Vehicle Dynamics Blockset™

User's Guide

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Vehicle Dynamics Blockset™ User's Guide

© COPYRIGHT 2018–2023 by The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2018	Online only	New for Version 1.0 (Release 2018a)
September 2018	Online only	Revised for Version 1.1 (Release 2018b)
March 2019	Online only	Revised for Version 1.2 (Release 2019a)
September 2019	Online only	Revised for Version 1.3 (Release 2019b)
March 2020	Online only	Revised for Version 1.4 (Release 2020a)
September 2020	Online only	Revised for Version 1.5 (Release 2020b)
March 2021	Online only	Revised for Version 1.6 (Release 2021a)
September 2021	Online only	Revised for Version 1.7 (Release 2021b)
March 2022	Online only	Revised for Version 1.8 (Release 2022a)
September 2022	Online only	Revised for Version 1.9 (Release 2022b)
March 2023	Online only	Revised for Version 2.0 (Release 2023a)

Contents

1

Getting Started

2

Coordinate Systems

3

Reference Applications

Project Templates

4

Maneuver Standards

5

Supporting Data

6

Vehicle Dynamics Blockset Examples

7

<div style="text-align: right">

3D Simulation

</div>

8

<div style="text-align: right">

Virtual Vehicle Composer

</div>

9

Getting Started

Vehicle Dynamics Blockset Product Description

Model and simulate vehicle dynamics in a virtual 3D environment

Vehicle Dynamics Blockset™ provides preassembled automotive vehicle dynamics reference applications for passenger cars, trucks, and two-wheelers. The blockset includes a component library for propulsion, steering, suspension, vehicle body, brakes, tires, and driver models, as well as component and supervisory controllers. You can use the built-in interface with Unreal Engine® to visualize simulations and communicate scene information back to your model.

Vehicle Dynamics Blockset offers the Virtual Vehicle Composer app for configuring and parameterizing models, as well as prebuilt workflows for Kinematics and Compliance (K&C) testing and calibrating models from test data. You can use these models for ride and handling analyses, chassis controls development, software integration testing, and hardware-in-the-loop (HIL) testing. The models are open, so you can incorporate your own subsystems and customize them as needed.

Key Features

- Preassembled vehicle dynamics models for passenger cars and trucks
- Preassembled maneuvers for common ride and handling tests, including a double-lane change
- 3D environment for visualizing simulations and communicating scene information to Simulink®
- Libraries of propulsion, steering, suspension, vehicle body, brake, and tire components
- Combined longitudinal and lateral slip dynamic tire models
- Predictive driver model for generating steering commands that track a predefined path
- Prebuilt 3D scenes, including straight roads, curved roads, and parking lots

Acknowledgements

- Vehicle Dynamics Blockset uses the Unreal® Engine. Unreal® is a trademark or registered trademark of Epic Games®, Inc. in the United States of America and elsewhere.

  Unreal® Engine, Copyright 1998–2023, Epic Games, Inc. All rights reserved.

- Vehicle Dynamics Blockset uses fitted tire data sets provided by the Global Center for Automotive Performance Simulation (GCAPS). GCAPS uses advanced physical data collection to develop tire models that cover a broad range of vehicle and environmental conditions.

Required and Recommended Products

Required Products

Vehicle Dynamics Blockset product requires current versions of these products:

- MATLAB
- Simulink

Recommended Products

You can extend the capabilities of the Vehicle Dynamics Blockset using the following recommended products.

Goal	Recommended Products
Model events	Stateflow®
Test closed-loop perception, planning, and control algorithms	Automated Driving Toolbox™ RoadRunner
Test vehicle-level integration Optimize vehicle energy consumption, ride and handling	Powertrain Blockset™
Generate optimized suspension parameters	Model-Based Calibration Toolbox™ Simscape™ Multibody™

See Also

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

Engine Calibration Maps

Calibration maps are a key part of the Mapped CI Engine and Mapped SI Engine blocks available in the Vehicle Dynamics Blockset. Engine models use the maps to represent engine behavior and to store optimal control parameters. Using calibration maps in control design leads to flexible, efficient control algorithms and estimators that are suitable for electronic control unit (ECU) implementation.

To develop the calibration maps for engine plant models in the reference applications, MathWorks® developed and used processes to measure performance data from 1.5–L spark-ignition (SI) and compression-ignition (CI) engine models provided by Gamma Technologies LLC.

To represent the behavior of engine plants specific to your application, you can develop your own engine calibration maps. The data required for calibration typically comes from engine dynamometer tests or engine hardware design models.
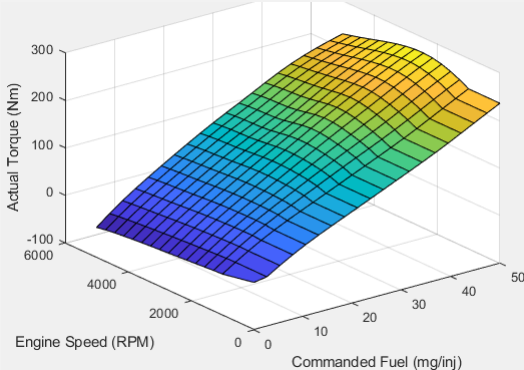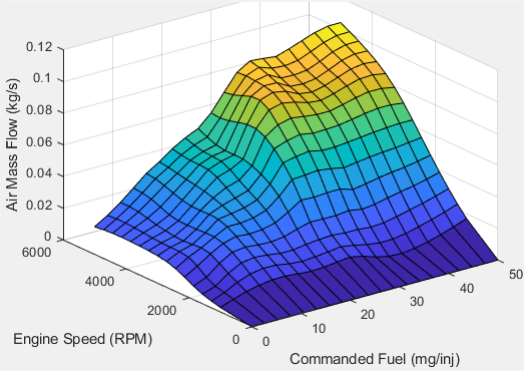
Engine Plant Calibration Maps
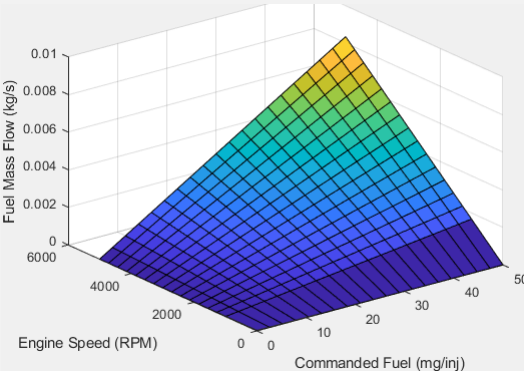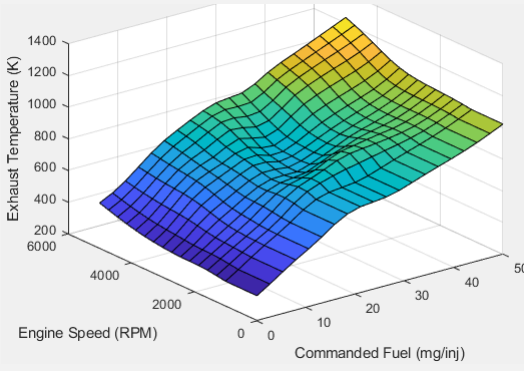
The engine plant model calibration maps in the Mapped CI Engine and Mapped SI Engine blocks affect the engine response to control inputs (for example, spark timing, throttle position, and cam phasing).
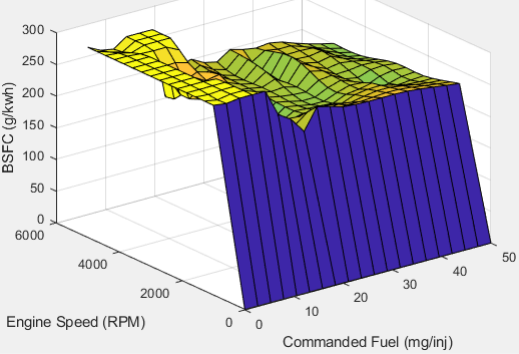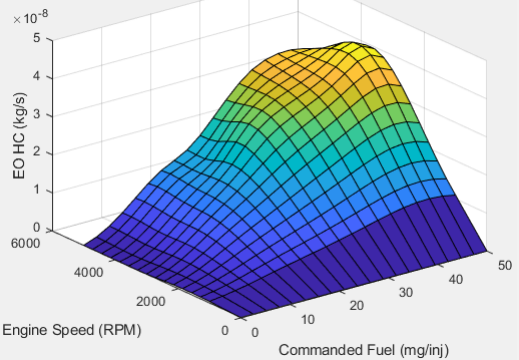
To develop the calibration maps in the engine plant models, MathWorks used GT-POWER models from the GT-SUITE modeling library in a Simulink-based virtual dynamometer. MathWorks used the Model-Based Calibration Toolbox to create design-of-experiment (DoE) test plans. The Simulink-based virtual dynamometer executed the DoE test plan on GT-POWER 1.5–L SI and CI reference engines. MathWorks used the Model-Based Calibration Toolbox to develop the engine plant model calibration maps from the GT-POWER.
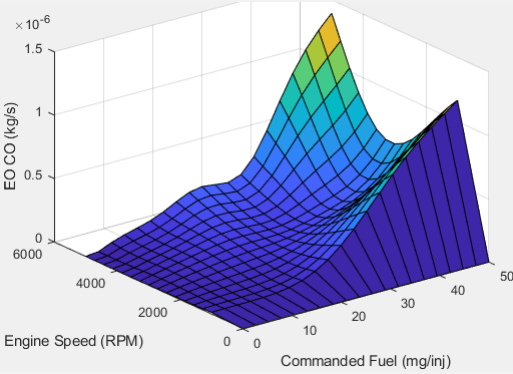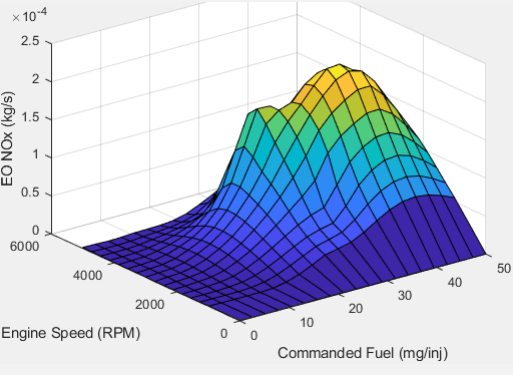
Calibration Maps in the Mapped CI Engine Block
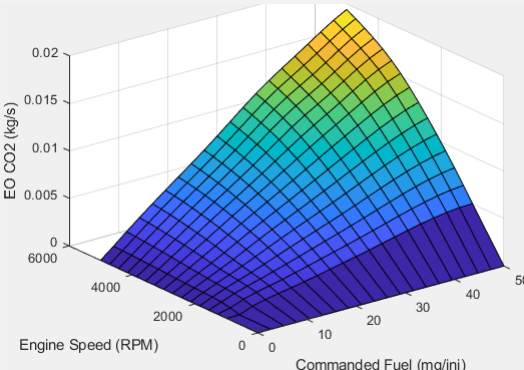
The Mapped CI Engine block implements these calibration maps.

Map	Used For	In	Description
Engine brake torque	Engine brake torque as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine brake torque lookup table is a function of commanded fuel mass and engine speed, $T_{brake} = f(F, N)$, where: • T_{brake} is engine torque, in N·m. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm.
Engine air mass flow	Engine air mass flow as a function of commanded fuel mass and engine speed	Mapped CI Engine	The air mass flow lookup table is a function of commanded fuel mass and engine speed, $\dot{m}_{intk} = f(F_{max}, N)$, where: • $\dot{m}_{intk}$ is engine air mass flow, in kg/s. • F_{max} is commanded fuel mass, in mg per injection. • N is engine speed, in rpm.

