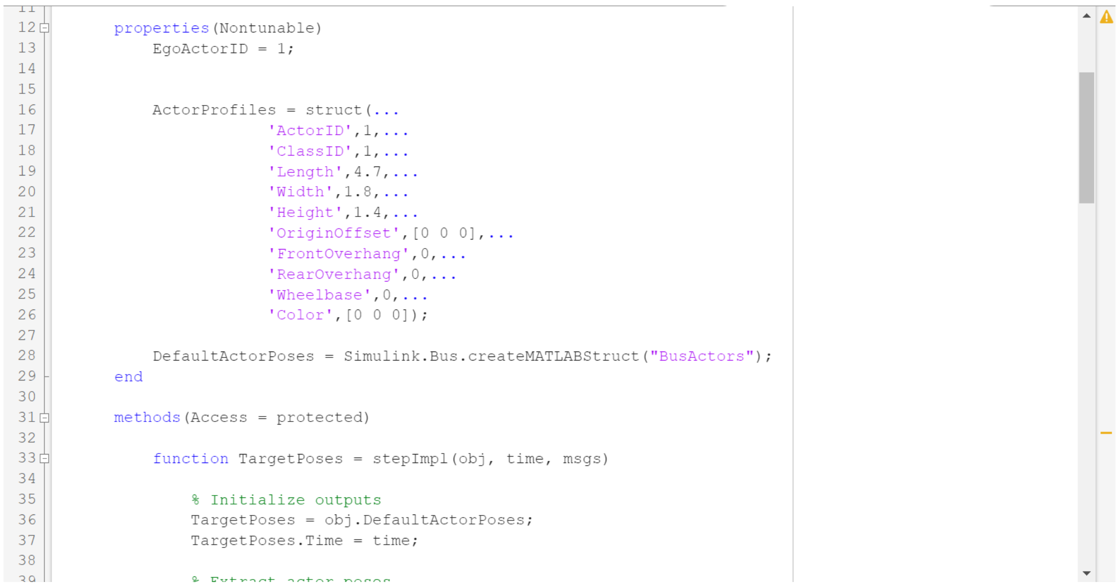
In practice, a user-defined helper function block is used to process actor pose data from RoadRunner. This block takes in the raw messages, extracts the necessary fields—such as position, velocity, and orientation—for each relevant actor, and organizes them into a structured output bus. The resulting TargetPoses signal is then fed into the rest of the Simulink model, supporting downstream modules like perception modeling or controller feedback. This setup makes it easy to adjust to different scenario configurations and ensures that pose data stays consistent throughout the simulation workflow.



Block Parameters: Pack Actor Poses

```
HelperPackDrivingScenarioPoses
packs Driving Scenario Compatible
Source code
Parameters
EgoActorID:        1
ActorProfiles:     actorProfiles
DefaultActorPo…    Simulink.Bus.createMATLABStruct( "BusActors" )   struct
```

Figure 3.15: Actor Pose Packaging in Simulink



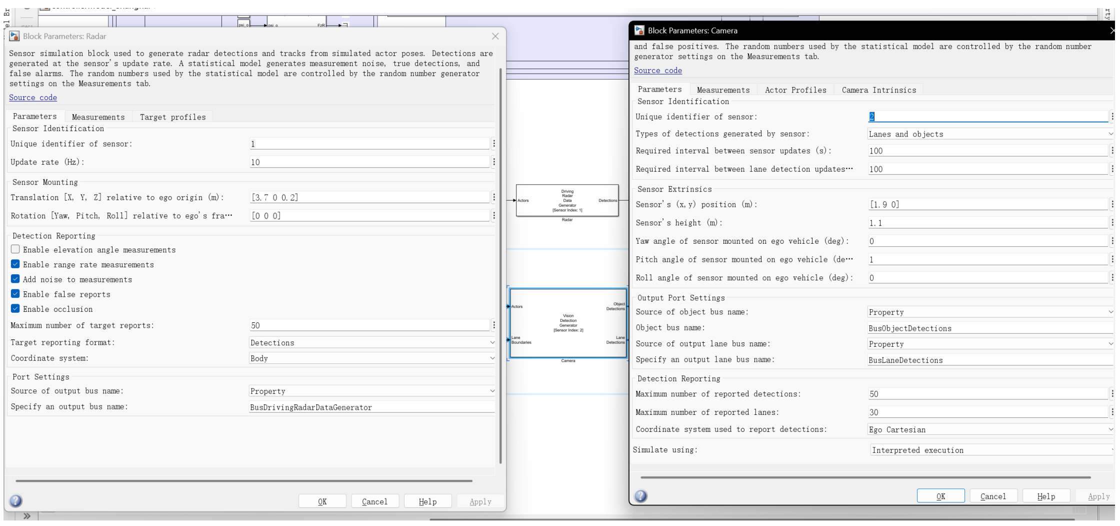
```
    properties(Nontunable)
        EgoActorID = 1;


        ActorProfiles = struct(...
                    'ActorID',1,...
                    'ClassID',1,...
                    'Length',4.7,...
                    'Width',1.8,...
                    'Height',1.4,...
                    'OriginOffset',[0 0 0],...
                    'FrontOverhang',0,...
                    'RearOverhang',0,...
                    'Wheelbase',0,...
                    'Color',[0 0 0]);

        DefaultActorPoses = Simulink.Bus.createMATLABStruct("BusActors");
    end

    methods(Access = protected)

        function TargetPoses = stepImpl(obj, time, msgs)

            % Initialize outputs
            TargetPoses = obj.DefaultActorPoses;
            TargetPoses.Time = time;

            % Extract actor poses
```

Figure 3.16: Actor Properties in Customized Helper Function Script

3.2.4 Sensor Models

Sensor modeling in this project is handled entirely within the Automated Driving Toolbox, with multiple options available depending on the level of fidelity and the requirements of each test.

- **Sensor Models in the Simulink Library:** For many scenarios, sensors such as radar and camera are added directly from the Simulink library. These blocks provide configurable models that simulate typical sensor outputs—like detections, lane boundaries, and object tracks—based on the actor and environment data coming from the scenario. This approach is straightforward and computationally efficient, making it well-suited for rapid prototyping and controller development.



Block Parameters: Radar

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

Source code

Parameters | Measurements | Target profiles

Sensor Identification

Unique identifier of sensor:	1
Update rate (Hz):	10

Sensor Mounting

Translation [X, Y, Z] relative to ego origin (m):	[3.7 0 0.2]
Rotation [Yaw, Pitch, Roll] relative to ego's fra…	[0 0 0]

Detection Reporting

- ☐ Enable elevation angle measurements
- ✓ Enable range rate measurements
- ✓ Add noise to measurements
- ✓ Enable false reports
- ✓ Enable occlusion

Maximum number of target reports:	50
Target reporting format:	Detections
Coordinate system:	Body

Port Settings

Source of output bus name:	Property
Specify an output bus name:	BusDrivingRadarDataGenerator

Block Parameters: Camera

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 Intrinsics

Sensor Identification

Unique identifier of sensor:	2
Types of detections generated by sensor:	Lanes and objects
Required interval between sensor updates (s):	100
Required interval between lane detection updates…	100

Sensor Extrinsics

Sensor's (x, y) position (m):	[1.9 0]
Sensor's height (m):	1.1
Yaw angle of sensor mounted on ego vehicle (deg):	0
Pitch angle of sensor mounted on ego vehicle (de…	1
Roll angle of sensor mounted on ego vehicle (deg):	0

Output Port Settings

Source of object bus name:	Property
Object bus name:	BusObjectDetections
Source of output lane bus name:	Property
Specify an output lane bus name:	BusLaneDetections

Detection Reporting

Maximum number of reported detections:	50
Maximum number of reported lanes:	30
Coordinate system used to report detections:	Ego Cartesian
Simulate using:	Interpreted execution

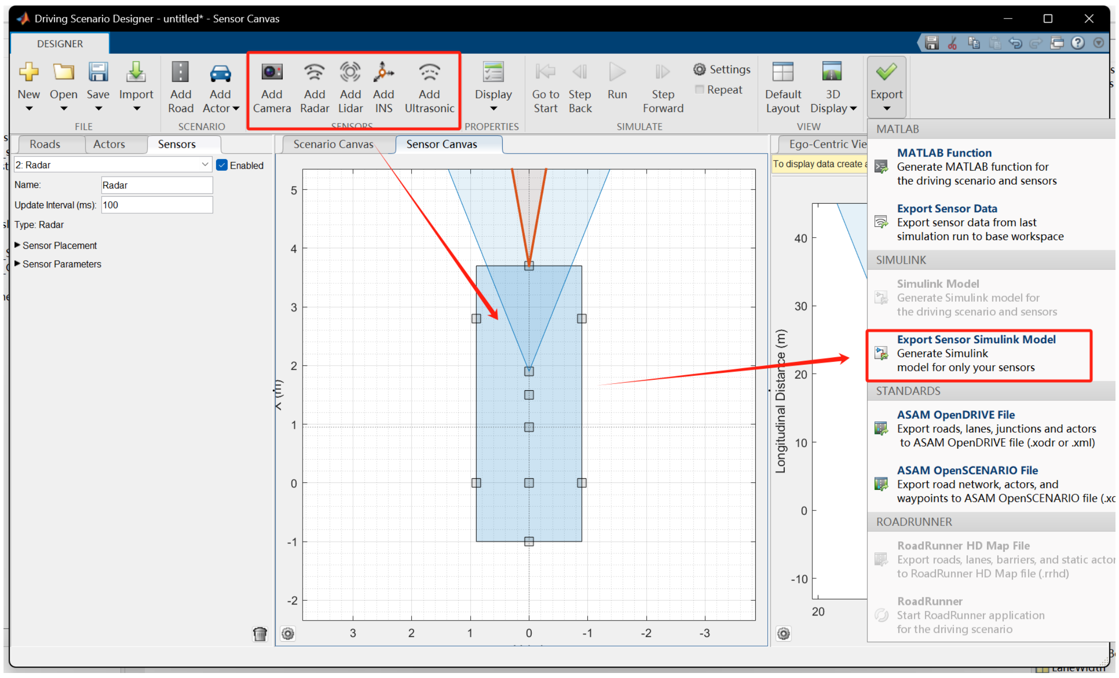
Figure 3.17: Radar and Camera Models in the Simulink Library

- **Driving Scenario Designer Generation:** When a more tailored setup is needed, the Driving Scenario Designer app can be used to lay out scenes and generate matching sensor models for Simulink. This workflow supports quick configuration and direct export of sensor blocks that match the test case setup, including customized field of view, range, resolution, and mounting position. It is helpful when working with repeatable 2D scenarios or batch-generating models for a large number of test cases.

- **3D-Simulation Sensor Models in Simulink:** For more detailed and realistic simulations, the simulation can also use 3D sensor models—such as the Simulation 3D Camera or Simulation 3D Radar—linked to vehicles rendered in the Unreal Engine environment. These blocks not only generate realistic images and detections, but also support the integration of ground truth, sensor noise, and even complex occlusion or lighting effects. This setup is more involved, but necessary for sensor fusion development and evaluating algorithms under more realistic perception conditions.

In practice, the choice of sensor model depends on the simulation objectives. All options are compatible with the co-simulation between Simulink and RoadRunner, and sensor data from any of these models can be routed directly to the sensor fusion algorithm or block for perception modeling.

Since the main focus of this thesis is on function testing and validation rather than on the development of perception algorithms, all perception modeling in this thesis is implemented using the standard sensor models and built-in blocks provided from the official reference case.