Map	Used For	In	Description
Engine fuel flow	Engine fuel flow as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine fuel flow lookup table is a function of commanded fuel mass and engine speed, *MassFlow*= ƒ(*F*, *N*), where: • *MassFlow* is engine fuel mass flow, in kg/s. • *F* is commanded fuel mass, in mg per injection. • *N* is engine speed, in rpm.
Engine exhaust temperature	Engine exhaust temperature as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine exhaust temperature table is a function of commanded fuel mass and engine speed, T_{exh}= ƒ(*F*, *N*), where: • T_{exh} is exhaust temperature, in K. • *F* is commanded fuel mass, in mg per injection. • *N* is engine speed, in rpm.
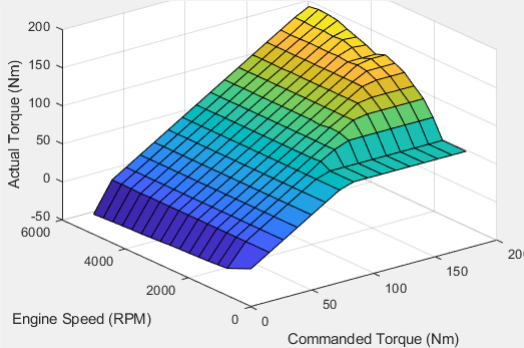
Map	Used For	In	Description
Brake-specific fuel consumption (BSFC) efficiency	BSFC efficiency as a function of commanded fuel mass and engine speed	Mapped CI Engine	The brake-specific fuel consumption (BSFC) efficiency is a function of commanded fuel mass and engine speed, $BSFC = f(F, N)$, where: • $BSFC$ is BSFC, in g/kWh. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm.
Engine-out (EO) hydrocarbon emissions	EO hydrocarbon emissions as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine-out hydrocarbon emissions are a function of commanded fuel mass and engine speed, $EO\ HC = f(F, N)$, where: • $EO\ HC$ is engine-out hydrocarbon emissions, in kg/s. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm.
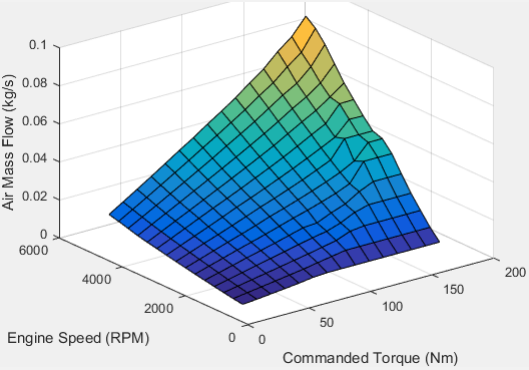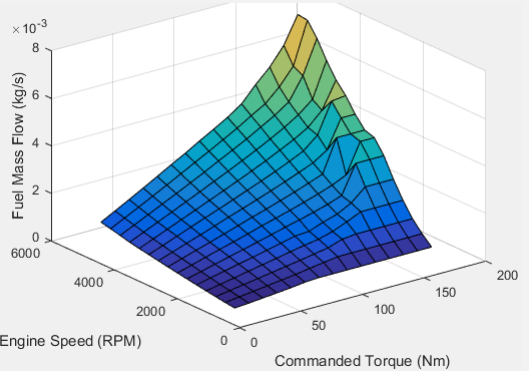
Map	Used For	In	Description
Engine-out (EO) carbon monoxide emissions	EO carbon monoxide emissions as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine-out carbon monoxide emissions are a function of commanded fuel mass and engine speed, *EO CO*= ƒ(*F*, *N*), where: • *EO CO* is engine-out carbon monoxide emissions, in kg/s. • *F* is commanded fuel mass, in mg per injection. • *N* is engine speed, in rpm.
Engine-out (EO) nitric oxide and nitrogen dioxide	EO nitric oxide and nitrogen dioxide emissions as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine-out nitric oxide and nitrogen dioxide emissions are a function of commanded fuel mass and engine speed, *EO NOx*= ƒ(*F*, *N*), where: • *EO NOx* is engine-out nitric oxide and nitrogen dioxide emissions, in kg/s. • *F* is commanded fuel mass, in mg per injection. • *N* is engine speed, in rpm.
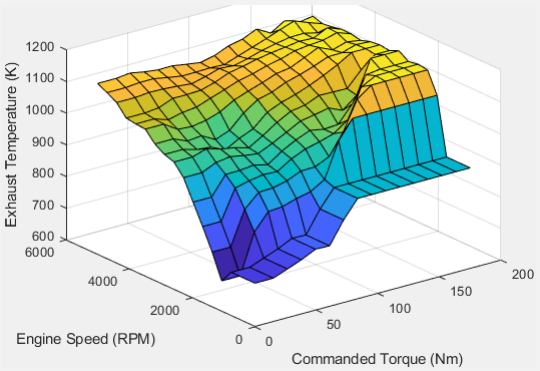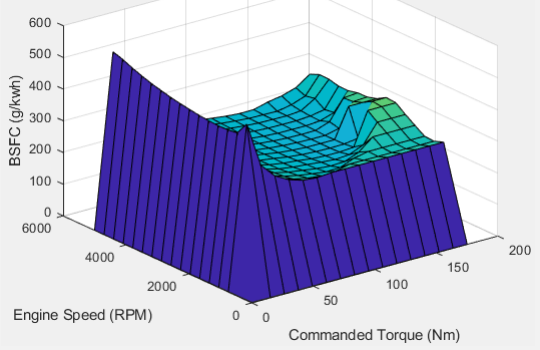
Map	Used For	In	Description
Engine-out (EO) carbon dioxide emissions	EO carbon dioxide emissions as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine-out carbon dioxide emissions are a function of commanded fuel mass and engine speed, $EO\ CO2 = f(F, N)$, where: • $EO\ CO2$ is engine-out carbon dioxide emissions, in kg/s. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm.
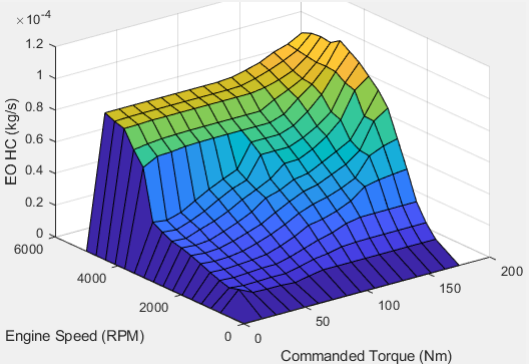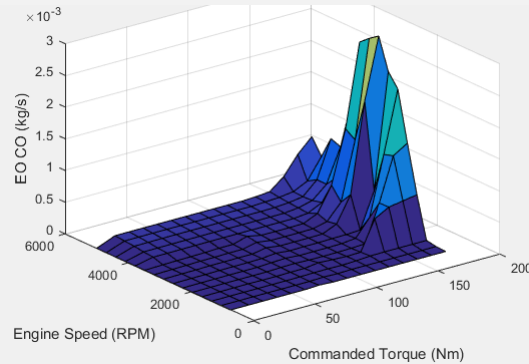
Calibration Maps in the Mapped SI Engine Block
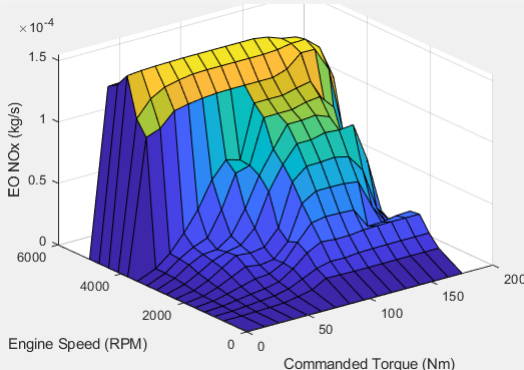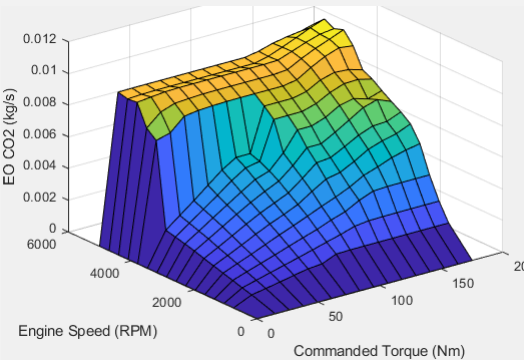
The Mapped SI Engine block implements these calibration maps.

Map	Used For	In	Description
Engine torque	Engine brake torque as a function of commanded torque and engine speed	Mapped SI Engine	The engine torque lookup table is a function of commanded engine torque and engine speed, $T = f(T_{cmd}, N)$, where: • T is engine torque, in N·m. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.

Map	Used For	In	Description
Engine air mass flow	Engine air mass flow as a function of commanded torque and engine speed	Mapped SI Engine	The engine air mass flow lookup table is a function of commanded engine torque and engine speed, $\dot{m}_{intk} = f(T_{cmd}, N)$, where: • $\dot{m}_{intk}$ is engine air mass flow, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.
Engine fuel flow	Engine fuel flow as a function of commanded torque mass and engine speed	Mapped SI Engine	The engine fuel mass flow lookup table is a function of commanded engine torque and engine speed, $MassFlow = f(T_{cmd}, N)$, where: • $MassFlow$ is engine fuel mass flow, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.

Map	Used For	In	Description
Engine exhaust temperature	Engine exhaust temperature as a function of commanded torque and engine speed	Mapped SI Engine	The engine exhaust temperature lookup table is a function of commanded engine torque and engine speed, $T_{exh} = f(T_{cmd}, N)$, where: • T_{exh} is exhaust temperature, in K. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.
Brake-specific fuel consumption (BSFC) efficiency	Brake-specific fuel consumption (BSFC) as a function of commanded torque and engine speed	Mapped SI Engine	The brake-specific fuel consumption (BSFC) efficiency is a function of commanded engine torque and engine speed, $BSFC = f(T_{cmd}, N)$, where: • $BSFC$ is BSFC, in g/kWh. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.

Map	Used For	In	Description
Engine-out (EO) hydrocarbon emissions	EO hydrocarbon emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out hydrocarbon emissions are a function of commanded engine torque and engine speed, *EO HC* = $\mathfrak{f}(T_{cmd}, N)$, where: • *EO HC* is engine-out hydrocarbon emissions, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • *N* is engine speed, in rpm.
Engine-out (EO) carbon monoxide emissions	EO carbon monoxide emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out carbon monoxide emissions are a function of commanded engine torque and engine speed, *EO CO* = $\mathfrak{f}(T_{cmd}, N)$, where: • *EO CO* is engine-out carbon monoxide emissions, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • *N* is engine speed, in rpm.

Map	Used For	In	Description
Engine-out (EO) nitric oxide and nitrogen dioxide emissions	EO nitric oxide and nitrogen dioxide emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out nitric oxide and nitrogen dioxide emissions are a function of commanded engine torque and engine speed, $EO\ NOx = f(T_{cmd}, N)$, where: • $EO\ NOx$ is engine-out nitric oxide and nitrogen dioxide emissions, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.
Engine-out (EO) carbon dioxide emissions	EO carbon dioxide emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out carbon dioxide emissions are a function of commanded engine torque and engine speed, $EO\ CO2 = f(T_{cmd}, N)$, where: • $EO\ CO2$ is engine-out carbon dioxide emissions, in kg/s. • T_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm.

See Also

Mapped CI Engine | Mapped SI Engine

External Websites

- Virtual Engine Calibration: Making Engine Calibration Part of the Engine Hardware Design Process

Yaw Stability on Varying Road Surfaces

This example shows how to run the vehicle dynamics double-lane change maneuver on different road surfaces, analyze the vehicle yaw stability, and determine the maneuver success.

ISO 3888-2 defines the double-lane change maneuver to test the obstacle avoidance performance of a vehicle. In the test, the driver:

- Accelerates until vehicle hits a target velocity
- Releases the accelerator pedal
- Turns steering wheel to follow path into the left lane
- Turns steering wheel to follow path back into the right lane

Typically, cones mark the lane boundaries. If the vehicle and driver can negotiate the maneuver without hitting a cone, the vehicle passes the test.

For more information about the reference application, see "Double-Lane Change Maneuver" on page 3-22.

```
helpersetupdlc;
```

Run a Double-Lane Change Maneuver

1. Open the Lane Change Reference Generator block. By default, the maneuver is set with these parameters:

- **Longitudinal entrance velocity setpoint** — 35 mph
- **Vehicle width** — 2 m
- **Lateral reference position breakpoints** and **Lateral reference data** — Values that specify the lateral reference trajectory as a function of the longitudinal distance

2. In the Visualization subsystem, open the 3D Engine block. By default, the **3D Engine** parameter is set to **Disabled**. For the 3D visualization engine platform requirements and hardware recommendations, see the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

3. Run the maneuver. As the simulation runs, view the vehicle information.

```
mdl = 'DLCReferenceApplication';
sim(mdl);

### Starting serial model reference simulation build.
### Model reference simulation target for Driveline is up to date.
### Model reference simulation target for PassVeh14DOF is up to date.
### Model reference simulation target for SiMappedEngineV is up to date.

Build Summary

0 of 3 models built (3 models already up to date)
Build duration: 0h 0m 5.7447s
```
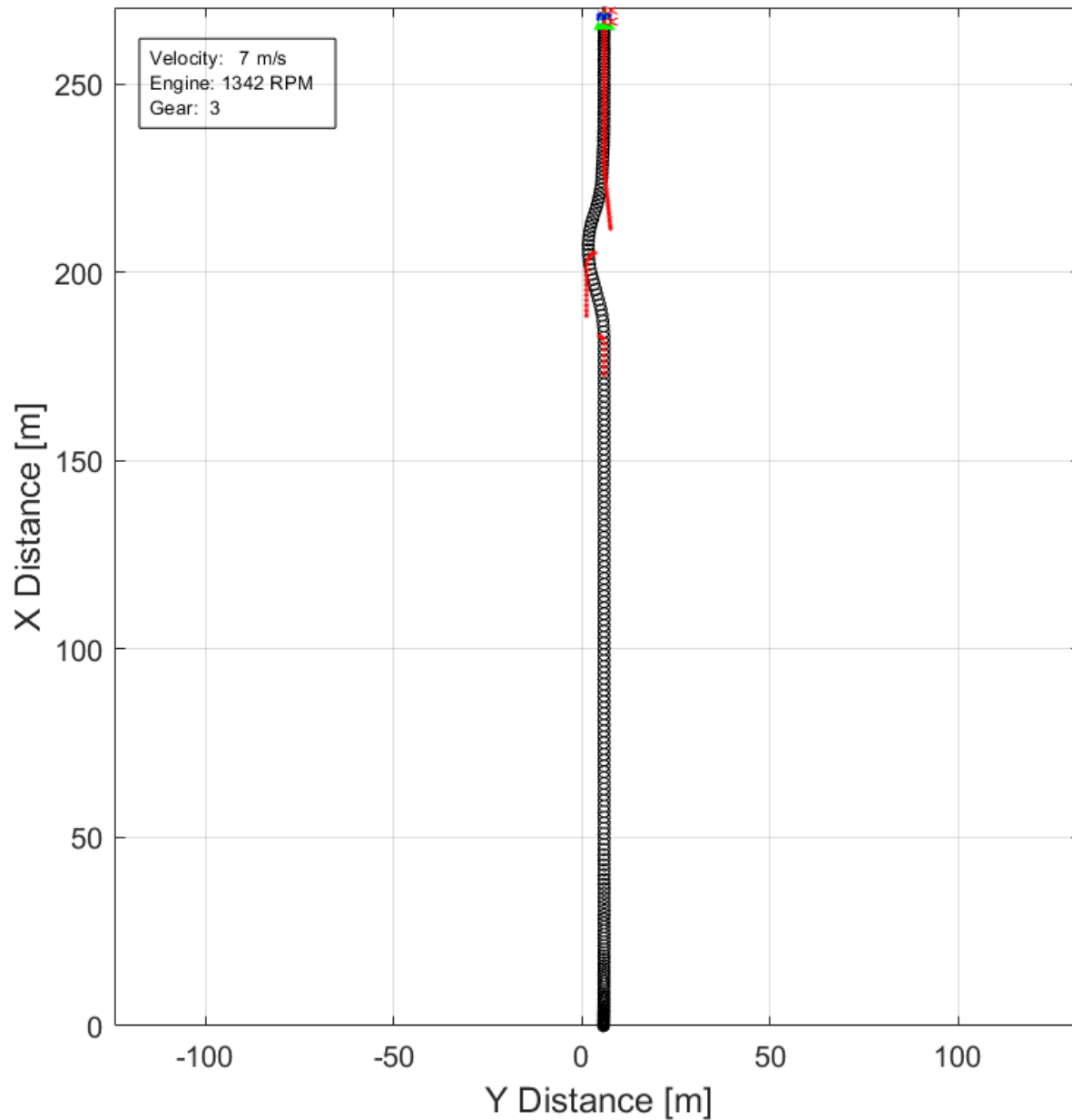
- In the Vehicle Position window, view the vehicle longitudinal distance as a function or the lateral distance.

- In the Visualization subsystem, open the Lane Change scope block to display the lateral displacement as a function of time. The red and orange lines mark the cone boundaries. The blue line marks the reference trajectory and the green line marks the actual trajectory. The green line does come close to the red line that marks the cones.
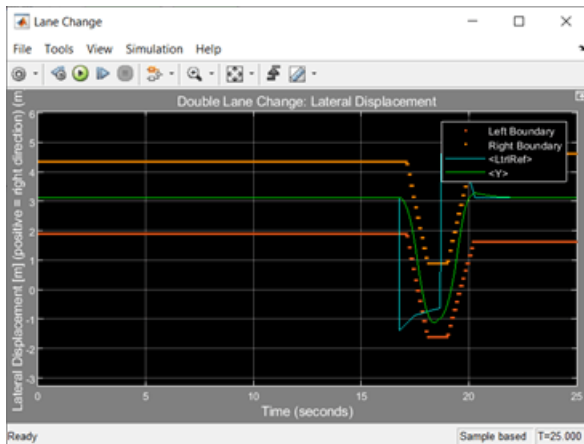
- In the Visualization subsystem, if you enable the 3D Engine block visualization environment, you can view the vehicle response in the `AutoVrtlEnv` window.

Sweep Surface Friction

Run the reference application on three road surfaces with different friction scaling coefficients. Use the results to analyze the yaw stability and help determine the success of the maneuver.

1. In the double-lane change reference application model DLCReferenceApplication, open the Environment subsystem. The Friction block parameter **Constant value** specifies the friction scaling coefficient. By default, the friction scaling coefficient is `1.0`. The reference application uses the coefficient to adjust the friction at every time step.

2. Enable signal logging for the velocity, lane, and ISO signals. You can use the Simulink® editor or, alternatively, these MATLAB® commands. Save the model.

- Enable signal logging for the Lane Change Reference Generator outport Lane signal.
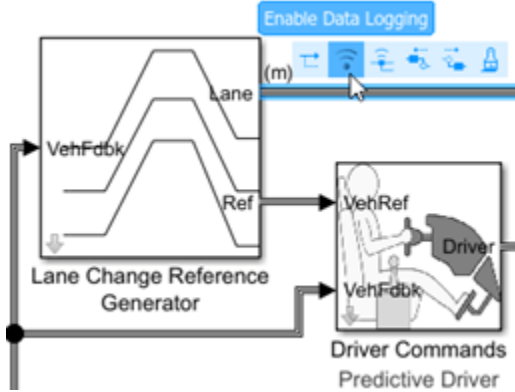
```
mdl = 'DLCReferenceApplication';
ph=get_param('DLCReferenceApplication/Lane Change Reference Generator','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- Enable signal logging for the Passenger Vehicle block outport signal.


```
ph=get_param('DLCReferenceApplication/Passenger Vehicle','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```
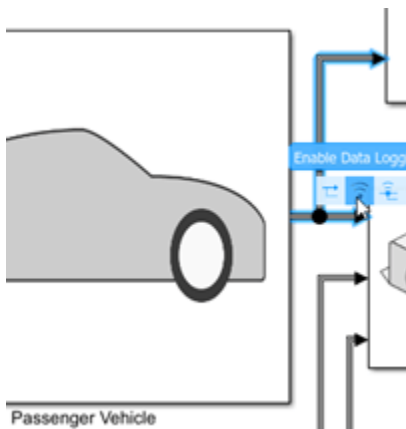
- In the Visualization subsystem, enable signal logging for the ISO block.


```
set_param([mdl '/Visualization/ISO 15037-1:2006'],'Measurement','Enable');
```

3. Set up a vector with the friction scaling coefficients, lambdamu, that you want to investigate. For example, to examine friction scaling coefficients equal to 0.9, 0.95, and 1.0, at the command line enter:

1-19

```
lambdamu = [0.9, 0.95, 1.0];
numExperiments = length(lambdamu);
```

4. Create an array of simulation inputs that sets `lambdamu` equal to the Friction constant block parameter.

```
for idx = numExperiments:-1:1
    in(idx) = Simulink.SimulationInput(mdl);
    in(idx) = in(idx).setBlockParameter([mdl '/Environment/Friction'],...
        'Value',['ones(4,1).*',num2str(lambdamu(idx))]);
end
```

5. Set the simulation stop time at 25 s. Save the model and run the simulations. If available, use parallel computing.

```
set_param(mdl,'StopTime','25')
save_system(mdl)
tic;
simout = parsim(in,'ShowSimulationManager','on');
toc;

[05-Jan-2023 09:06:25] Checking for availability of parallel pool...
[05-Jan-2023 09:06:25] Starting Simulink on parallel workers...
[05-Jan-2023 09:06:27] Loading project on parallel workers...
[05-Jan-2023 09:06:27] Configuring simulation cache folder on parallel workers...
[05-Jan-2023 09:06:27] Loading model on parallel workers...
[05-Jan-2023 09:06:56] Running simulations...
[05-Jan-2023 09:08:45] Completed 1 of 3 simulation runs
[05-Jan-2023 09:08:45] Completed 2 of 3 simulation runs
[05-Jan-2023 09:08:46] Completed 3 of 3 simulation runs
[05-Jan-2023 09:08:46] Cleaning up parallel workers...
Elapsed time is 152.193850 seconds.
```

6. After the simulations complete, close the Simulation Data Inspector windows.

Use Simulation Data Inspector to Analyze Results

Use the Simulation Data Inspector to examine the results. You can use the UI or, alternatively, command-line functions.

1. Open the Simulation Data Inspector. On the Simulink Toolstrip, on the **Simulation** tab, under **Review Results**, click **Data Inspector**.

- In the Simulation Data Inspector, select **Import**.

- In the **Import** dialog box, clear `logsout`. Select `simout(1)`, `simout(2)`, and `simout(3)`. Select **Import**.

- Use the Simulation Data Inspector to examine the results.

2. Alternatively, use these MATLAB commands to create 6 plots. The first three plots mark the upper lane boundary, UB, lower lane boundary, LB, and lateral vehicle distance, Y, for each run.