2: Radar

Name: Radar

Update Interval (ms): 100

Type: Radar

MATLAB Function
Generate MATLAB function for the driving scenario and sensors

Export Sensor Data
Export sensor data from last simulation run to base workspace

Simulink Model
Generate Simulink model for the driving scenario and sensors

Export Sensor Simulink Model
Generate Simulink model for only your sensors

ASAM OpenDRIVE File
Export roads, lanes, junctions and actors to ASAM OpenDRIVE file (.xodr or .xml)

ASAM OpenSCENARIO File
Export road network, actors, and waypoints to ASAM OpenSCENARIO file (.xc

RoadRunner HD Map File
Export roads, lanes, barriers, and static actor to RoadRunner HD Map file (.rrhd)

RoadRunner
Start RoadRunner application for the driving scenario

Figure 3.18: Sensor Models Generated by the Driving Scenario Designer app

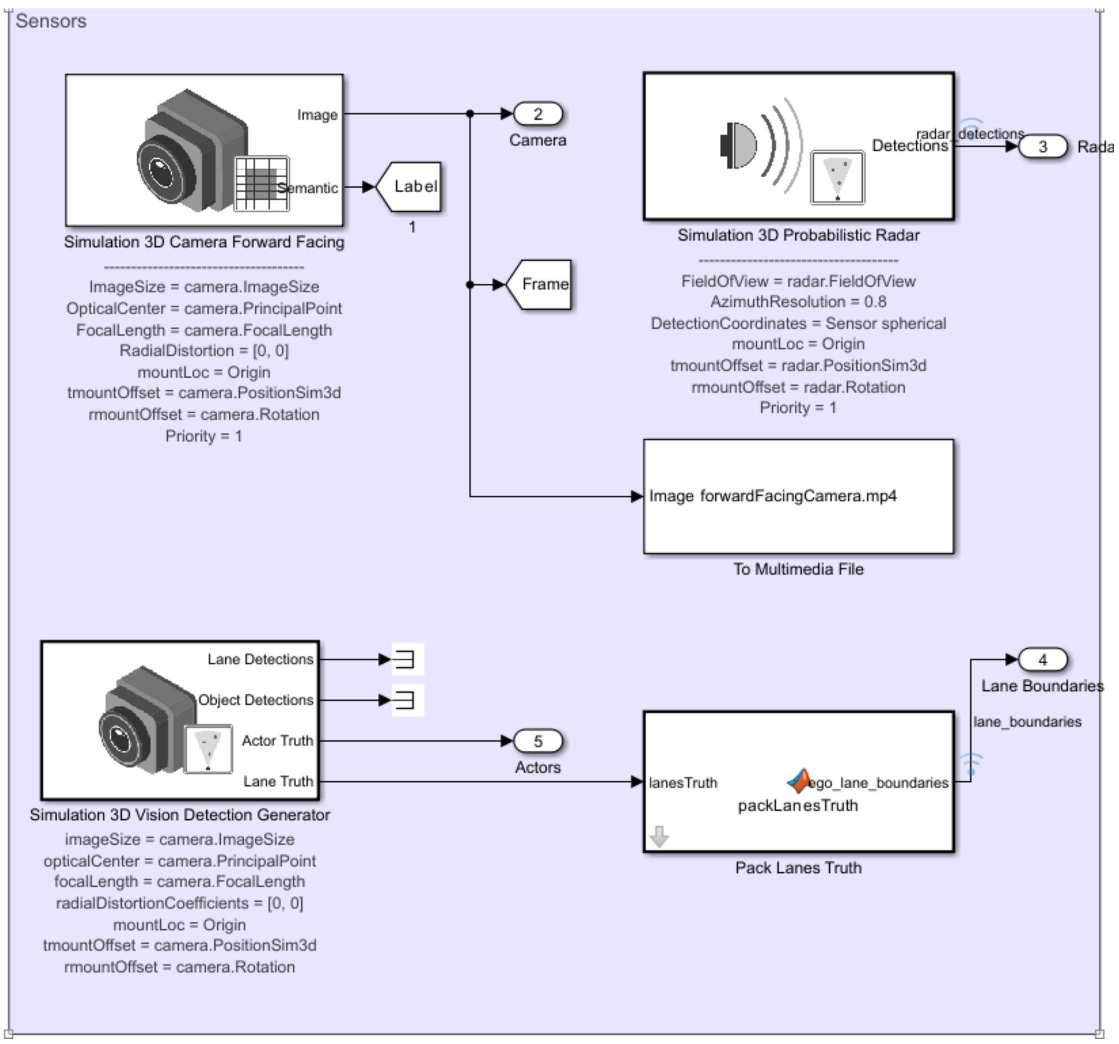

Sensors
Image
Semantic
Simulation 3D Camera Forward Facing
ImageSize = camera.ImageSize
OpticalCenter = camera.PrincipalPoint
FocalLength = camera.FocalLength
RadialDistortion = [0, 0]
mountLoc = Origin
tmountOffset = camera.PositionSim3d
rmountOffset = camera.Rotation
Priority = 1
2
Camera
Label
1
Frame
Detections
radar_detections
3
Rada
Simulation 3D Probabilistic Radar
FieldOfView = radar.FieldOfView
AzimuthResolution = 0.8
DetectionCoordinates = Sensor spherical
mountLoc = Origin
tmountOffset = radar.PositionSim3d
rmountOffset = radar.Rotation
Priority = 1
Image forwardFacingCamera.mp4
To Multimedia File
Lane Detections
Object Detections
Actor Truth
Lane Truth
Simulation 3D Vision Detection Generator
imageSize = camera.ImageSize
opticalCenter = camera.PrincipalPoint
focalLength = camera.FocalLength
radialDistortionCoefficients = [0, 0]
mountLoc = Origin
tmountOffset = camera.PositionSim3d
rmountOffset = camera.Rotation
5
Actors
lanesTruth
ego_lane_boundaries
packLanesTruth
Pack Lanes Truth
4
Lane Boundaries
lane_boundaries

Figure 3.19: 3D-Simulation Sensor Models in Simulink

3.2.5 Controller and Vehicle Dynamics Integration

The starting point of this thesis is the validation of ADAS functions in simulation, with a primary focus on testing and evaluating adaptive cruise control (ACC) strategies under various scenarios. Two types of controllers are integrated into the simulation framework: one based on a model predictive control (MPC) approach, and the other implementing a classical ACC logic[12].

(1) **Control Logic Layer**

Before reaching the controller, outputs from the perception layer are processed by a control logic module. This module identifies the lead vehicle and extracts relevant variables, including relative distance, relative speed, and lane information. The logic is not custom-developed in this study, but is directly utilized from the official platform to relay necessary parameters to the controller modules.

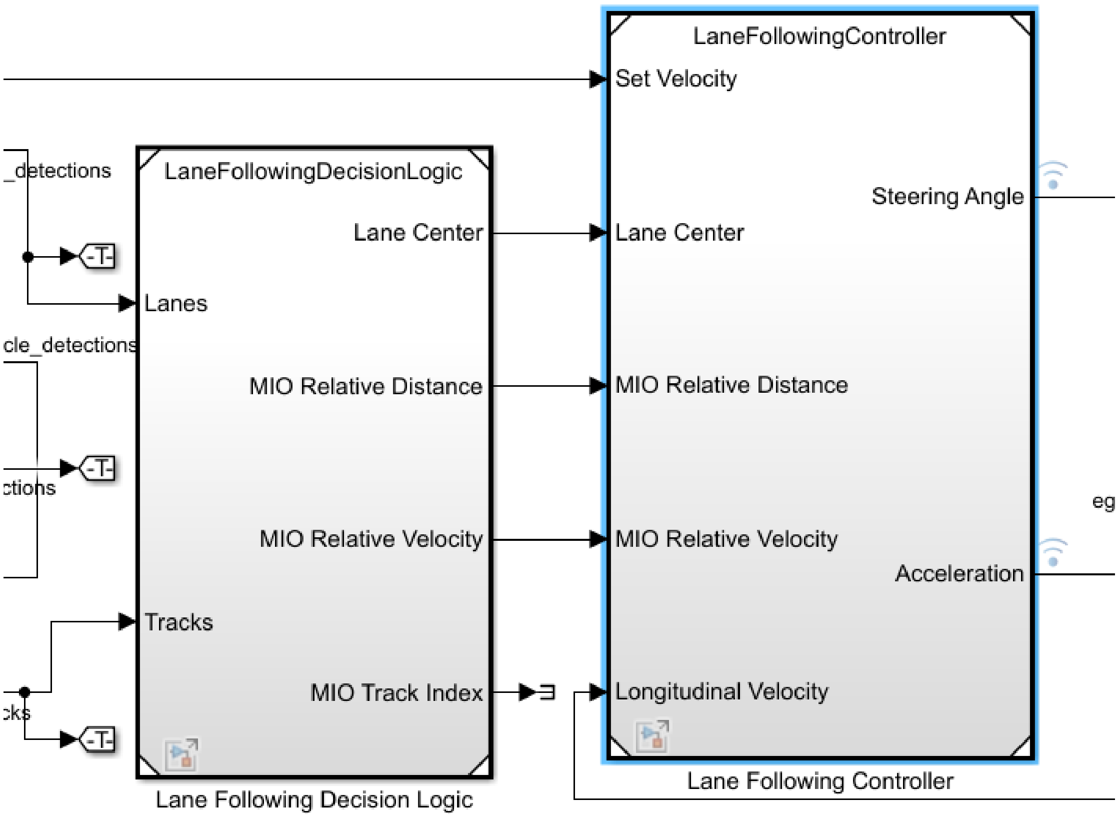


Figure 3.20: Controller Configuration(1)

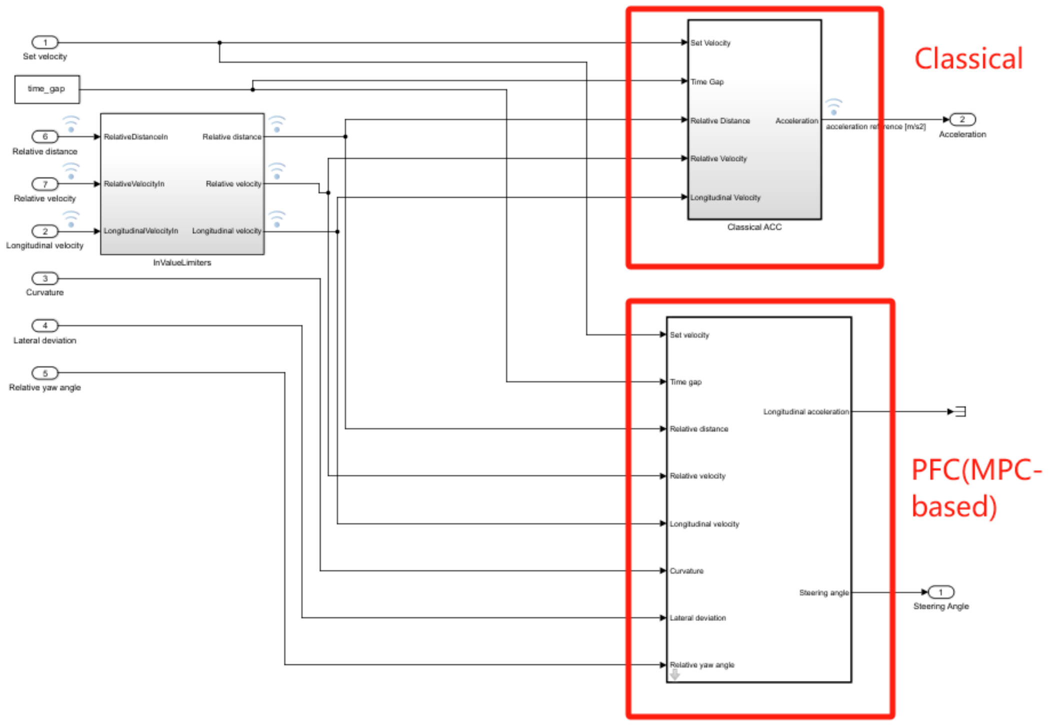


Figure 3.21: Controller Configuration(2)

(2) **Model Predictive Controller (Path Following Control System)**

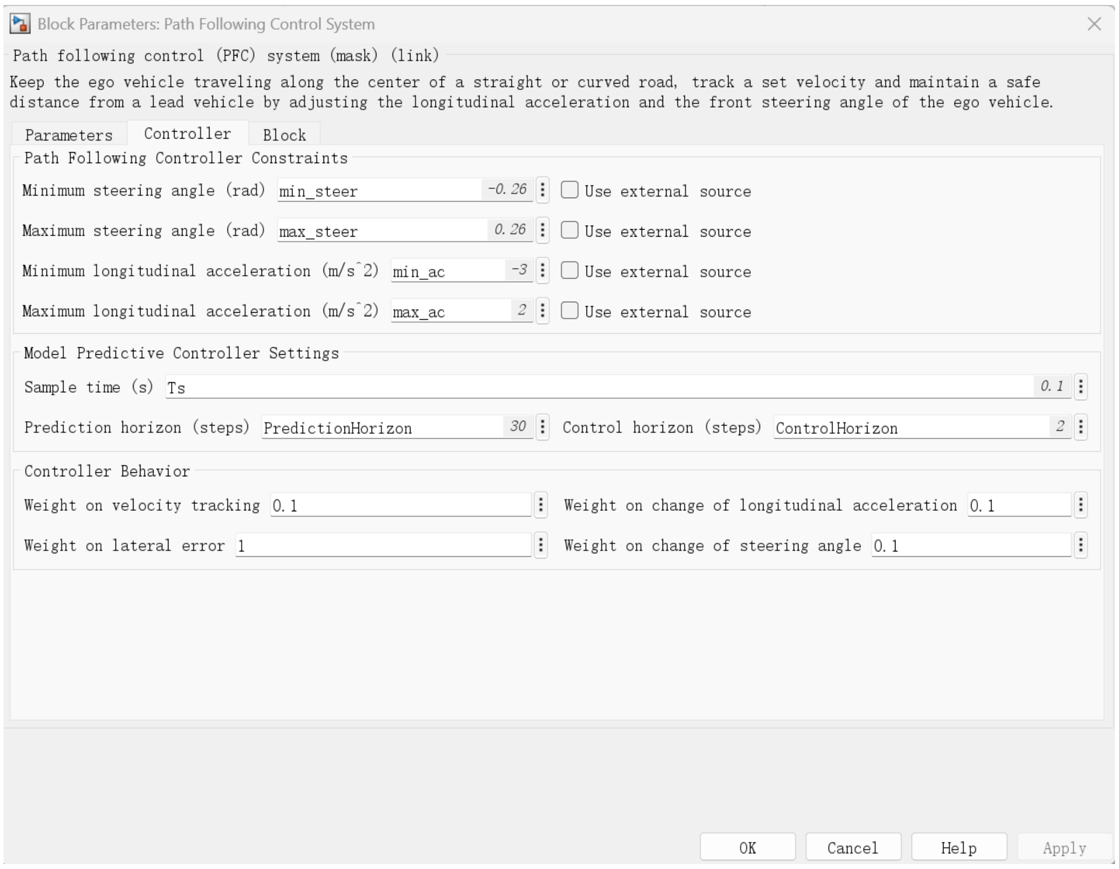
One of the main controllers evaluated is the Path Following Control System block, which applies MPC to simultaneously manage lane keeping and car-following. The block receives the lane centerline, curvature, and a previewed trajectory as references, and computes steering and acceleration commands to keep the ego vehicle centered in the lane while maintaining a target speed and safe following distance. The controller automatically enforces constraints on speed, acceleration, and steering angle, optimizing control inputs at each time step. This structure integrates lane keeping and ACC functions within a unified MPC framework.