The next three plots provide the lateral acceleration, ay, lateral vehicle distance, Y, and yaw rate, r, for each run.

```
for idx = 1:numExperiments
    % Create sdi run object
    simoutRun(idx)=Simulink.sdi.Run.create;
    simoutRun(idx).Name=['lambdamu = ', num2str(lambdamu(idx))];
    add(simoutRun(idx),'vars',simout(idx));
end
sigcolor=[1 0 0];
for idx = 1:numExperiments
    % Extract the maneuver upper and lower lane boundaries
    ubsignal(idx)=getSignalsByName(simoutRun(idx), 'Lane Change Reference Generator:1.LeftBnd');
    ubsignal(idx).LineColor = sigcolor;
    lbsignal(idx)=getSignalsByName(simoutRun(idx), 'Lane Change Reference Generator:1.RightBnd');
    lbsignal(idx).LineColor = sigcolor;
end
sigcolor=[0 1 0;0 0 1;1 0 1];
for idx = 1:numExperiments
    % Extract the lateral acceleration, position, and yaw rate
    ysignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y');
    ysignal(idx).LineColor =sigcolor((idx),:);
    rsignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.BdyFrm.Cg.AngVel.r');
    rsignal(idx).LineColor =sigcolor((idx),:);
    asignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.BdyFrm.Cg.Acc.ay');
    asignal(idx).LineColor =sigcolor((idx),:);
end
Simulink.sdi.view
Simulink.sdi.setSubPlotLayout(numExperiments,2);
for idx = 1:numExperiments
    %  Plot the lateral position and lane boundaries
    plotOnSubPlot(ubsignal(idx),(idx),1,true);
```

```
        plotOnSubPlot(lbsignal(idx),(idx),1,true);
        plotOnSubPlot(ysignal(idx),(idx),1,true);
    end
    for idx = 1:numExperiments
        % Plot the lateral acceleration, position, and yaw rate
        plotOnSubPlot(asignal(idx),1,2,true);
        plotOnSubPlot(ysignal(idx),2,2,true);
        plotOnSubPlot(rsignal(idx),3,2,true);
    end
```

The results are similar to these plots, which indicate that the vehicle has a yaw rate of about .66 rad/s when the friction scaling coefficient is equal to 1.
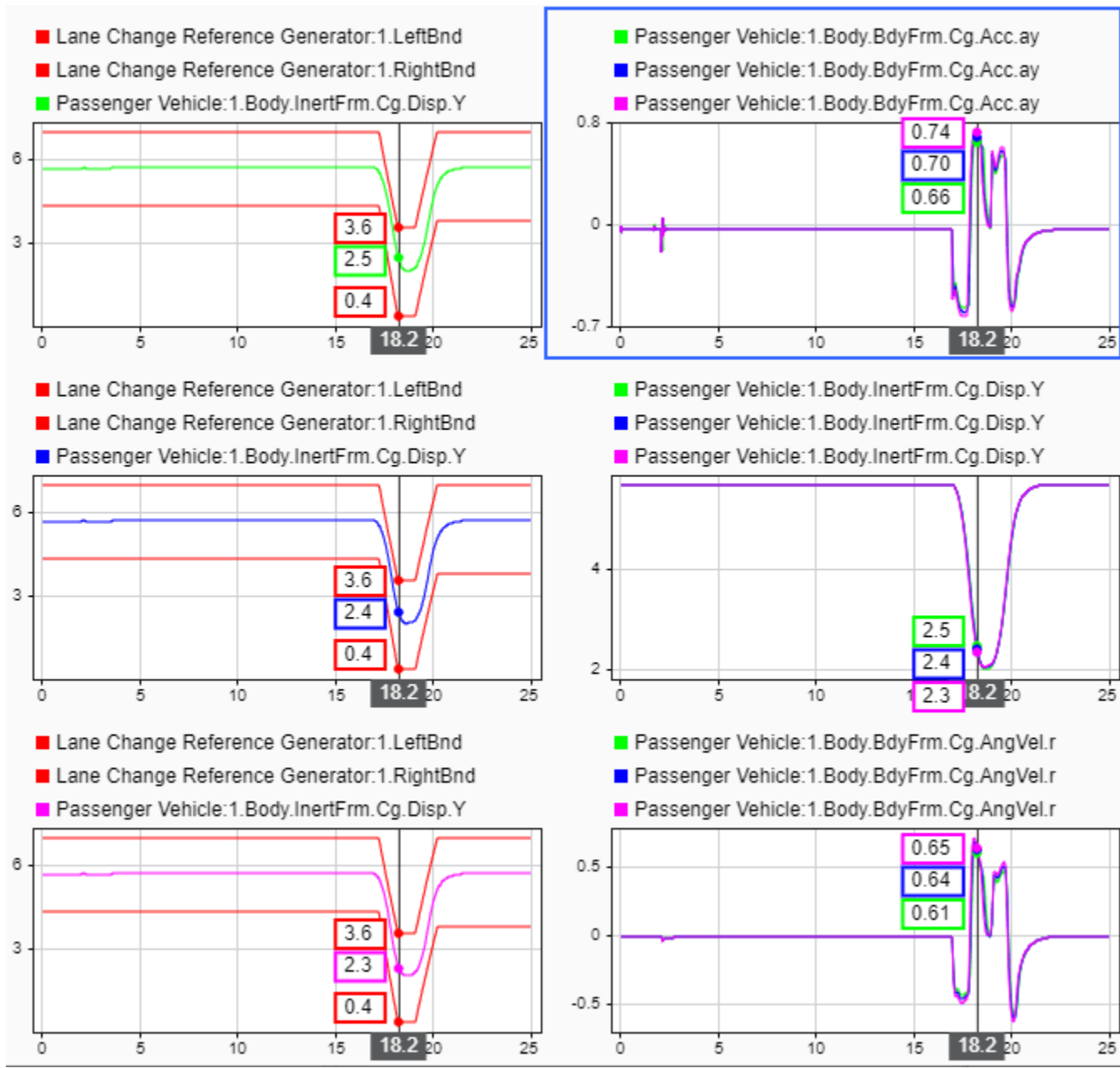
Further Analysis

To explore the results further, use these commands to extract the lateral acceleration, steering angle, and vehicle trajectory from the `simout` object.

1. Extract the lateral acceleration and steering angle. Plot the data. The results are similar to this plot. They indicate that the greatest lateral acceleration occurs when the friction scaling coefficient is 1.

```matlab
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    sa=log.get('Steering-wheel angle').Values;
    ay=log.get('Lateral acceleration').Values;
    legend_labels{idx} = ['lambdamu = ', num2str(lambdamu(idx))];
    % Plot steering angle vs. lateral acceleration
    plot(sa.Data,ay.Data)
    hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Lateral Acceleration')
xlabel('Steering Angle [deg]')
ylabel('Acceleration [m/s^2]')
grid on
```
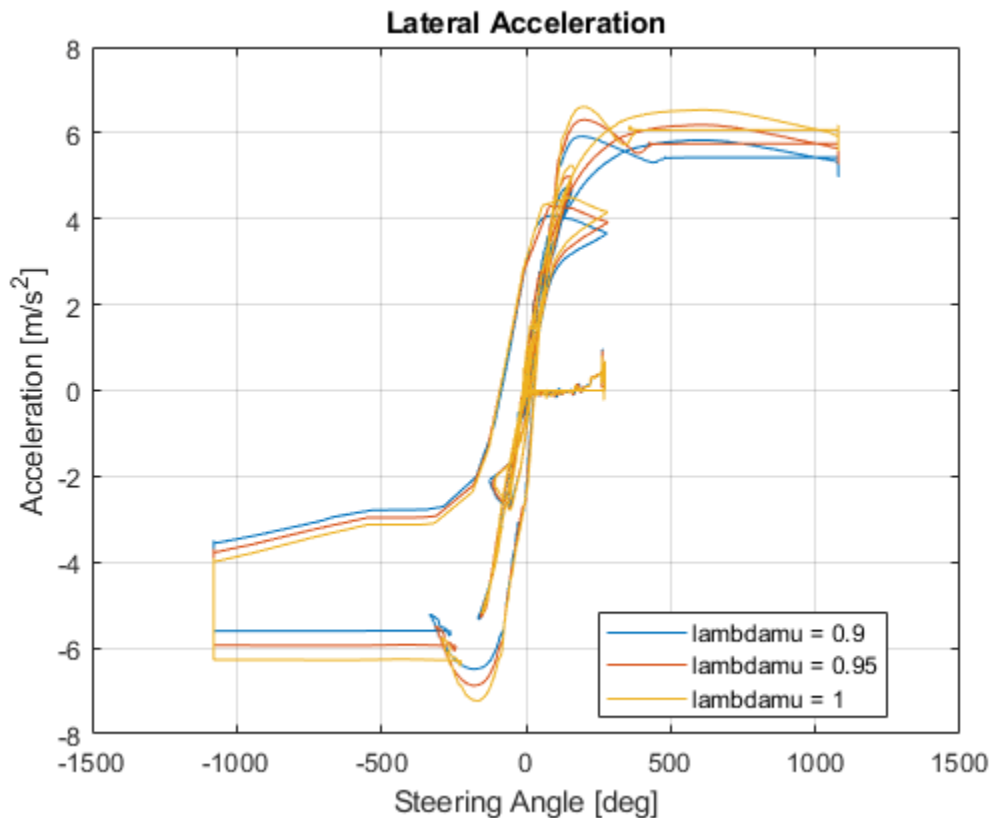
2. Extract the vehicle path. Plot the data. The results are similar to this plot. They indicate that the greatest lateral vehicle position occurs when the friction scaling coefficient is `0.9`.

```matlab
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    xValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.X').Va`
    yValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y').Va`
    x = xValues.Data;
    y = yValues.Data;
    legend_labels{idx} = ['lambdamu = ', num2str(lambdamu(idx))];
    % Plot vehicle location
    plot(y,x)
    hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Vehicle Path')
xlabel('Y Position [m]')
ylabel('X Position [m]')
grid on
```
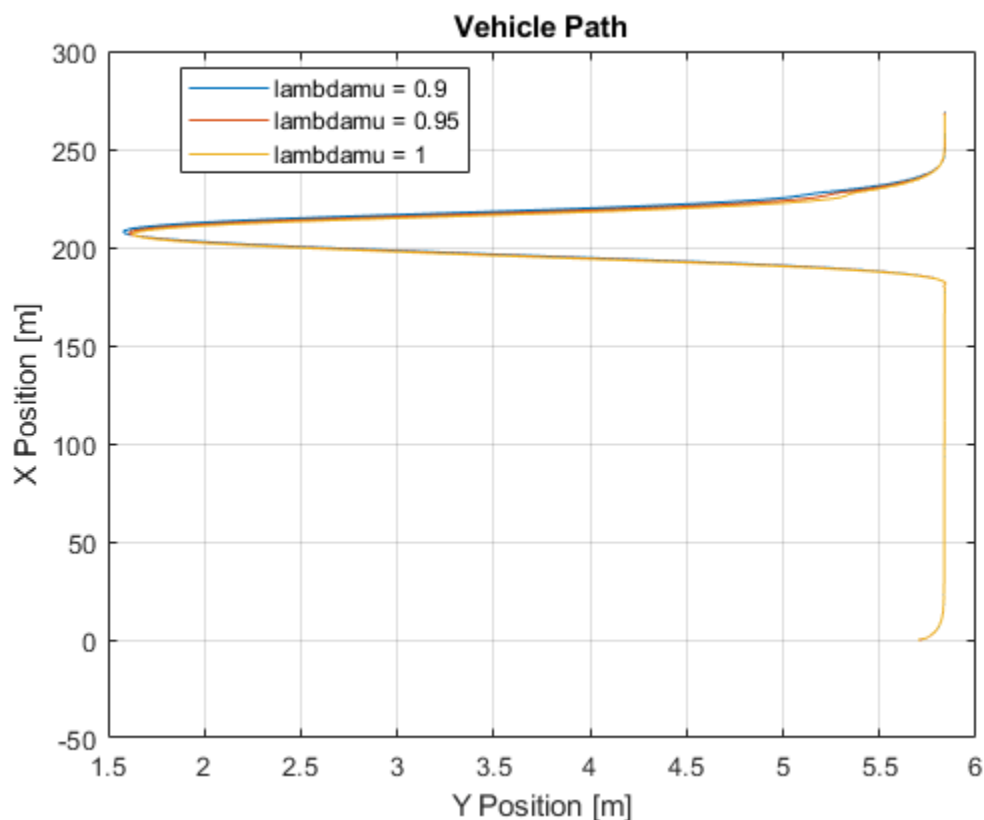

See Also

`Simulink.SimulationInput` | `Simulink.SimulationOutput`

References

[1] ISO 3888-2: 2011. *Passenger cars — Test track for a severe lane-change manoeuvre.*

See Also

Related Examples

- "Send and Receive Double-Lane Change Scene Data" on page 3-93

More About

- "Double-Lane Change Maneuver" on page 3-22
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8
- Simulation Data Inspector

Vehicle Steering Gain at Different Speeds

This example shows how to use the vehicle dynamics slowly increasing steering reference application to analyze the impact of the steering angle and speed on vehicle handling. Specifically, you can calculate the steering gain when you run the maneuver with different speed set points. Based on the constant speed, variable steer test defined in SAE J266, the slowly increasing steering maneuver helps characterize the lateral dynamics of the vehicle. In the test, the driver:

- Accelerates until vehicle hits a target velocity.
- Maintains a target velocity.
- Linearly increases the steering wheel angle from 0 degrees to a maximum angle.
- Maintains the steering wheel angle for a specified time.
- Linearly decreases the steering wheel angle from maximum angle to 0 degrees.

For more information about the reference application, see "Slowly Increasing Steering Maneuver" on page 3-53.

```
helpersetupsis;
```
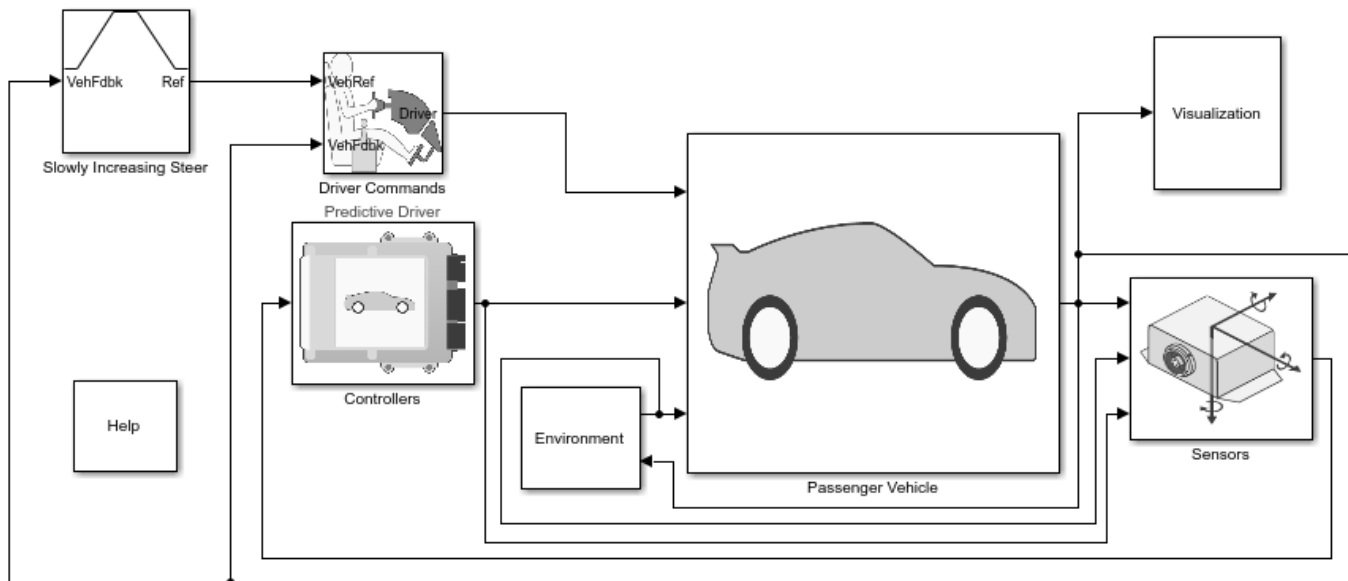

Copyright 2018-2022 The MathWorks, Inc.

Run a Slowly Increasing Steering Maneuver

1. Open the Slowly Increasing Steer block. By default, the maneuver is set with these parameters:

- **Longitudinal speed setpoint** — 50 mph
- **Handwheel rate** — 13.5 deg
- **Maximum handwheel angle** — 270 deg

2. In the Visualization subsystem, open the 3D Engine block. By default, the **3D Engine** parameter is set to **Disabled**. For the 3D visualization engine platform requirements and hardware

recommendations, see the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

3. Run the maneuver with the default settings. As the simulation runs, view the vehicle information.

```
mdl = 'ISReferenceApplication';
sim(mdl);

### Starting serial model reference simulation build.
### Model reference simulation target for Driveline is up to date.
### Model reference simulation target for PassVeh14DOF is up to date.
### Model reference simulation target for SiMappedEngineV is up to date.

Build Summary

0 of 3 models built (3 models already up to date)
Build duration: 0h 0m 4.7948s
```
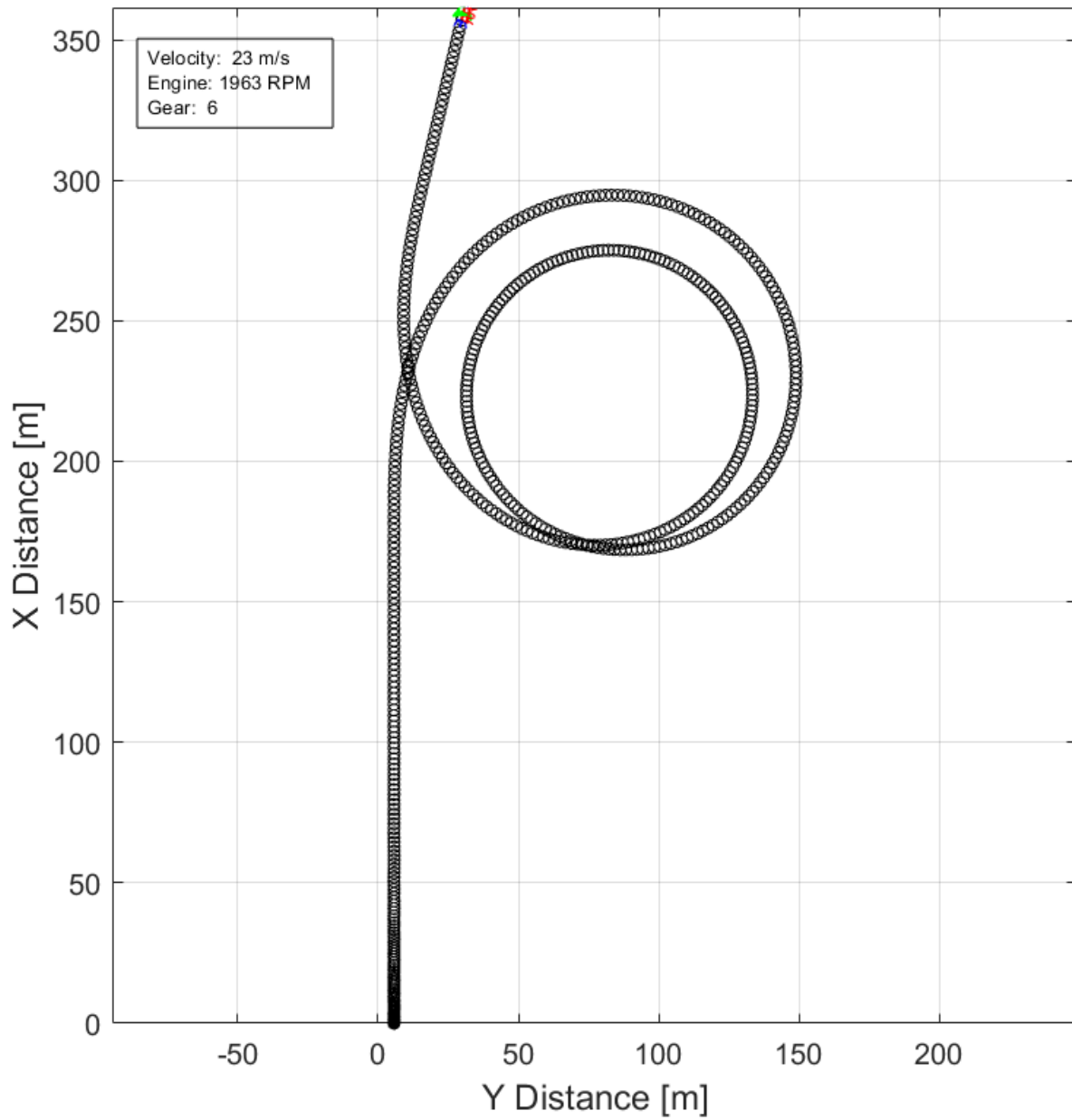
Velocity: 23 m/s
Engine: 1963 RPM
Gear: 6

- In the Vehicle Position window, view the vehicle longitudinal distance as a function or the lateral distance. The yellow line displays the yaw rate. The blue line shows the steering angle.

- In the Visualization subsystem, open the Yaw Rate and Steer Scope block to display the yaw rate and steering angle versus time.

Sweep Speed Set Points

Run the slowly increasing steering angle reference application with three different speed set points.

1. In the slowly increasing steering reference application model ISReferenceApplication, open the Slowly Increasing Steer block. The **Longitudinal speed set point, xdot_r** block parameter sets the vehicle speed. By default, the speed is 50 mph.

2. Enable signal logging for the velocity, lane, and ISO signals. You can use the Simulink® editor or, alternatively, these MATLAB® commands. Save the model.

- Enable signal logging for the Slowly Increasing Steer Ref signal outport.

```
mdl = 'ISReferenceApplication';
ph=get_param('ISReferenceApplication/Slowly Increasing Steer','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- Enable signal logging for the Passenger Vehicle block outport signal.


```
ph=get_param('ISReferenceApplication/Passenger Vehicle','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- In the Visualization subsystem, enable signal logging for the ISO block.


```
set_param([mdl '/Visualization/ISO 15037-1:2006'],'Measurement','Enable');
```
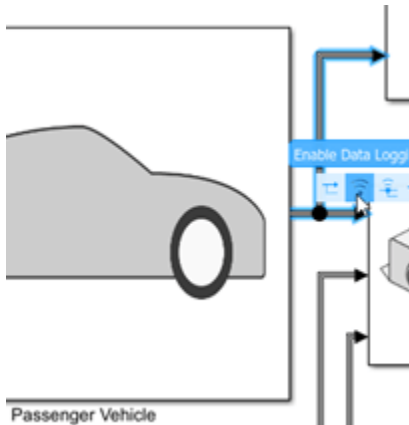
3. Set up a speed set point vector, `xdot_r`, that you want to investigate. For example, at the command line, type:

```
vmax = [45, 50, 55];
numExperiments = length(vmax);
```

4. Create an array of simulation inputs that set the Slowly Increasing Steer block parameter **Longitudinal speed setpoint, xdot_r** equal to `xdot_r`.

```
for idx = numExperiments:-1:1
    in(idx) = Simulink.SimulationInput(mdl);
    in(idx) = in(idx).setBlockParameter([mdl '/Slowly Increasing Steer'], ...
        'xdot_r', num2str(vmax(idx)));
end
```

5. Save the model and run the simulations. If available, use parallel computing.

```
save_system(mdl)
tic;
```

```
simout = parsim(in,'ShowSimulationManager','on');
toc;

[05-Jan-2023 11:33:33] Checking for availability of parallel pool...
[05-Jan-2023 11:33:33] Starting Simulink on parallel workers...
[05-Jan-2023 11:33:34] Loading project on parallel workers...
[05-Jan-2023 11:33:34] Configuring simulation cache folder on parallel workers...
[05-Jan-2023 11:33:34] Loading model on parallel workers...
[05-Jan-2023 11:33:51] Running simulations...
[05-Jan-2023 11:34:57] Completed 1 of 3 simulation runs
[05-Jan-2023 11:34:59] Completed 2 of 3 simulation runs
[05-Jan-2023 11:36:44] Completed 3 of 3 simulation runs
[05-Jan-2023 11:36:44] Cleaning up parallel workers...
Elapsed time is 201.857620 seconds.
```

6. After the simulations complete, close the Simulation Data Inspector windows.

Use Simulation Data Inspector to Analyze Results

Use the Simulation Data Inspector to examine the results. You can use the UI or, alternatively, command-line functions.