(3) **Classical Adaptive Cruise Control (ACC) Controller**

The simulation also incorporates a classical ACC module based on a standard time gap policy. The safe distance to the lead vehicle is computed as:

$$D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \cdot V_x \tag{3.1}$$

where D_{default} and T_{gap} are design parameters, and V_x is the ego vehicle's longitudinal velocity. The controller calculates the required acceleration or


Block Parameters: Path Following Control System

Path following control (PFC) system (mask) (link)

Keep the ego vehicle traveling along the center of a straight or curved road, track a set velocity and maintain a safe distance from a lead vehicle by adjusting the longitudinal acceleration and the front steering angle of the ego vehicle.

Path Following Controller Constraints

Minimum steering angle (rad)	min_steer	-0.26	☐ Use external source
Maximum steering angle (rad)	max_steer	0.26	☐ Use external source
Minimum longitudinal acceleration (m/s^2)	min_ac	-3	☐ Use external source
Maximum longitudinal acceleration (m/s^2)	max_ac	2	☐ Use external source

Model Predictive Controller Settings

Sample time (s) Ts 0.1

Prediction horizon (steps) PredictionHorizon 30 Control horizon (steps) ControlHorizon 2

Controller Behavior

Weight on velocity tracking 0.1 Weight on change of longitudinal acceleration 0.1

Weight on lateral error 1 Weight on change of steering angle 0.1

Figure 3.22: Path Following System Controller

deceleration to maintain this gap, using current speed, relative distance, and relative velocity between the ego and lead vehicle as inputs. Control outputs are saturated to remain within acceleration limits. In this work, the classical ACC controller and its logic are directly adopted from toolbox examples without modification.

(4) **Vehicle Dynamics Model**

The plant dynamics are modeled using standard vehicle models from the Automated Driving Toolbox. For most cases, a bicycle model with force input is used to represent both longitudinal and lateral vehicle motion, while some scenarios employ a higher-fidelity 3DOF model as needed. All vehicle models use default parameter settings, since the primary goal is to assess controller performance rather than detailed physical modeling. The plant receives acceleration and steering commands from the controller and outputs the updated ego pose, velocities, and orientation.

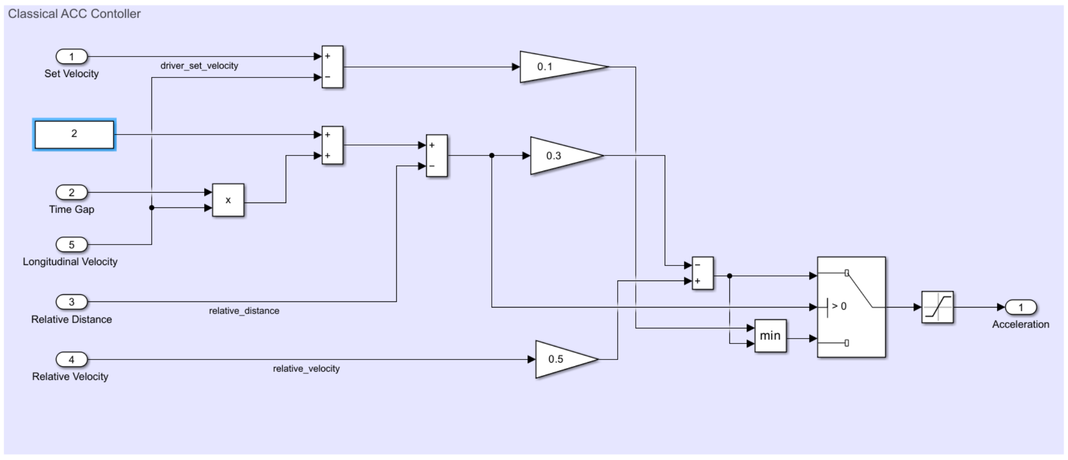


Figure 3.23: Classical Adaptive Cruise Control (ACC) Controller

All controller and plant modules are modular and can be replaced or reconfigured for further testing if required. This modular design allows a consistent and fair comparison of different control strategies under identical scenarios and vehicle conditions.

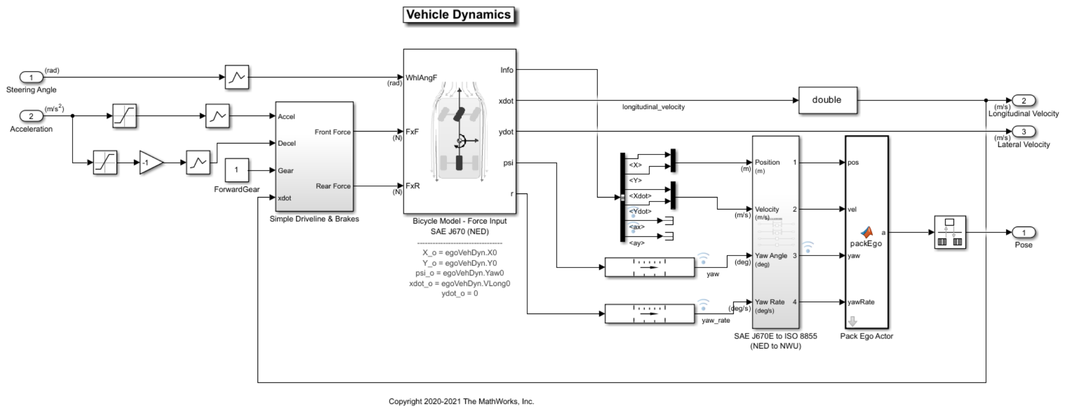


Figure 3.24: Vehicle Dynamics Using Bicycle Model

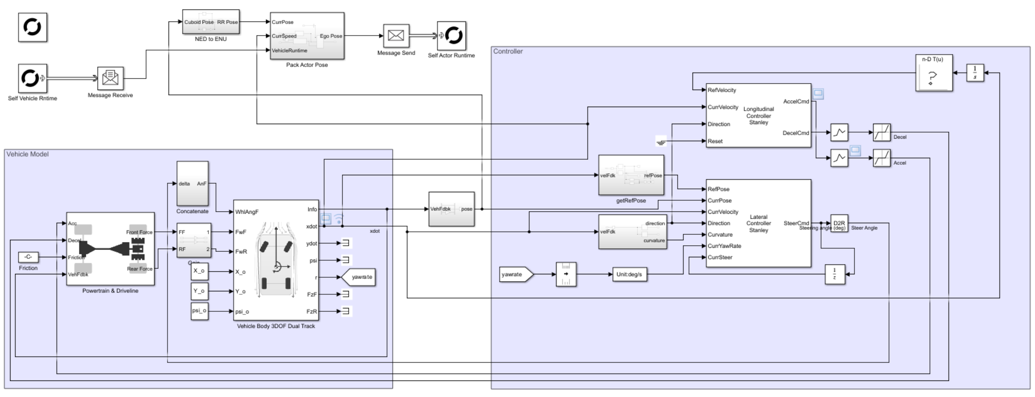


Figure 3.25: Vehicle Dynamics Using 3DOF Dual Track and Stanley Controller

3.2.6 Coordinate Transformation and Metrics Logging

- **Coordinate Transformation:** In practice, when running co-simulation between Simulink and RoadRunner, it's necessary to convert vehicle pose data between different coordinate systems. Each tool may use its own convention, so a coordinate transformation block is typically added before sending signals from Simulink to RoadRunner. In this setup, dedicated conversion modules—such as "NED to ENU" or "SAE J670E to ISO 8855 (NED to NWU)"—are used to map simulation outputs from the vehicle model into the format that RoadRunner expects[13][14].

  Here, NED (North-East-Down), ENU (East-North-Up), and NWU (North-West-Up) refer to common 3D spatial coordinate frames. Depending on how the scenario or map is set up, the underlying scene may use any of these conventions. To keep vehicle trajectories, velocities, and orientations consistent throughout the simulation, coordinate mapping is always performed before data is passed to RoadRunner.

  This transformation is a routine part of data handling in co-simulation. It can be managed using standard conversion blocks provided in the toolbox, and does not require any custom code. As long as the coordinate axes and physical meanings are aligned at each step, scenario playback and validation proceed as intended.

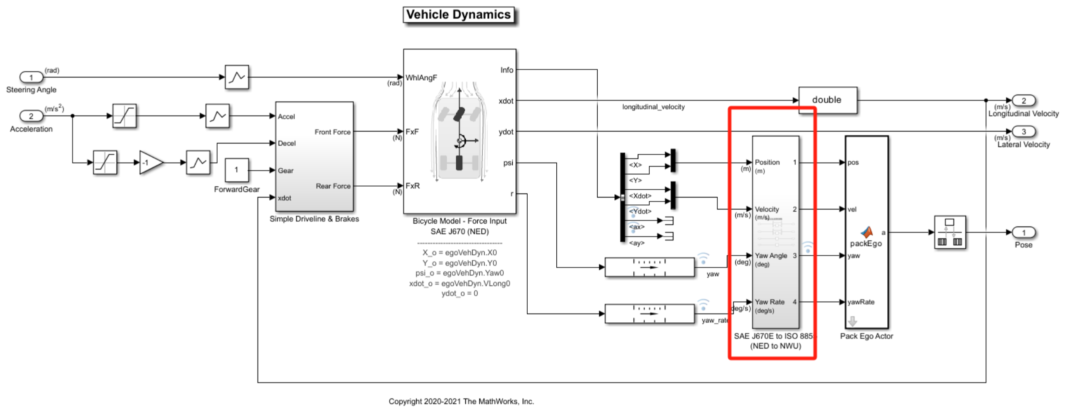


Figure 3.26: Coordinate Transformation(1)

- **Metrics Logging:** For performance evaluation, a set of standardized metrics modules from the official Automated Driving Toolbox examples are used. These modules are not custom developed but are adopted directly from the reference implementations. System-level metrics such as lane keeping, time gap, collision detection, and acceleration are recorded based on the ground

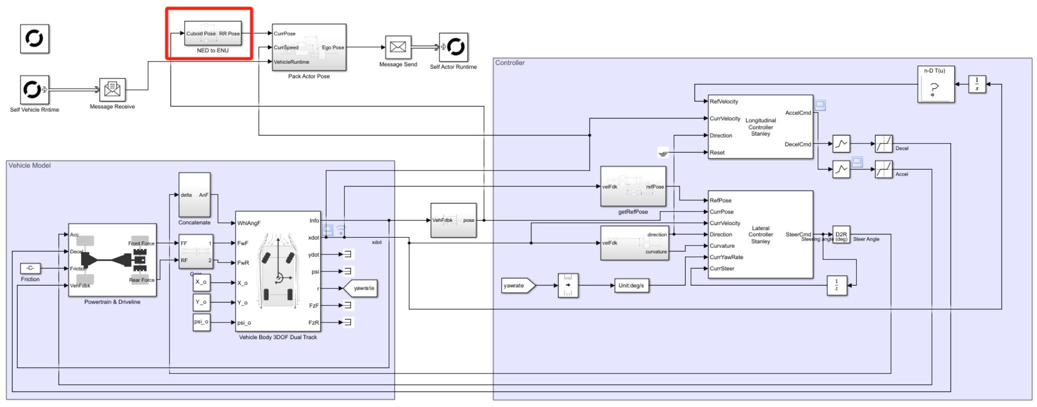


Figure 3.27: Coordinate Transformation(2)

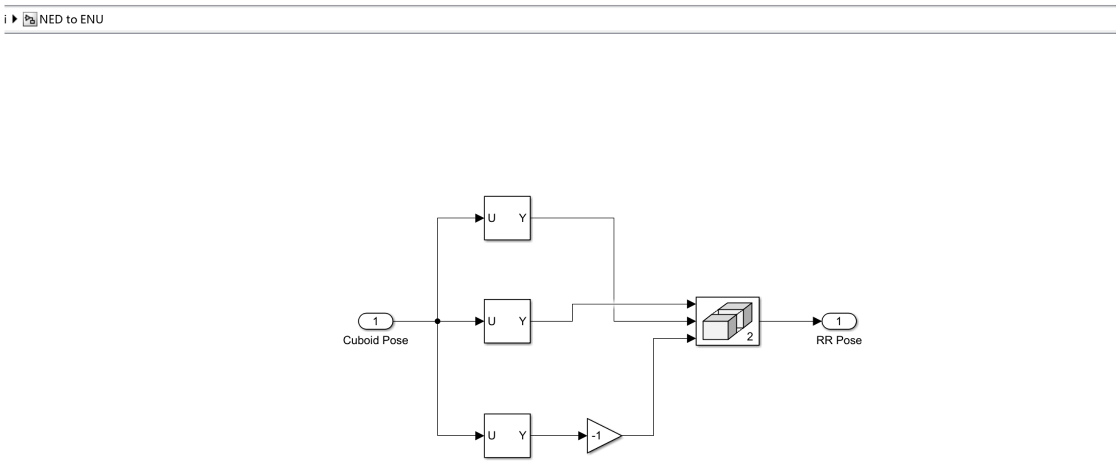


Figure 3.28: Coordinate Transformation(3)

truth and detected signals. On the component level, outputs like lane detection accuracy, vehicle bounding boxes, and sensor fusion results are also logged. The metrics structure is fully modular, allowing for batch comparison across different controllers or scenarios without further redesign.