1. Open the Simulation Data Inspector. On the Simulink Toolstrip, on the **Simulation** tab, under **Review Results**, click **Data Inspector**.

- In the Simulation Data Inspector, select **Import**.

- In the **Import** dialog box, clear `logsout`. Select `simout(1)`, `simout(2)`, and `simout(3)`. Select **Import**.

- Use the Simulation Data Inspector to examine the results.

2. Alternatively, use these MATLAB commands to plot the longitudinal velocity, steering wheel angle, lateral acceleration, longitudinal position, and lateral position.

```matlab
for idx = 1:numExperiments
    % Create sdi run object
    simoutRun(idx)=Simulink.sdi.Run.create;
    simoutRun(idx).Name=['Velocity = ', num2str(vmax(idx))];
    add(simoutRun(idx),'vars',simout(idx));
end
sigcolor=[0 1 0;0 0 1;1 0 1];
for idx = 1:numExperiments
    % Extract the lateral acceleration, position, and steering
    msignal(idx)=getSignalsByName(simoutRun(idx), 'xdot_mph');
    msignal(idx).LineColor =sigcolor((idx),:);
    ssignal(idx)=getSignalsByName(simoutRun(idx), 'SteerAngle');
    ssignal(idx).LineColor =sigcolor((idx),:);
    asignal(idx)=getSignalsByName(simoutRun(idx), 'Lateral acceleration');
    asignal(idx).LineColor =sigcolor((idx),:);
    xsignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.X')
    xsignal(idx).LineColor =sigcolor((idx),:);
    ysignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y')
    ysignal(idx).LineColor =sigcolor((idx),:);
end
Simulink.sdi.view
Simulink.sdi.setSubPlotLayout(5,1);
for idx = 1:numExperiments
    % Plot the lateral position, steering angle, and lateral acceleration
    plotOnSubPlot(msignal(idx),1,1,true);
    plotOnSubPlot(ssignal(idx),2,1,true);
    plotOnSubPlot(asignal(idx),3,1,true);
    plotOnSubPlot(xsignal(idx),4,1,true);
    plotOnSubPlot(ysignal(idx),5,1,true);
end
```

The results are similar to these plots, which indicate that the greatest lateral acceleration occurs when the vehicle velocity is 45 mph.

Further Analysis

To explore the results further, use these commands to extract the lateral acceleration, steering angle, and vehicle trajectory from the `simout` object.

1. Extract the lateral acceleration and steering angle. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    sa=log.get('Steering-wheel angle').Values;
    ay=log.get('Lateral acceleration').Values;
```

```
firstorderfit = polyfit(sa.Data,ay.Data,1);
gain(idx)=firstorderfit(1);
legend_labels{idx} = [num2str(vmax(idx)), ' mph: Gain = ', ...
    num2str(gain(idx)), ' m/(deg s^2)'];
% Plot steering angle vs. lateral acceleration
plot(sa.Data,ay.Data)
hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Lateral Acceleration')
xlabel('Steering Angle [deg]')
ylabel('Acceleration [m/s^2]')
grid on
```


2. Extract the vehicle path. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    xValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.X').Va
    yValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y').Va
    x = xValues.Data;
    y = yValues.Data;
    legend_labels{idx} = [num2str(vmax(idx)), ' mph'];
    % Plot vehicle location
    axis('equal')
```

```
    plot(y,x)
    hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Vehicle Path')
xlabel('Y Position [m]')
ylabel('X Position [m]')
grid on
```


References

[1] SAE J266. *Steady-State Directional Control Test Procedures For Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

See Also

Simulink.SimulationInput | Simulink.SimulationOutput | polyfit

More About

*   "Slowly Increasing Steering Maneuver" on page 3-53
*   "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8
*   Simulation Data Inspector

Vehicle Lateral Acceleration at Different Speeds

This example shows how to use the vehicle dynamics constant radius reference application to analyze the impact of speed on the vehicle lateral dynamics. Specifically, you can examine the lateral acceleration when you run the maneuver with different speeds. For information about similar maneuvers, see standards SAE J266_199601 and ISO 4138:2012.

During the maneuver, the vehicle uses a predictive driver model to maintain a pre-specified turn radius at a set velocity.

For more information about the reference application, see "Constant Radius Maneuver" on page 3-65.

```
helpersetupcr;
```
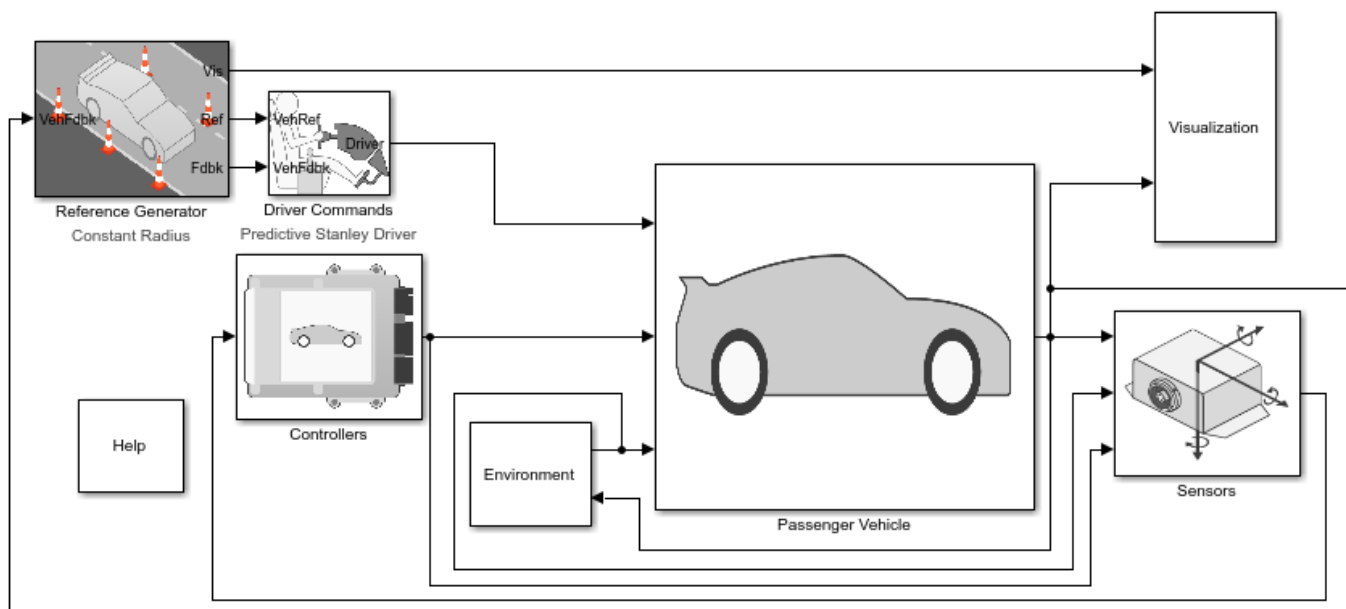

Copyright 2018-2022 The MathWorks, Inc.

Run a Constant Radius Maneuver

1. Open the Reference Generator block. By default, the maneuver is set with these parameters:

- **Maneuver** — Constant radius
- **Use maneuver-specific driver, initial position, and scene** — on
- **Longitudinal velocity reference** — 35 mph
- **Radius value** — 100 m

2. Select the Reference Generator block 3D Engine tab. By default, the 3D Engine parameter is **Disabled**. For the 3D visualization engine platform requirements and hardware recommendations, see the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

3. Run the maneuver with the default settings. As the simulation runs, view the vehicle information.

```
mdl = 'CRReferenceApplication';
sim(mdl);

### Starting serial model reference simulation build.
### Model reference simulation target for Driveline is up to date.
### Model reference simulation target for PassVeh14DOF is up to date.
### Model reference simulation target for SiMappedEngineV is up to date.

Build Summary

0 of 3 models built (3 models already up to date)
Build duration: 0h 0m 14.186s
```
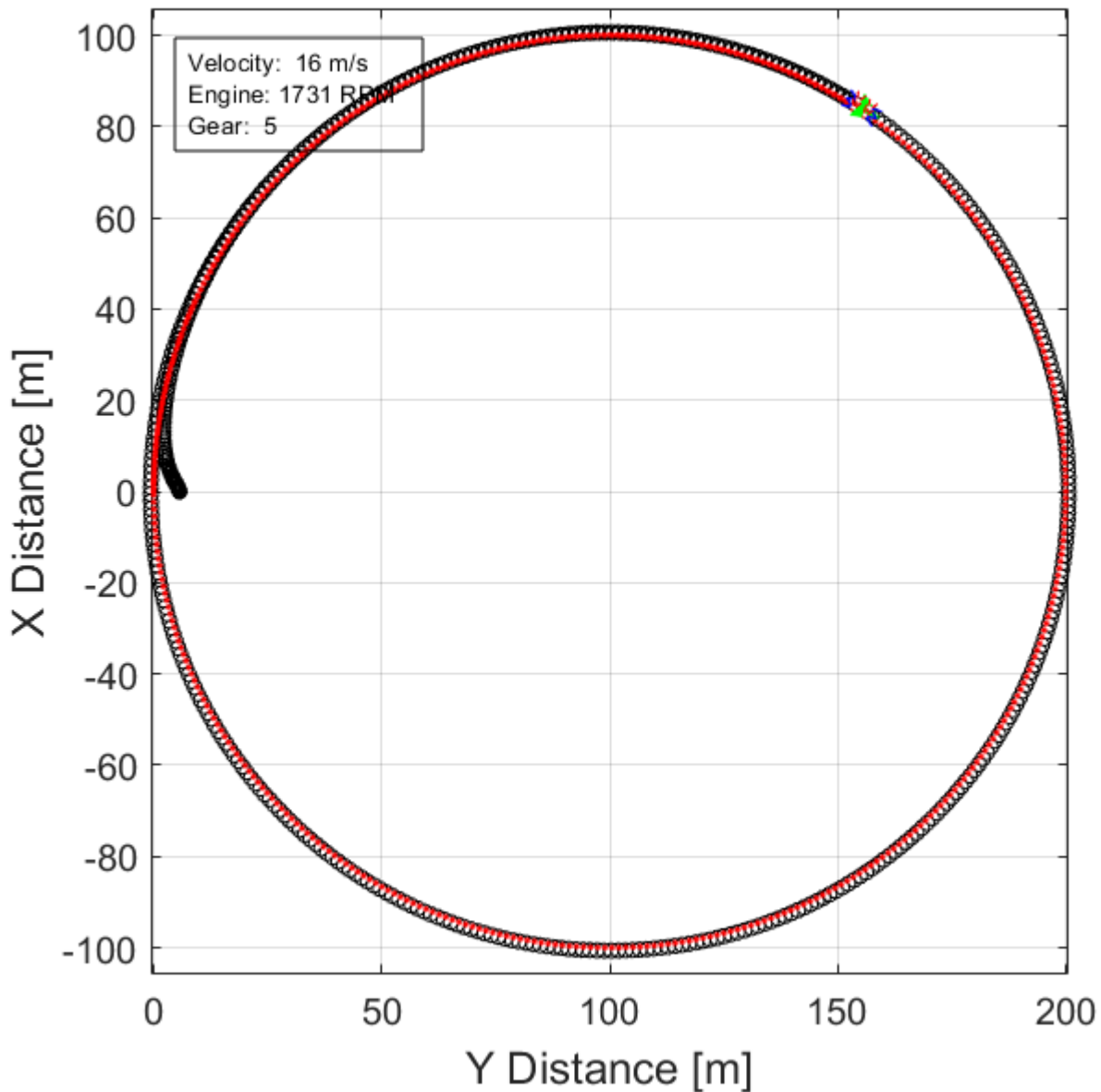
- In the Vehicle Position window, view the vehicle longitudinal distance as a function or the lateral distance. The yellow line displays the yaw rate. The blue line shows the steering angle.

- In the Visualization subsystem, open the Steer, Velocity, Lat Accel Scope block to display the steering angle, velocity, and lateral acceleration versus time.
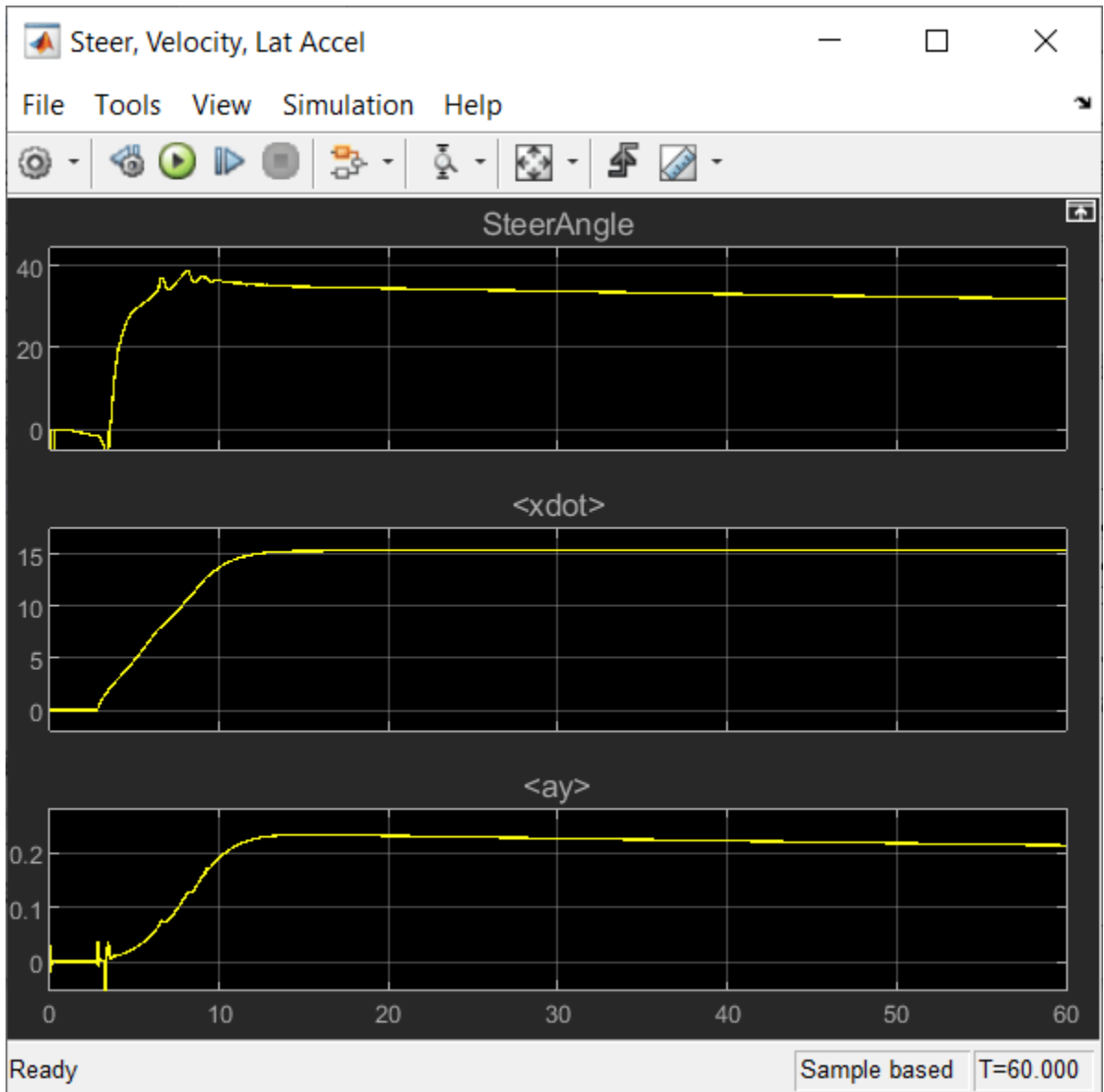
Sweep Speed

Run the constant radius reference application with three different speeds. Stop the simulation if the vehicle exceeds a lateral acceleration threshold of .5 g.

1. In the constant radius reference application model CRReferenceApplication, open the Reference Generator block. The **Longitudinal speed set point, xdot_r** block parameter sets the vehicle speed. By default, the speed is 50 mph.

2. Enable signal logging for the velocity, lane, and ISO signals. You can use the Simulink® editor or, alternatively, these MATLAB® commands. Save the model.

- Select the Reference Generator block **Stop simulation at lateral acceleration threshold** parameter.

Lateral acceleration threshold, ay_max [g]: 0.5

☑ Stop simulation at lateral acceleration threshold
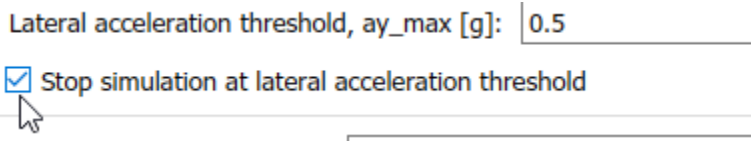
```
set_param([mdl '/Reference Generator'],'cr_ay_stop','on');
```

- Enable signal logging for the Reference Generator Vis signal outport.
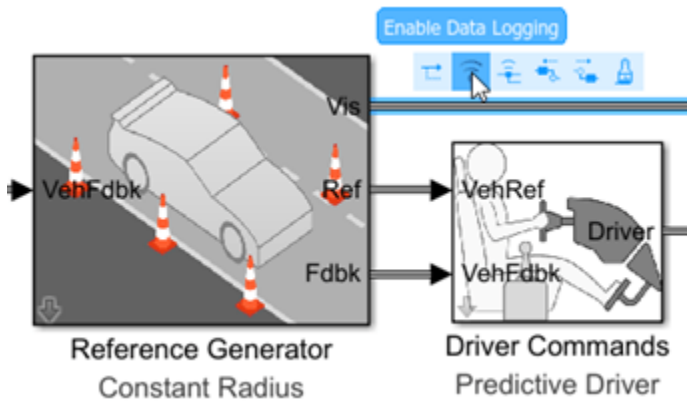

```
ph=get_param('CRReferenceApplication/Reference Generator','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- Enable signal logging for the Passenger Vehicle block outport signal.


```
ph=get_param('CRReferenceApplication/Passenger Vehicle','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- In the Visualization subsystem, enable signal logging for the ISO block.

```
set_param([mdl '/Visualization/ISO 15037-1:2006'],'Measurement','Enable');
```
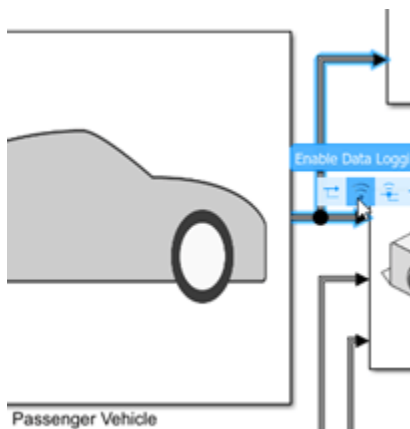
3. Set up a speed set point vector, xdot_r, that you want to investigate. For example, at the command line, type:

```
vmax = [35, 40, 45];
numExperiments = length(vmax);
```

4. Create an array of simulation inputs that set the Reference Generator block parameter **Longitudinal velocity reference, xdot_r** equal to xdot_r.

```
for idx = numExperiments:-1:1
    in(idx) = Simulink.SimulationInput(mdl);
    in(idx) = in(idx).setBlockParameter([mdl '/Reference Generator'], ...
        'xdot_r', num2str(vmax(idx)));
end
```

5. Save the model and run the simulations.

```
save_system(mdl)
tic;
simout = parsim(in,'ShowSimulationManager','on');
toc;

[09-Jan-2023 12:17:49] Checking for availability of parallel pool...
Starting parallel pool (parpool) using the 'Processes' profile ...
Connected to parallel pool with 6 workers.
[09-Jan-2023 12:18:45] Starting Simulink on parallel workers...
[09-Jan-2023 12:19:08] Loading project on parallel workers...
[09-Jan-2023 12:19:08] Configuring simulation cache folder on parallel workers...
[09-Jan-2023 12:19:20] Loading model on parallel workers...
[09-Jan-2023 12:20:08] Running simulations...
[09-Jan-2023 12:22:18] Completed 1 of 3 simulation runs
[09-Jan-2023 12:22:19] Completed 2 of 3 simulation runs
[09-Jan-2023 12:22:19] Completed 3 of 3 simulation runs
[09-Jan-2023 12:22:19] Cleaning up parallel workers...
Elapsed time is 290.844402 seconds.
```

6. Close the Simulation Data Inspector windows.

Use Simulation Data Inspector to Analyze Results

Use the Simulation Data Inspector to examine the results. You can use the UI or, alternatively, command-line functions.

1. Open the Simulation Data Inspector. On the Simulink Toolstrip, on the **Simulation** tab, under **Review Results**, click **Data Inspector**.

- In the Simulation Data Inspector, select **Import**.

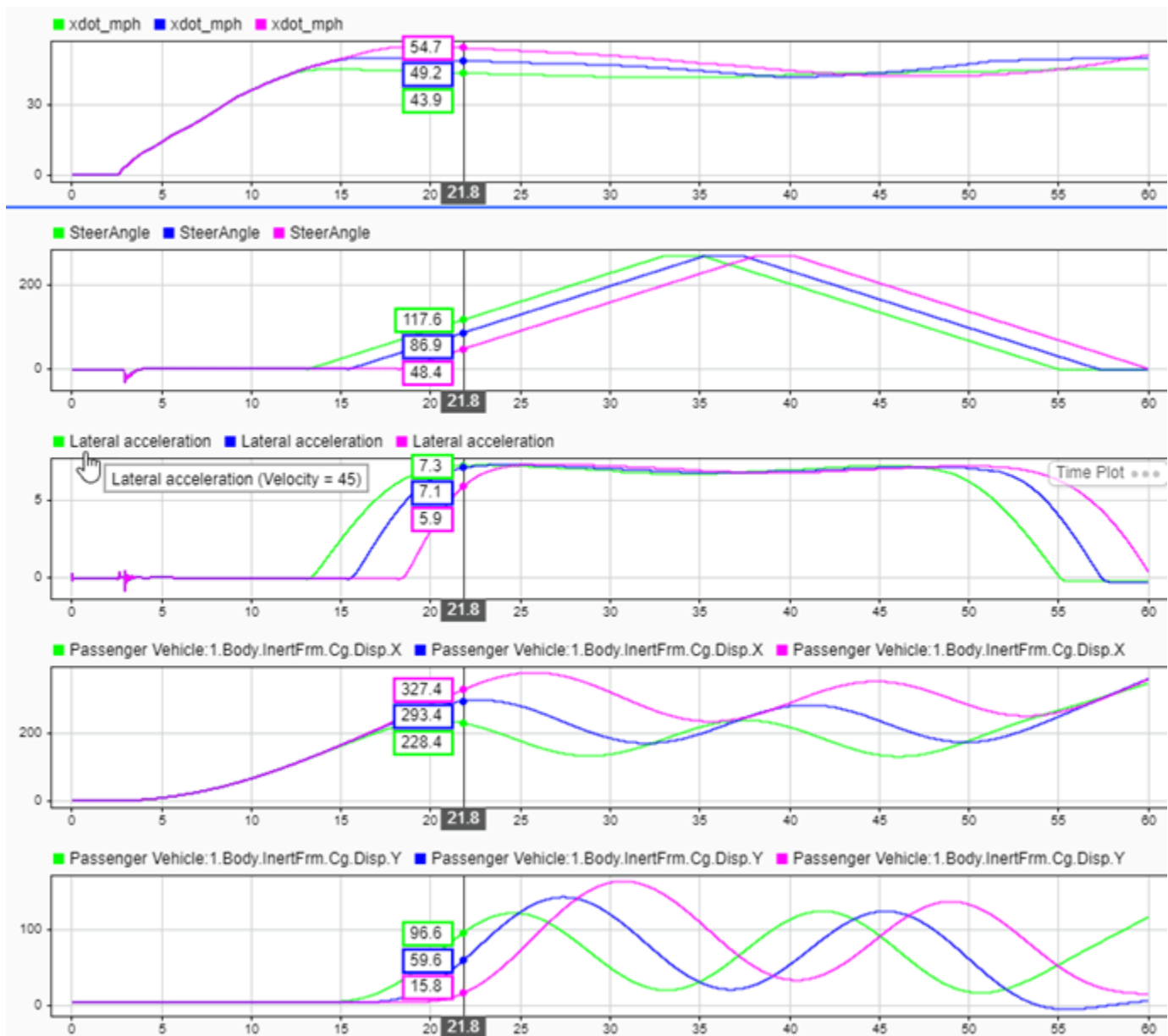- In the **Import** dialog box, clear logsout. Select `simout(1)`, `simout(2)`, and `simout(3)`. Select **Import**.