All results—including trajectory, speed profile, relative distance, and scenario events—are saved at each run and provide the basis for later analysis and benchmarking.

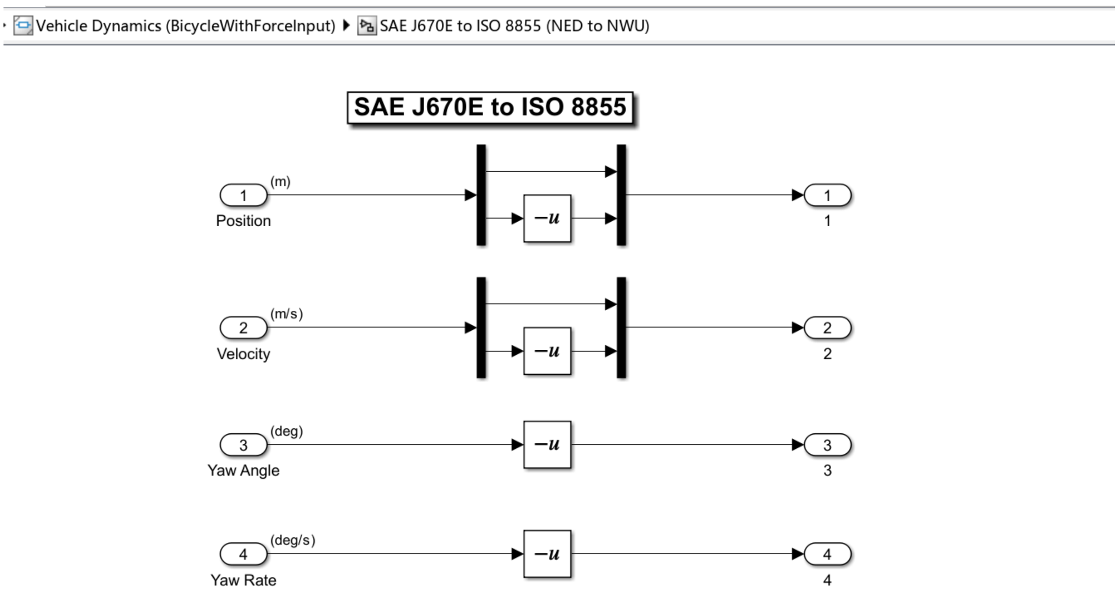


Figure 3.29: Coordinate Transformation(4)

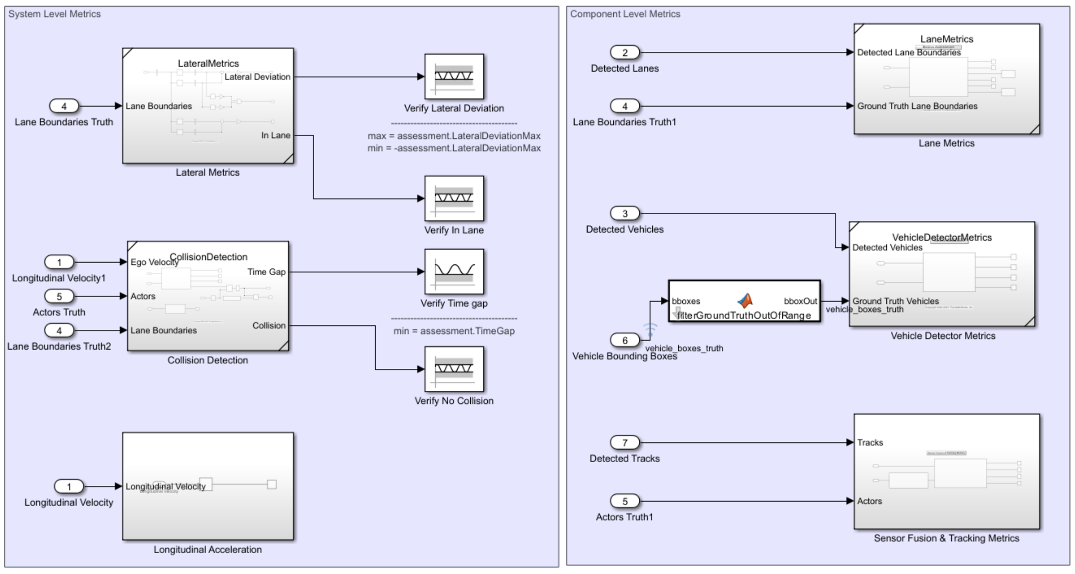


Figure 3.30: System Metrics Assessment

3.3 MATLAB Environment Setup

Scenario-based experiments can be initialized and managed through MATLAB scripts. Instead of manually opening RoadRunner and Simulink projects, the simulation environment is configured programmatically to ensure repeatability and streamline the setup process.

The workflow begins by specifying the installation path for RoadRunner and the directory of the active project within MATLAB. Using a set of built-in commands, MATLAB connects to the RoadRunner application, opens the desired project, and loads the required scenes or scenarios. This enables one-click switching between different test environments and simplifies the management of multiple simulation runs.

Typical setup steps include[15]:

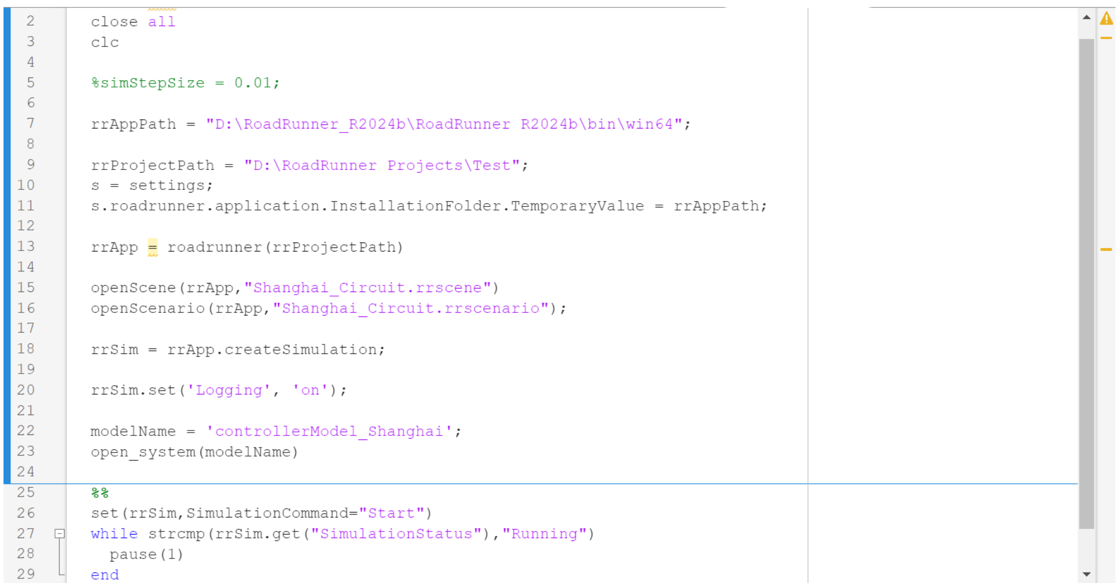
(1) Setting the path to the RoadRunner installation and project folder.

(2) Initializing a RoadRunner application object in MATLAB.

(3) Programmatically loading the target scene or scenario file for each test.

(4) (Optional) Copying or updating relevant asset files, such as behavior scripts or scene definitions, as needed for batch processing.

By centralizing environment configuration within MATLAB, the entire simulation pipeline—from scenario selection to result collection—can be run as a single script or automated job. This reduces manual intervention, eliminates potential configuration errors, and supports efficient parameter sweeps or regression testing.



```
close all
clc

%simStepSize = 0.01;

rrAppPath = "D:\RoadRunner_R2024b\RoadRunner R2024b\bin\win64";

rrProjectPath = "D:\RoadRunner Projects\Test";
s = settings;
s.roadrunner.application.InstallationFolder.TemporaryValue = rrAppPath;

rrApp = roadrunner(rrProjectPath)

openScene(rrApp,"Shanghai_Circuit.rrscene")
openScenario(rrApp,"Shanghai_Circuit.rrscenario");

rrSim = rrApp.createSimulation;

rrSim.set('Logging', 'on');

modelName = 'controllerModel_Shanghai';
open_system(modelName)

%%
set(rrSim,SimulationCommand="Start")
while strcmp(rrSim.get("SimulationStatus"),"Running")
  pause(1)
end
```

Figure 3.31: Example of MATLAB Environment Setup Script

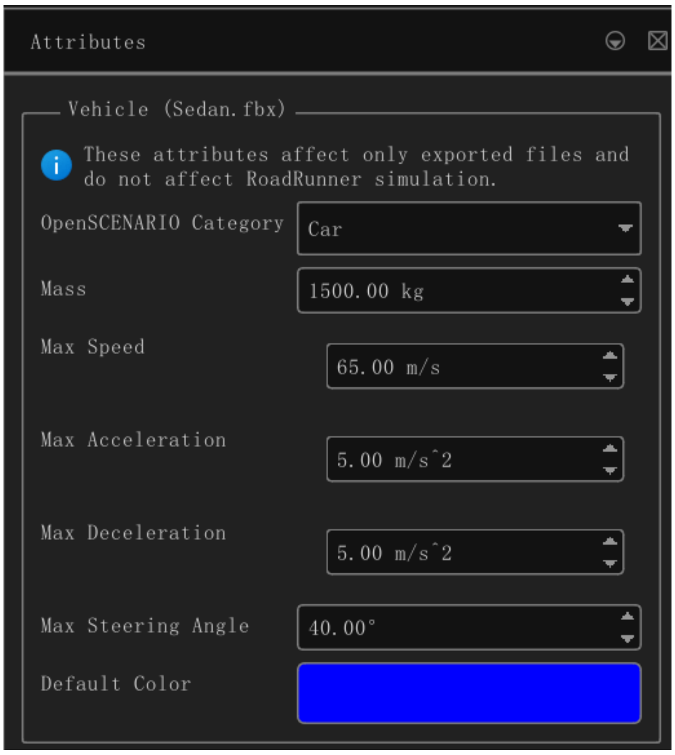
Chapter 4

Simulation Results

4.1 Simulation Results

Before the simulations start, here are some default setups:

- **Ego Vehicle Parameter Setups:** see Figure 4.1.
- **Ego Vehicle Velocity Setup:** The velocity of the ego vehicle is set to 25 meters per second (see Figure 4.2).
- **Simulation Time Setup:** The simulation time is set to 30 seconds (see Figure 4.3)
- **Simulation Step Size Setup:** Step size of the simulation is set to 0.1 second (see Figure 4.4).
- **Sensor Models Setups:** see Figure 4.5.



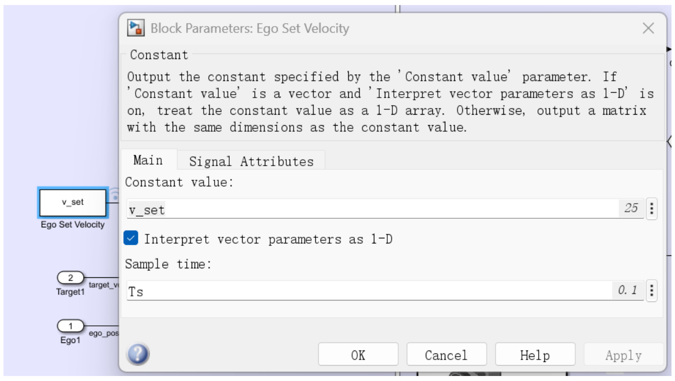
Attributes

Vehicle (Sedan.fbx)

These attributes affect only exported files and do not affect RoadRunner simulation.

OpenSCENARIO Category	Car
Mass	1500.00 kg
Max Speed	65.00 m/s
Max Acceleration	5.00 m/s^2
Max Deceleration	5.00 m/s^2
Max Steering Angle	40.00°
Default Color	🟦

Figure 4.1: Ego Vehicle Setups



Block Parameters: Ego Set Velocity

Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Constant value:

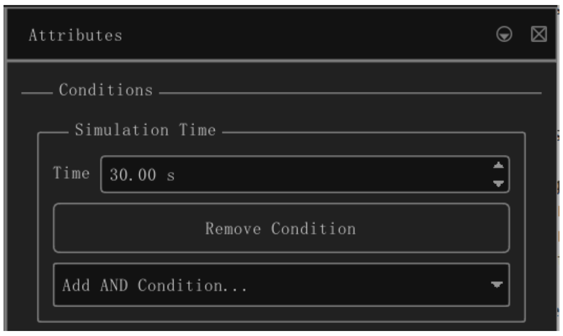
v_set 25

☑ Interpret vector parameters as 1-D

Sample time:

Ts 0.1

Figure 4.2: Ego Vehicle Velocity Setup

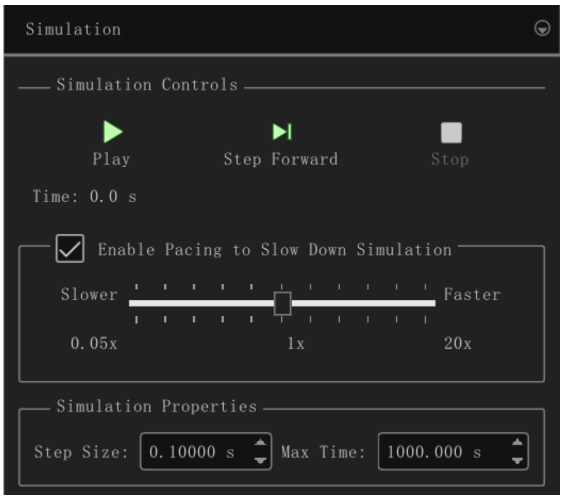

Attributes

Conditions

Simulation Time

Time: 30.00 s

Figure 4.3: Simulation Time Setup


Simulation

Simulation Controls

Time: 0.0 s

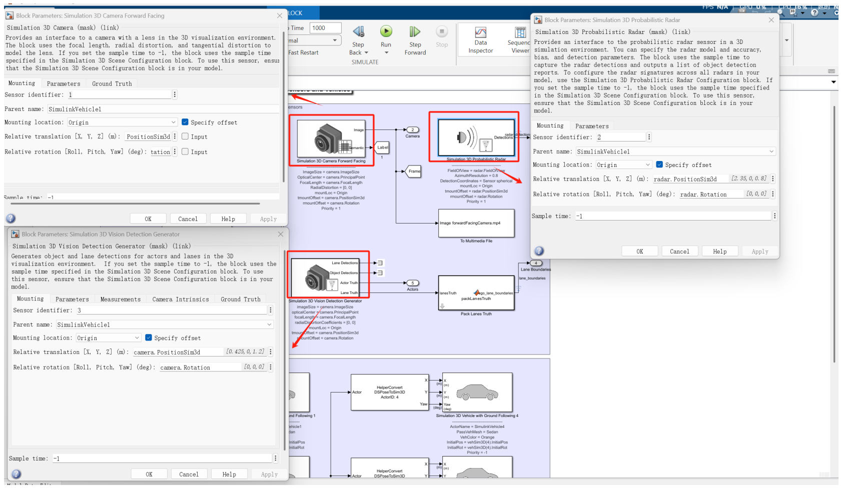
☑ Enable Pacing to Slow Down Simulation

Simulation Properties

Step Size: 0.10000 s

Max Time: 1000.000 s

Figure 4.4: Simulation Step Size Setup


Block Parameters: Simulation 3D Camera Forward Facing

Simulation 3D Camera (mask) (link)

Provides an interface to a camera with a lens in the 3D visualization environment. The block uses the focal length, radial distortion, and tangential distortion to model the lens. If you set the sample time to -1, the block uses the sample time specified in the Simulation 3D Scene Configuration block. To use this sensor, ensur that the Simulation 3D Scene Configuration block is in your model.

Mounting | Parameters | Ground Truth

Sensor identifier: 1

Parent name: SimulinkVehicle1

Mounting location: Origin ☑ Specify offset

Relative translation [X, Y, Z] (m): PositionSim3d ☐ Input

Relative rotation [Roll, Pitch, Yaw] (deg): tation ☐ Input

Sample time: -1

Block Parameters: Simulation 3D Vision Detection Generator

Simulation 3D Vision Detection Generator (mask) (link)

Generates object and lane detections for actors and lanes in the 3D visualization environment. If you set the sample time to -1, the block uses the sample time specified in the Simulation 3D Scene Configuration block. To use this sensor, ensure that the Simulation 3D Scene Configuration block is in your model.

Mounting | Parameters | Measurements | Camera Intrinsics | Ground Truth

Sensor identifier: 3

Parent name: SimulinkVehicle1

Mounting location: Origin ☑ Specify offset

Relative translation [X, Y, Z] (m): camera.PositionSim3d [0.425, 0, 1.2]

Relative rotation [Roll, Pitch, Yaw] (deg): camera.Rotation [0, 0, 0]

Sample time: -1

Block Parameters: Simulation 3D Probabilistic Radar

Simulation 3D Probabilistic Radar (mask) (link)

Provides an interface to the probabilistic radar sensor in a 3D simulation environment. You can specify the radar model and accuracy, bias, and detection parameters. The block uses the sample time to capture the radar detections and outputs a list of object detection reports. To configure the radar signatures across all radars in your model, use the Simulation 3D Probabilistic Radar Configuration block. If you set the sample time to -1, the block uses the sample time specified in the Simulation 3D Scene Configuration block. To use this sensor, ensure that the Simulation 3D Scene Configuration block is in your model.

Mounting | Parameters

Sensor identifier: 2

Parent name: SimulinkVehicle1

Mounting location: Origin ☑ Specify offset

Relative translation [X, Y, Z] (m): radar.PositionSim3d [2.35, 0, 0.8]

Relative rotation [Roll, Pitch, Yaw] (deg): radar.Rotation [0, 0, 0]

Sample time: -1

Figure 4.5: Sensor Models Setups

4.1.1 Senario1: Lead Car Deceleration

- **Description:** This scenario examines the ACC controller's response to a typical highway situation where the lead vehicle, initially cruising at a constant speed, begins to decelerate unexpectedly. The ego vehicle follows the lead car in the same lane, maintaining a steady time gap before the deceleration event is triggered.

  The test is conducted on a multi-lane road with gentle curves, designed in RoadRunner and simulated in Simulink. The initial relative distance and velocities are set to represent real-world car-following conditions. As the lead car initiates a deceleration maneuver, the performance of both ACC controllers is evaluated, particularly their ability to maintain a safe gap, ensure smooth deceleration, and avoid abrupt braking.

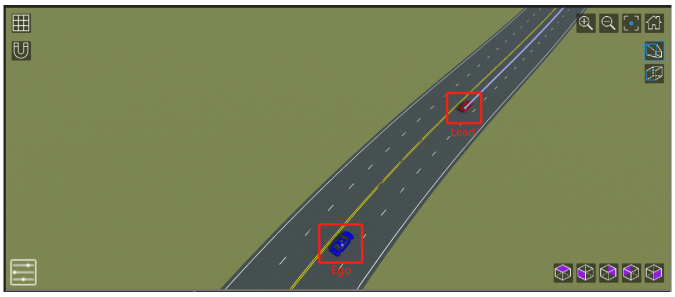


Figure 4.6: Overview of Scenario 1(1)

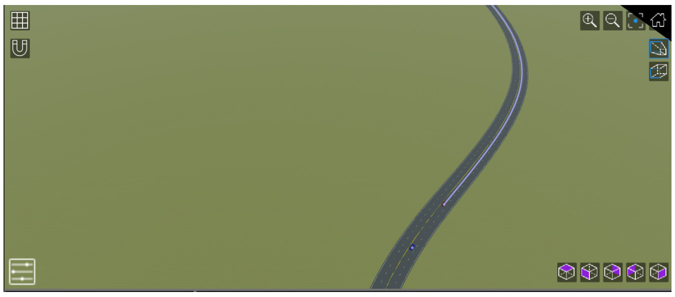


Figure 4.7: Overview of Scenario 1(2)

- **Results:** The figures below show the main outputs from the simulation of Scenario 1. Figure 4.8 illustrates the ego vehicle's forward view, where lane markers and the detected lead vehicle are clearly identified. The right panel shows the corresponding lane detection results.

  Figure 4.9 presents another perspective, combining the camera view with a bird's-eye plot of the scene. The detection zones for the ego vehicle and the trajectory of the lead car are displayed, allowing for a straightforward assessment of relative position and safety margin throughout the maneuver.

  These results confirm that the perception system correctly identifies both the lanes and the lead vehicle during a typical car-following scenario, providing reliable inputs to the ACC controller.

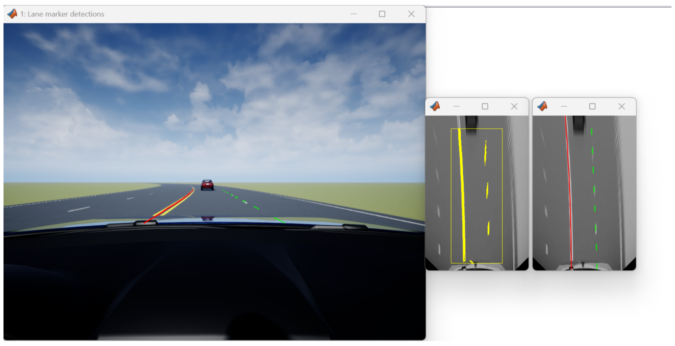


Figure 4.8: Output of Scenario 1(1)

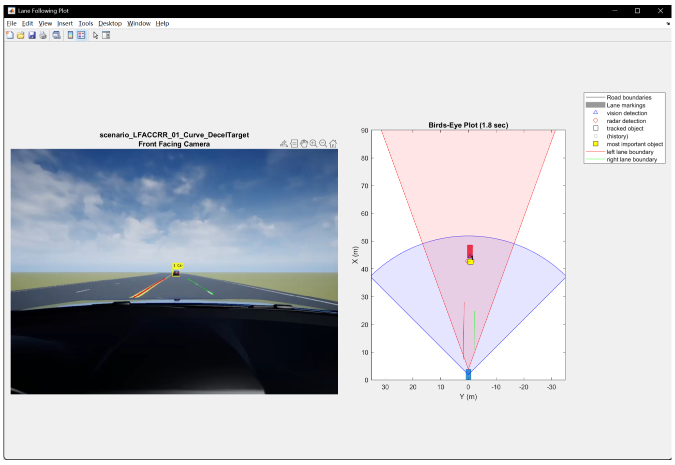


Figure 4.9: Output of Scenario 1(2)

- **Data Plots:** These results demonstrate that the simulation framework is effective in distinguishing the behavioral differences between controllers under identical driving conditions. By applying the same scenario to both controllers, the platform makes it possible to directly compare their performance in terms of safety, responsiveness, and comfort.

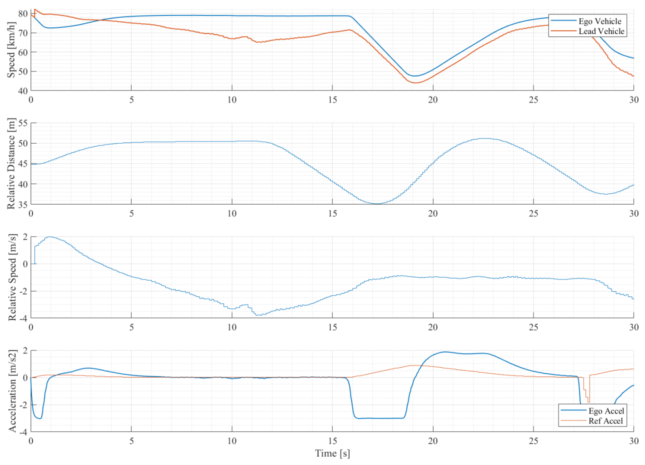


Figure 4.10: Data plots of Path Following Control System Controller

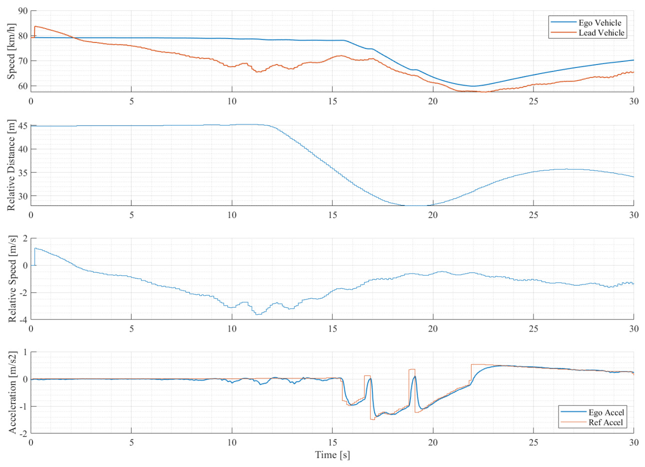


Figure 4.11: Data plots of Classical Adaptive Cruise Control (ACC) Controller

4.1.2 Senario2: Lead Car Stop and Go

- **Description:** This scenario replicates typical stop-and-go traffic conditions often encountered in urban expressways or during congestion on highways. The blue ego vehicle follows a red lead car in the same lane, while other vehicles travel in adjacent lanes.

  During the simulation, the lead vehicle suddenly decelerates to a complete stop and then accelerates back to cruising speed, mimicking the unpredictable nature of dense traffic flow. The ego vehicle must continuously adjust its speed, maintaining a safe and comfortable following distance while responding smoothly to each stop-and-go event.

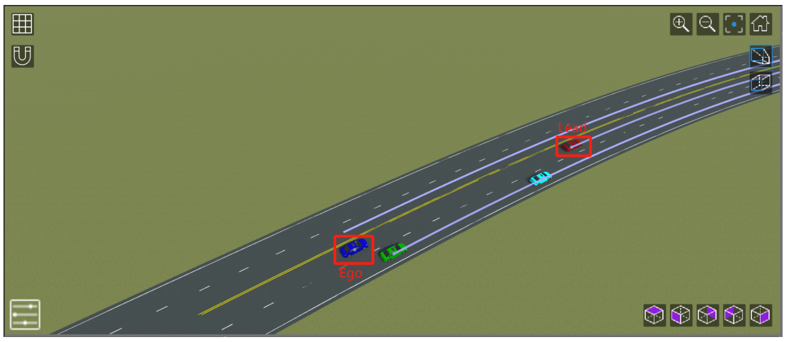


Figure 4.12: Overview of Scenario 2(1)

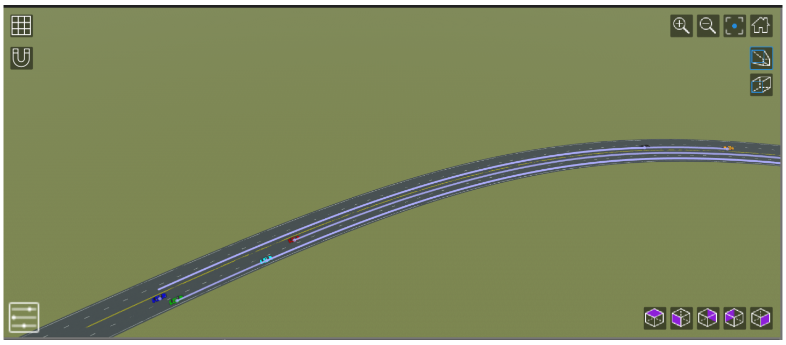


Figure 4.13: Overview of Scenario 2(2)

- **Results:** The figures below show the main outputs from the simulation of Scenario 1. Figure 4.14 illustrates the ego vehicle's forward view, where lane markers and the detected lead vehicle are clearly identified. The right panel shows the corresponding lane detection results.

  Figure 4.15 presents another perspective, combining the camera view with a bird's-eye plot of the scene. The detection zones for the ego vehicle and the trajectory of the lead car are displayed, allowing for a straightforward assessment of relative position and safety margin throughout the maneuver.

  These results confirm that the perception system correctly identifies both the lanes and the lead vehicle during a typical car-following scenario, providing reliable inputs to the ACC controller.

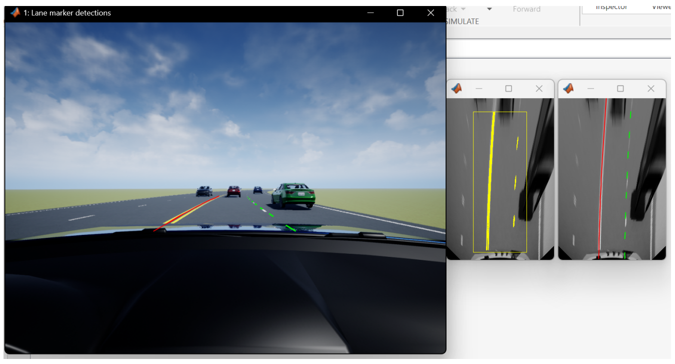


Figure 4.14: Output of Scenario 2(1)

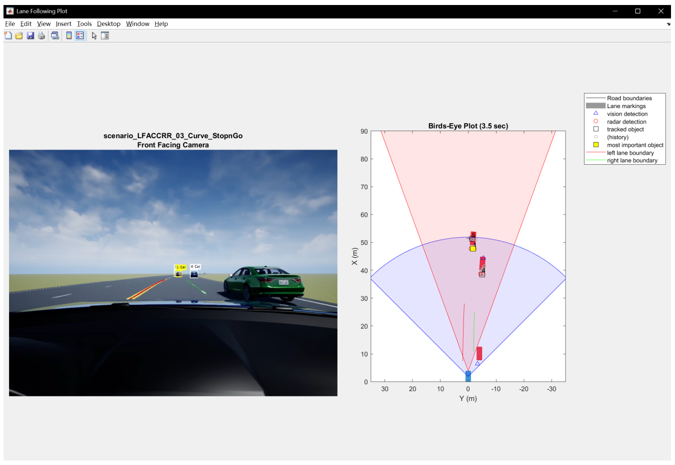


Figure 4.15: Output of Scenario 2(2)

- **Data Plots:** These results demonstrate that the simulation framework is effective in distinguishing the behavioral differences between controllers under identical driving conditions. By applying the same scenario to both controllers, the platform makes it possible to directly compare their performance in terms of safety, responsiveness, and comfort.

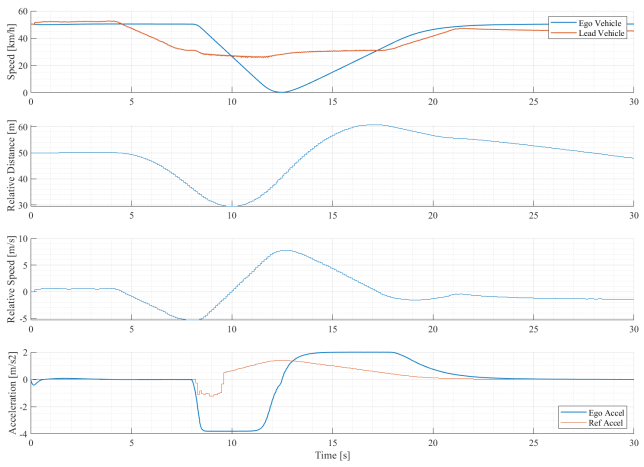


Figure 4.16: Data plots of Path Following Control System Controller

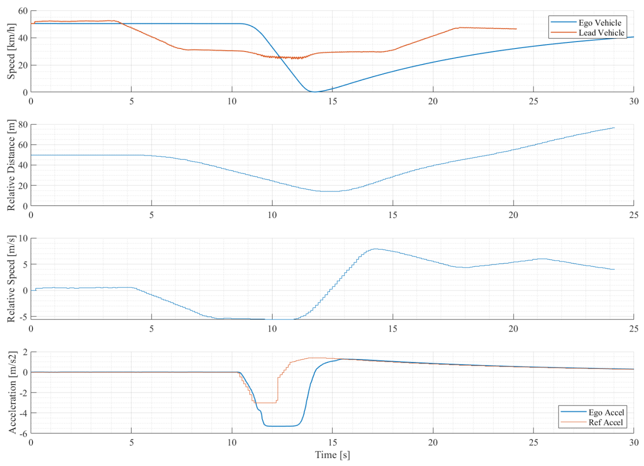


Figure 4.17: Data plots of Classical Adaptive Cruise Control (ACC) Controller

4.1.3 Senario2: Cut-In and Out

- **Description:** In this scenario, the ego vehicle (blue) follows a lead car (red) in the middle lane, while a black vehicle executes a cut-in and cut-out maneuver. The black car, initially traveling in the adjacent lane, suddenly merges into the ego vehicle's lane between the ego and lead cars, stays momentarily, and then changes back out to its original lane.

  This type of situation is common in real-world highway driving, especially during dense or unpredictable traffic. The maneuver is designed to test the ACC controller's ability to respond promptly to sudden intrusions, maintain safe following distances, and avoid unnecessary or harsh braking.

  The simulation is set up on a multi-lane curved road to reflect realistic driving conditions. By introducing unexpected lane changes, the test challenges each controller's robustness and adaptability to abrupt changes in the traffic environment.

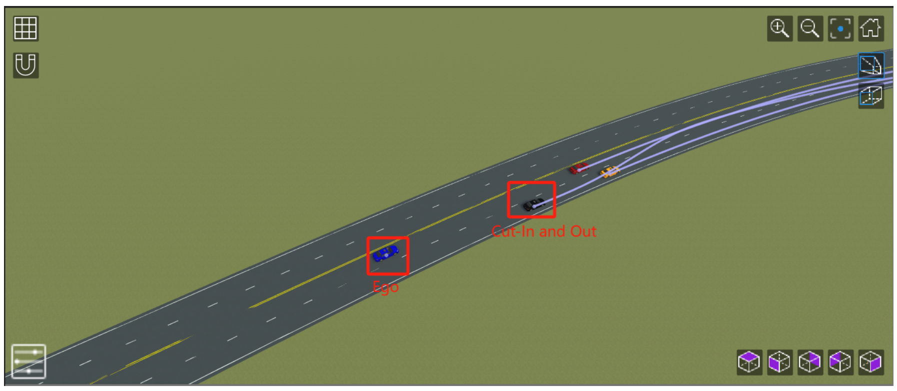


Figure 4.18: Overview of Scenario 3(1)

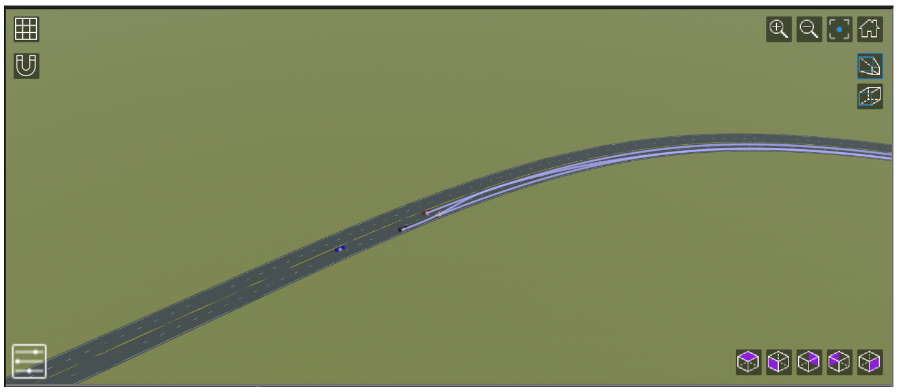


Figure 4.19: Overview of Scenario 3(2)

- **Results:** The figures below show the main outputs from the simulation of Scenario 1. Figure 4.20 illustrates the ego vehicle's forward view, where lane markers and the detected lead vehicle are clearly identified. The right panel shows the corresponding lane detection results.

  Figure 4.21 presents another perspective, combining the camera view with a bird's-eye plot of the scene. The detection zones for the ego vehicle and the trajectory of the lead car are displayed, allowing for a straightforward assessment of relative position and safety margin throughout the maneuver.

  These results confirm that the perception system correctly identifies both the lanes and the lead vehicle during a typical car-following scenario, providing reliable inputs to the ACC controller.

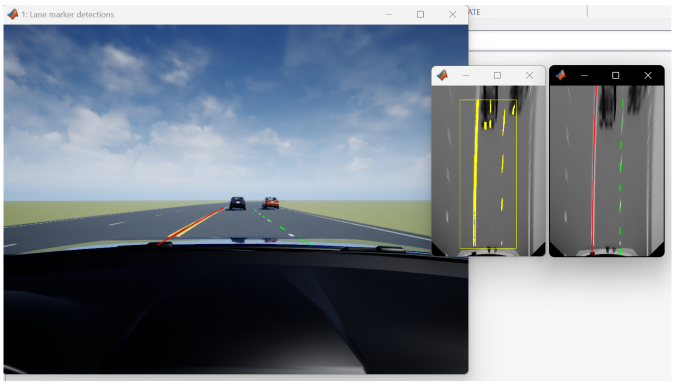


Figure 4.20: Output of Scenario 3(1)

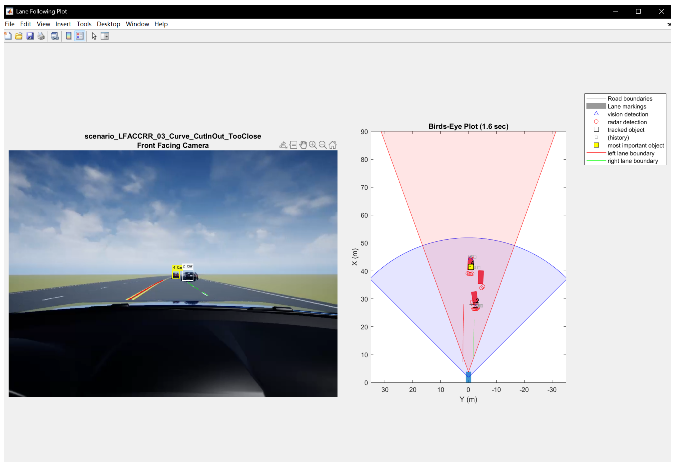


Figure 4.21: Output of Scenario 3(2)

- **Data Plots:** These results demonstrate that the simulation framework is effective in distinguishing the behavioral differences between controllers under identical driving conditions. By applying the same scenario to both controllers, the platform makes it possible to directly compare their performance in terms of safety, responsiveness, and comfort.

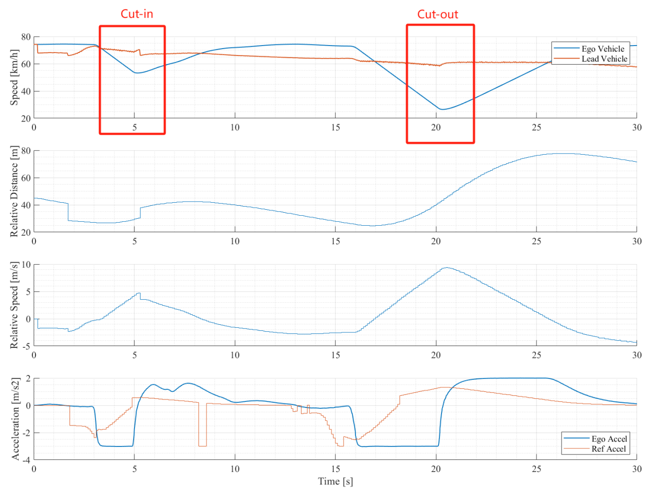


Figure 4.22: Data plots of Path Following Control System Controller

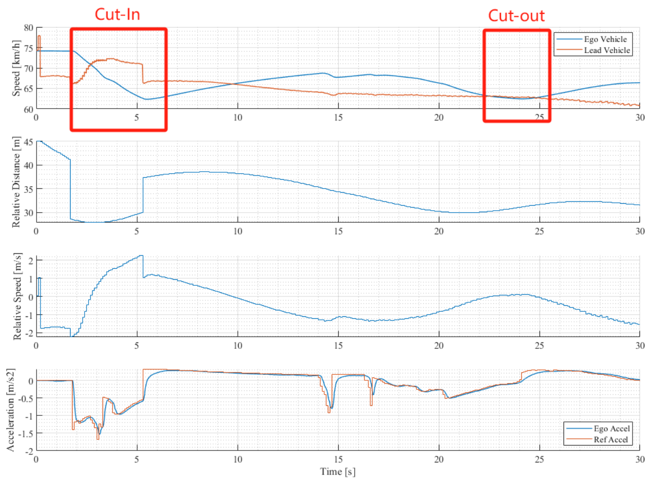


Figure 4.23: Data plots of Classical Adaptive Cruise Control (ACC) Controller

4.2 Additional Simulation: Shanghai International Circuit

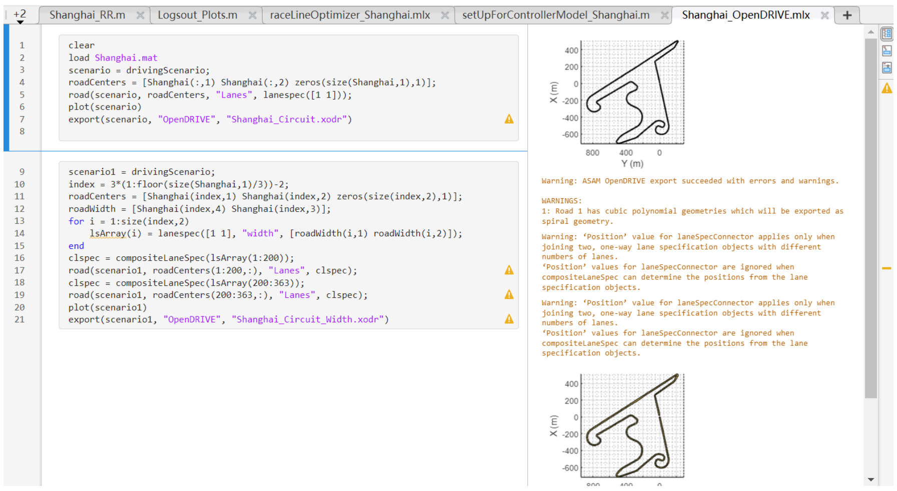
This additional simulation is designed to validate the capability of the platform to recreate real-world scenarios using measured GPS data. It also allows for testing custom vehicle dynamics and control algorithms under complex and realistic driving conditions.

4.2.1 Real-World Map Data Import and Scene Generation

The process starts with collecting high-resolution GPS data along the full lap of the Shanghai International Circuit. The raw coordinates are imported and used to define the centerline of the track. In MATLAB, the road geometry is reconstructed using the drivingScenario function, allowing for both the basic track shape and lane configurations to be accurately recreated.

To create an optimized reference path, the raw GPS data is fed into a racing line optimizer. Here, vehicle parameters (mass, acceleration limits, maximum lateral force, etc.) are defined, and the tool computes a minimum-curvature or time-optimal trajectory. This reference line serves as the target for controller testing and velocity profiling[16][17].

This workflow combines real-world measurement with simulation, enabling tests that reflect true track geometry and dynamics rather than idealized or synthetic road models.



```
clear
load Shanghai.mat
scenario = drivingScenario;
roadCenters = [Shanghai(:,1) Shanghai(:,2) zeros(size(Shanghai,1),1)];
road(scenario, roadCenters, "Lanes", lanespec([1 1]));
plot(scenario)
export(scenario, "OpenDRIVE", "Shanghai_Circuit.xodr")
```

```
scenario1 = drivingScenario;
index = 3*(1:floor(size(Shanghai,1)/3))-2;
roadCenters = [Shanghai(index,1) Shanghai(index,2) zeros(size(index,2),1)];
roadWidth = [Shanghai(index,4) Shanghai(index,3)];
for i = 1:size(index,2)
    lsArray(i) = lanespec([1 1], "width", [roadWidth(i,1) roadWidth(i,2)]);
end
clspec = compositeLaneSpec(lsArray(1:200));
road(scenario1, roadCenters(1:200,:), "Lanes", clspec);
clspec = compositeLaneSpec(lsArray(200:363));
road(scenario1, roadCenters(200:363,:), "Lanes", clspec);
plot(scenario1)
export(scenario1, "OpenDRIVE", "Shanghai_Circuit_Width.xodr")
```

```
Warning: ASAM OpenDRIVE export succeeded with errors and warnings.

WARNINGS:
1: Road 1 has cubic polynomial geometries which will be exported as
spiral geometry.

Warning: 'Position' value for laneSpecConnector applies only when
joining two, one-way lane specification objects with different
numbers of lanes.
'Position' values for laneSpecConnector are ignored when
compositeLaneSpec can determine the positions from the lane
specification objects.

Warning: 'Position' value for laneSpecConnector applies only when
joining two, one-way lane specification objects with different
numbers of lanes.
'Position' values for laneSpecConnector are ignored when
compositeLaneSpec can determine the positions from the lane
specification objects.
```

Figure 4.24: Map Data Transformation

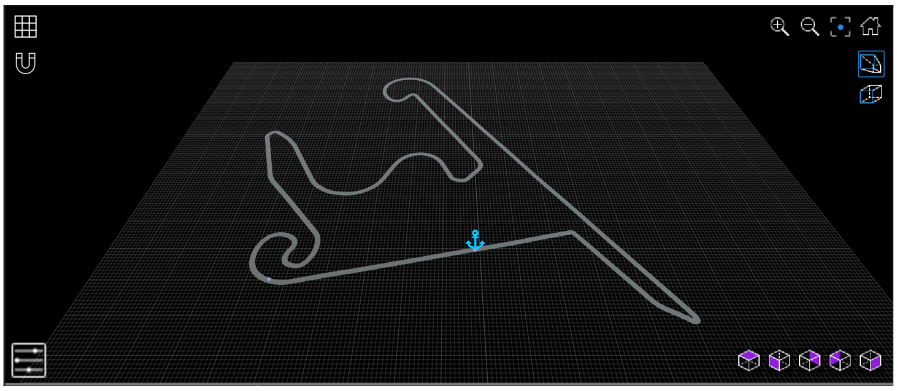
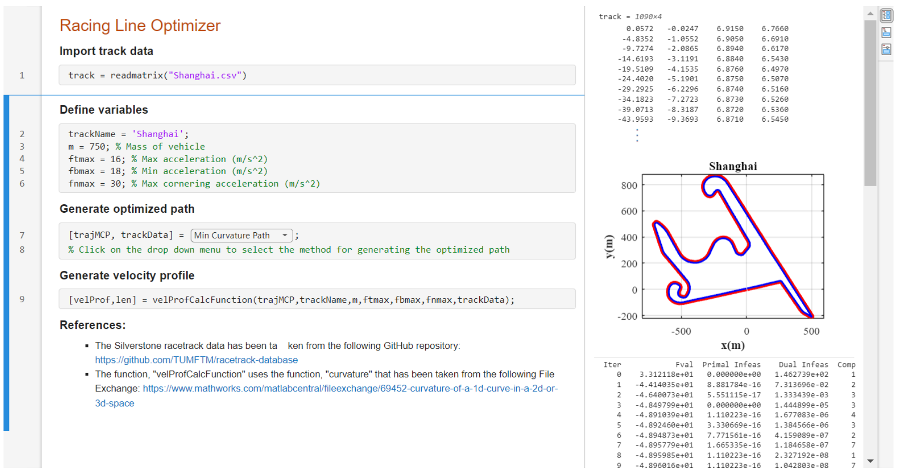


Figure 4.25: Shanghai International Circuit imported in RoadRunner

4.2.2 Vehicle Dynamics and Controller Integration

The simulation framework supports any custom vehicle dynamics model. For this test, we used a vehicle model implemented in Simulink, together with a raceline optimizer controller. The setup is straightforward: the controller takes the optimized path and speed profile as input, and then computes steering, throttle, and brake commands for the vehicle model[18].

This approach makes it easy to swap in different vehicle models or controllers as needed. The main point here is to verify that our workflow handles both the actual Shanghai track geometry and a user-defined dynamics-control stack without issues.



Racing Line Optimizer

Import track data

```
track = readmatrix("Shanghai.csv")
```

Define variables

```
trackName = 'Shanghai';
m = 750; % Mass of vehicle
ftmax = 16; % Max acceleration (m/s^2)
fbmax = 18; % Min acceleration (m/s^2)
fnmax = 30; % Max cornering acceleration (m/s^2)
```

Generate optimized path

```
[trajMCP, trackData] = Min Curvature Path ;
% Click on the drop down menu to select the method for generating the optimized path
```

Generate velocity profile

```
[velProf,len] = velProfCalcFunction(trajMCP,trackName,m,ftmax,fbmax,fnmax,trackData);
```

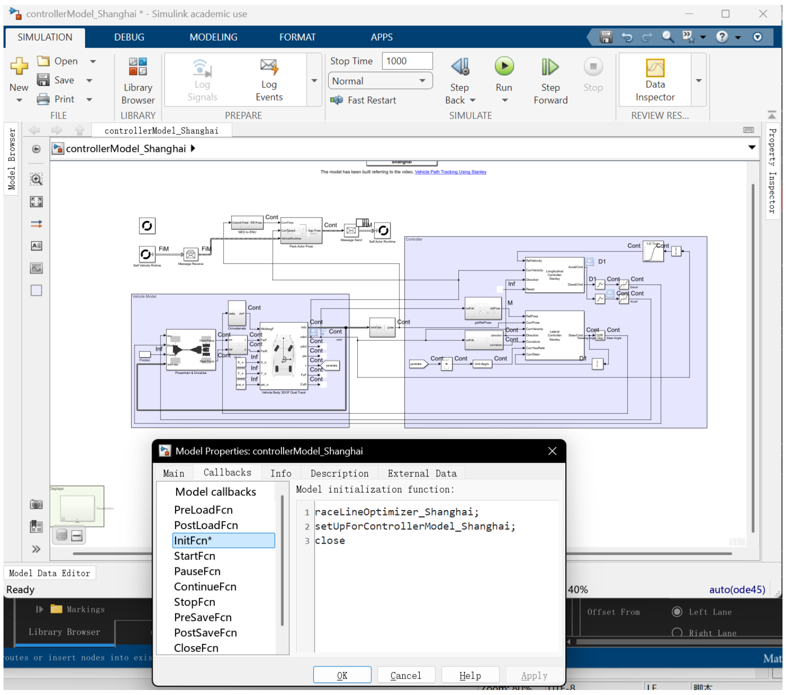
References:

- The Silverstone racetrack data has been ta ken from the following GitHub repository: https://github.com/TUMFTM/racetrack-database
- The function, "velProfCalcFunction" uses the function, "curvature" that has been taken from the following File Exchange: https://www.mathworks.com/matlabcentral/fileexchange/69452-curvature-of-a-1d-curve-in-a-2d-or-3d-space

```
track = 1090×4
     0.0572   -0.0247    6.9150    6.7660
    -4.8352   -1.0552    6.9050    6.6910
    -9.7274   -2.0865    6.8940    6.6170
   -14.6193   -3.1191    6.8840    6.5430
   -19.5109   -4.1535    6.8760    6.4970
   -24.4020   -5.1901    6.8750    6.5070
   -29.2925   -6.2296    6.8740    6.5160
   -34.1823   -7.2723    6.8730    6.5260
   -39.0713   -8.3187    6.8720    6.5360
   -43.9593   -9.3693    6.8710    6.5450
        ⋮
```

```
 Iter          Fval   Primal Infeas     Dual Infeas   Comp
    0  3.312118e+01    0.000000e+00    1.462739e+02      1
    1 -4.414035e+01    8.881784e-16    7.313696e-02      2
    2 -4.640073e+01    5.551115e-17    1.333439e-03      3
    3 -4.849799e+01    0.000000e+00    1.444899e-05      3
    4 -4.891039e+01    1.110223e-16    1.677083e-06      4
    5 -4.892460e+01    3.330669e-16    1.384566e-06      3
    6 -4.894873e+01    7.771561e-16    4.159089e-07      2
    7 -4.895779e+01    1.665335e-16    1.184658e-07      7
    8 -4.895985e+01    1.110223e-16    2.327192e-08      1
    9 -4.896016e+01    1.110223e-16    1.042803e-08      7
```

Figure 4.26: Raceline Optimizer



Stop Time 1000

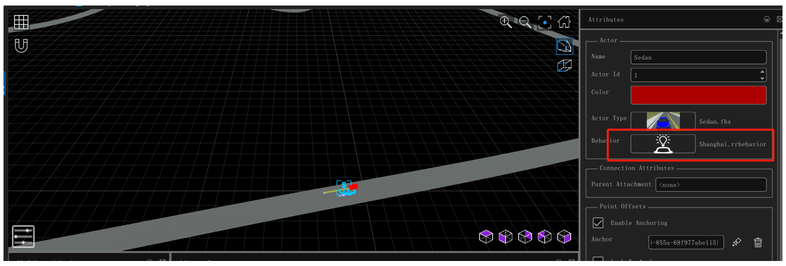
Normal

Fast Restart

Model initialization function:

```
raceLineOptimizer_Shanghai;
setUpForControllerModel_Shanghai;
close
```

Figure 4.27: Vehicle Dynamics and Controller Integration



Name Sedan

Actor Id 1

Color

Actor Type Sedan.fbx

Behavior Shanghai.rrbehavior

Parent Attachment (none)

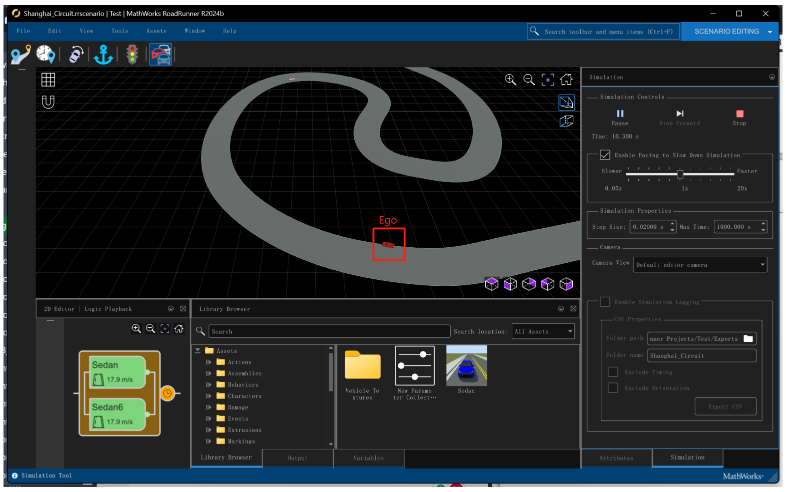
Enable Anchoring

Figure 4.28: Behavior Integration in RoadRunner

4.2.3 Results and Summary

The simulation ran smoothly with the custom vehicle model and raceline optimizer controller on the Shanghai International Circuit scene. The car was able to follow the optimized path for a full lap without leaving the track or triggering faults.

Overall, this test shows that the platform can handle real-world track geometry and user-defined vehicle-control setups without extra tuning or adjustments. It confirms that the workflow is practical for validating new control strategies or vehicle models on actual road layouts.



Time: 10.300 s

[✓] Enable Pacing to Slow Down Simulation

Simulation Properties

Step Size: 0.02000 s Max Time: 1000.000 s

Camera

Camera View: Default editor camera

[ ] Enable Simulation Logging

CSV Properties

Folder path: nner Projects/Test/Exports

Folder name: Shanghai_Circuit

[ ] Exclude Timing

[ ] Exclude Orientation

Export CSV

Figure 4.29: Simulation in Shanghai International Circuit

Chapter 5

Conclusions and Future Works

This thesis systematically investigated the scenario-based evaluation of ADAS/ACC algorithms under different driving conditions. Three typical traffic scenarios—Cut-In and Out, lead vehicle stop-and-go, and lead vehicle deceleration—were constructed to comprehensively assess the performance boundaries and robustness of the control strategy in urban, suburban, and mixed-traffic contexts. Additionally, by integrating GPS-derived trajectory data from the Shanghai International Circuit, the methodology was further extended to a motorsport environment, enabling the quantitative study of lap-time minimization and optimal trajectory planning in a highly dynamic and competitive context.

Through the development of a modular simulation environment and the design of representative traffic scenarios, this work demonstrated the effectiveness of scenario-based validation for advanced control algorithms. The results from Cut-In/Out scenarios confirmed the algorithm's capacity to maintain safe following distances and perform timely, stable responses to sudden lane changes by adjacent vehicles. In stop-and-go and deceleration cases, the strategy exhibited robust adaptation to varying lead vehicle behaviors, ensuring longitudinal safety and comfort even under frequent acceleration and deceleration cycles.

The introduction of the Shanghai Circuit simulation not only validated the flexibility of the control framework under non-standard conditions but also showcased the algorithm's potential for trajectory optimization. By incorporating real GPS track data, the simulation environment successfully reproduced realistic racing scenarios, providing a new perspective for the integration of ADAS/ACC functions in high-performance and motorsport applications.

Despite the promising results obtained in this study, several limitations and future research directions remain:

1. **Scenario Diversity and Coverage:** The current scenarios, while representative, do not fully capture the entire operational design domain (ODD) of real-world traffic. Future work should extend the scenario library to include more complex situations, such as multi-agent interactions, vulnerable road users (VRUs), adverse weather, and night-time conditions, to improve the completeness and stress coverage of the validation framework.

2. **Sensor and Perception Model Fidelity:** In this thesis, perception errors and sensor noise were not explicitly modeled. A logical next step is to introduce realistic sensor simulation—integrating noise, delays, and misdetections—so as to evaluate the closed-loop robustness of the controller against perception-level uncertainties.

3. **Closed-Loop Testing with Physical Vehicles:** While the current work is grounded in high-fidelity simulation, transitioning to hardware-in-the-loop (HIL) or vehicle-in-the-loop (VIL) testing would provide more practical insights. Real-time validation with physical vehicles, especially in controlled proving ground scenarios, will further bridge the gap between simulation and deployment.

4. **Adaptive and Learning-Based Control:** The control strategy in this thesis was implemented in a rule-based or model-driven manner. Future studies could explore the integration of data-driven and learning-based approaches, such as reinforcement learning, to enhance adaptability in complex or unforeseen traffic situations, and to enable continuous policy improvement with real-world feedback.

5. **Generalization to Different Road Classes:** The Shanghai International Circuit experiment illustrates the extensibility of the approach to special road networks. Expanding the methodology to cover a wider variety of real-world road geometries—such as mountainous highways, rural roads, and complex intersections—will help assess the universality and transferability of the control policies.

6. **Standardization and Benchmarking:** Establishing standardized metrics and public scenario benchmarks, especially for time-optimal control and safety-critical events, would facilitate comparison across different algorithms and research groups, supporting the broader advancement of scenario-based ADAS validation.

In summary, the scenario-based validation methodology proposed in this thesis provides a solid foundation for further research and development in the field of advanced driver assistance systems. Future work will look at adding more types of scenarios, making the perception and control modules closer to what you'd find in real cars, and finding ways to bring the simulation results closer to what actually happens on the road.

List of Tables

List of Figures

Bibliography

[1] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.* Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104. URL: https://doi.org/10.4271/J3016_202104 (cit. on p. 1).

[2] United Nations Economic Commission for Europe. *UN Regulation No. 157 – Automated Lane Keeping Systems (ALKS).* Standard. 2021. URL: https://unece.org/transport/documents/2021/03/standards/un-regulation-no-157-automated-lane-keeping-systems-alks (cit. on p. 3).

[3] National Highway Traffic Safety Administration. *Automated Vehicles for Safety.* Online; accessed 2025-07-14. 2024. URL: https://www.nhtsa.gov/vehicle-safety/automated-vehicles-safety (cit. on p. 3).

[4] Central Desk. *China Launches Pilot Program for Vehicle-Road-Cloud Integration in ICVs.* Auto World Journal, published July 3, 2024. Online; accessed 2025-07-14. 2024. URL: https://autoworldjournal.com/china-launches-pilot-program-for-vehicle-road-cloud-integration-in-icvs (cit. on p. 4).

[5] Gabriella. *China approves nine automakers for L3 intelligent connected vehicle pilot program.* Gasgoo Auto News, published June 6, 2024. Online; accessed 2025-07-14. 2024. URL: https://autonews.gasgoo.com/icv/70033401.html (cit. on p. 4).

[6] Grant Maloy Smith. *Types of ADAS Sensors in Use Today.* DEWESoft Blog, published Feb. 14, 2023. Online; accessed 2025-07-14. 2023. URL: https://dewesoft.com/blog/types-of-adas-sensors (cit. on p. 7).

[7] Stefan Chamraz and Richard Balogh. «Two approaches to the adaptive cruise control (ACC) design». In: *2018 Cybernetics & Informatics (K&I).* IEEE, 2018, pp. 1–5. DOI: 10.1109/CYBERI.2018.8337542. URL: https://ieeexplore.ieee.org/document/8337542 (cit. on p. 8).

[8] Antonio Sciarretta and Ardalan Vahidi. *Energy-Efficient Driving of Road Vehicles: Toward Cooperative, Connected, and Automated Mobility*. Cham: Springer, 2020. ISBN: 978-3-030-24127-8. URL: `https://link.springer.com/book/10.1007/978-3-030-24127-8` (cit. on p. 9).

[9] Simona Onori, Lorenzo Serrao, and Giorgio Rizzoni. *Hybrid Electric Vehicles: Energy Management Strategies*. Springer, 2015. ISBN: 978-1-4471-6779-2. URL: `https://books.google.it/books/about/Hybrid_Electric_Vehicles.html?id=HCY3CwAAQBAJ` (cit. on p. 9).

[10] Abdussalam Ali Ahmed, Johnson Santhosh, and Ftema W. Aldbea. «Vehicle Dynamics Modeling and Simulation with Control Using Single Track Model». In: *2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*. IEEE, 2020, pp. 1–5. DOI: `10.1109/WIECON-ECE52138.2020.9397983`. URL: `https://ieeexplore.ieee.org/document/9397983` (cit. on p. 9).

[11] Bingjian Yue, Shuming Shi, Shuo Wang, and Nan Lin. «Low-Cost Urban Test Scenario Generation Using Microscopic Traffic Simulation». In: *IEEE Access* 8 (2020), pp. 122194–122203. DOI: `10.1109/ACCESS.2020.3006561`. URL: `https://ieeexplore.ieee.org/document/9129787` (cit. on p. 21).

[12] Liangyao Yu and Ruyue Wang. «Researches on Adaptive Cruise Control system: A state of the art review». In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 236.2-3 (2021), pp. 509–529. DOI: `10.1177/09544070211019254`. URL: `https://journals.sagepub.com/doi/10.1177/09544070211019254` (cit. on p. 32).

[13] SAE International. *Vehicle Dynamics Terminology, SAE Standard J670_202206*. Reaffirmed 2022-06-09. 2022. URL: `https://www.sae.org/standards/content/j670_202206/` (cit. on p. 37).

[14] International Organization for Standardization. *ISO 8855:2011(en): Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*. Accessed: 2025-07-14. 2011. URL: `https://www.iso.org/standard/50685.html` (cit. on p. 37).

[15] MathWorks. *Connect MATLAB and RoadRunner to Control and Analyze Simulations*. Online; accessed 2025-07-14. 2025. URL: `https://ww2.mathworks.cn/help/driving/ug/connect-matlab-and-roadrunner.html` (cit. on p. 40).

[16] putta54. *MW208_Raceline_Optimization*. GitHub repository; accessed 2025-07-14. 2020. URL: `https://github.com/putta54/MW208_Raceline_Optimization` (cit. on p. 54).

[17] Alexander Heilmeier. *racetrack-database*. GitHub repository; accessed 2025-07-14. 2019. URL: `https://github.com/TUMFTM/racetrack-database` (cit. on p. 54).

[18] MathWorks Student Competitions Team. *Vehicle Path Tracking Using Stanley Controller*. MATLAB Central File Exchange; version 1.0.2; accessed 2025-07-14. 2021. URL: `https://ww2.mathworks.cn/matlabcentral/fileexchange/88977-vehicle-path-tracking-using-stanley-controller` (cit. on p. 55).