- Use the Simulation Data Inspector to examine the results.

2. Alternatively, use these MATLAB commands to plot the longitudinal velocity, lateral acceleration, and the steering wheel angle.

```matlab
for idx = 1:numExperiments
    % Create sdi run object
    simoutRun(idx)=Simulink.sdi.Run.create;
    simoutRun(idx).Name=['Velocity = ', num2str(vmax(idx))];
    add(simoutRun(idx),'vars',simout(idx));
end
sigcolor=[0 1 0;0 0 1;1 0 1];
for idx = 1:numExperiments
    % Extract the lateral acceleration, position, and steering
    msignal(idx)=getSignalsByName(simoutRun(idx), 'xdot_mph');
    msignal(idx).LineColor =sigcolor((idx),:);
    ssignal(idx)=getSignalsByName(simoutRun(idx), 'SteerAngle');
    ssignal(idx).LineColor =sigcolor((idx),:);
    asignal(idx)=getSignalsByName(simoutRun(idx), 'ay');
    asignal(idx).LineColor =sigcolor((idx),:);
end
Simulink.sdi.view
Simulink.sdi.setSubPlotLayout(3,1);
for idx = 1:numExperiments
    % Plot the lateral position, steering angle, and lateral acceleration
    plotOnSubPlot(msignal(idx),1,1,true);
    plotOnSubPlot(ssignal(idx),2,1,true);
```

```
        plotOnSubPlot(asignal(idx),3,1,true);
end
```

The results are similar to these plots, which indicate that the greatest lateral acceleration occurs when the vehicle velocity is 45 mph.

Further Analysis

To explore the results further, use these commands to extract the lateral acceleration, steering angle, and vehicle trajectory from the `simout` object.
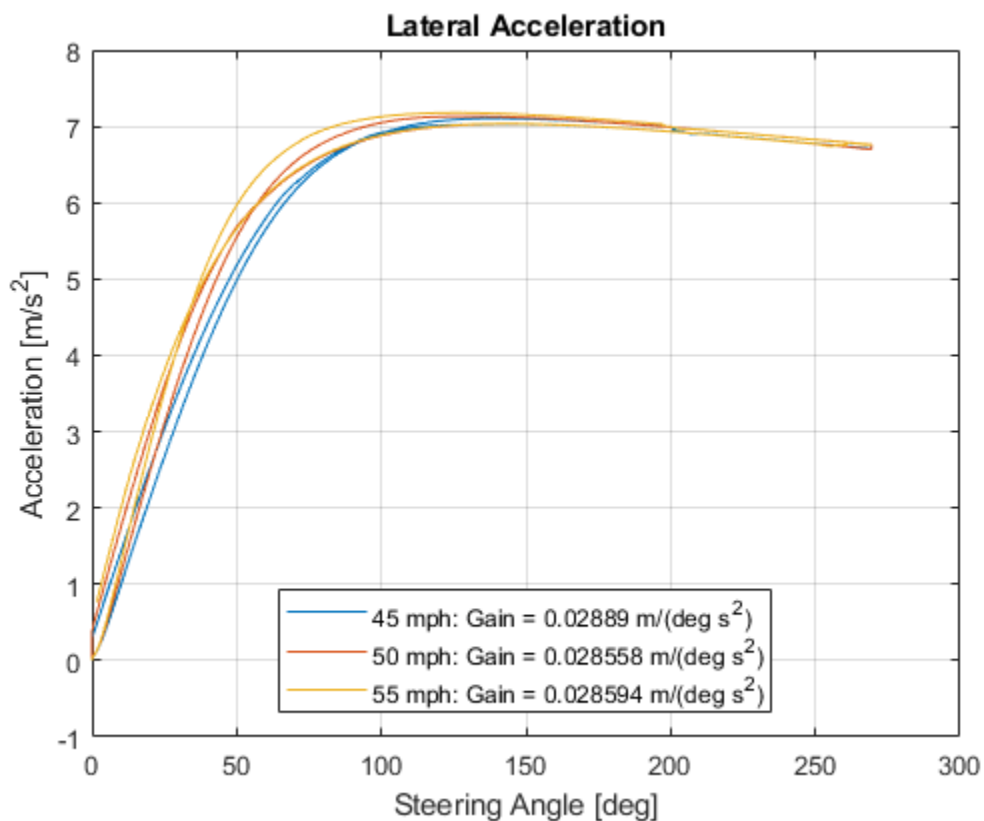
1. Extract the lateral acceleration and steering angle. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    sa=log.get('Steering-wheel angle').Values;
    ay=log.get('Lateral acceleration').Values;
    firstorderfit = polyfit(sa.Data,ay.Data,1);
    gain(idx)=firstorderfit(1);
    legend_labels{idx} = [num2str(vmax(idx)), ' mph: Gain = ', ...
        num2str(gain(idx)), ' m/(deg s^2)'];
    % Plot steering angle vs. lateral acceleration
    plot(sa.Data,ay.Data)
    hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Lateral Acceleration')
xlabel('Steering Angle [deg]')
ylabel('Acceleration [m/s^2]')
grid on
```


Lateral Acceleration

Legend:
- 35 mph: Gain = 0.0017405 m/(deg s^2)
- 40 mph: Gain = 0.0019419 m/(deg s^2)
- 45 mph: Gain = 0.0021781 m/(deg s^2)

2. Extract the vehicle path. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    xValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.X').Va
```

1-43

```
        yValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y').Va
        x = xValues.Data;
        y = yValues.Data;
        legend_labels{idx} = [num2str(vmax(idx)), ' mph'];
        % Plot vehicle location
        axis('equal')
        plot(y,x)
        hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Vehicle Path')
xlabel('Y Position [m]')
ylabel('X Position [m]')
grid on
```


References

[1] J266_199601. *Steady-State Directional Control Test Procedures for Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

[2] ISO 4138:2012. *Passenger cars -- Steady-state circular driving behaviour -- Open-loop test methods*. ISO (International Organization for Standardization), 2012.

See Also

Simulink.SimulationInput | Simulink.SimulationOutput | polyfit

More About

- "Constant Radius Maneuver" on page 3-65
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8
- Simulation Data Inspector

Frequency Response to Steering Angle Input

This example shows how to use the vehicle dynamics swept-sine steering reference application to analyze the dynamic steering response to steering inputs. Specifically, you can examine the vehicle frequency response and lateral acceleration when you run the maneuver with different sinusoidal wave steering amplitudes.

The swept-sine steering maneuver tests the vehicle frequency response to steering inputs. In the test, the driver:

- Accelerates until the vehicle hits a target velocity.
- Commands a sinusoidal steering wheel input.
- Linearly increase the frequency of the sinusoidal wave.

For more information about the reference application, see "Swept-Sine Steering Maneuver" on page 3-41.

```
helpersetupsss;
```
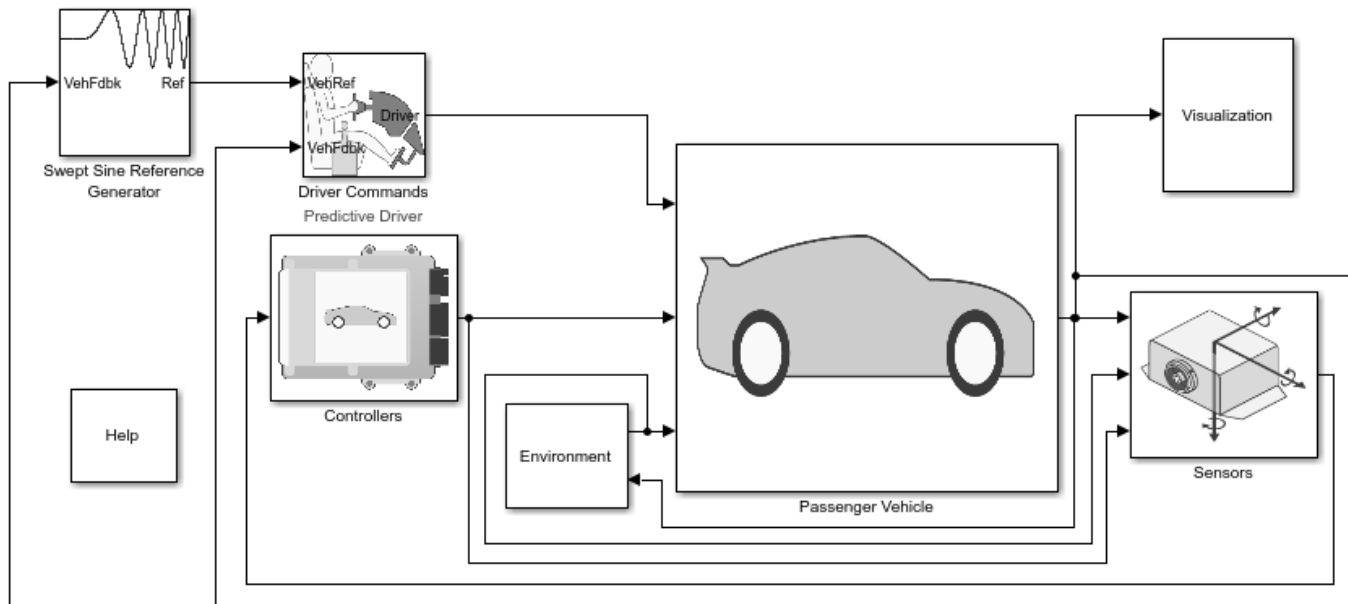

Copyright 2018-2022 The MathWorks, Inc.

Run a Swept-Sine Steering Maneuver

1. Open the Swept Sine Reference Generator block. By default, the maneuver is set with these parameters:

- **Longitudinal velocity setpoint** — 30 mph
- **Steering amplitude** — 90 deg
- **Final frequency** — 0.7 Hz

2. In the Visualization subsystem, open the 3D Engine block. By default, the **3D Engine** parameter is set to **Disabled**. For the 3D visualization engine platform requirements and hardware recommendations, see the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

3. Run the maneuver with the default settings. As the simulation runs, view the vehicle information.

```
mdl = 'SSSReferenceApplication';
sim(mdl);
```

```
### Starting serial model reference simulation build.
### Model reference simulation target for Driveline is up to date.
### Model reference simulation target for PassVeh14DOF is up to date.
### Model reference simulation target for SiMappedEngineV is up to date.

Build Summary

0 of 3 models built (3 models already up to date)
Build duration: 0h 1m 2.9953s
```
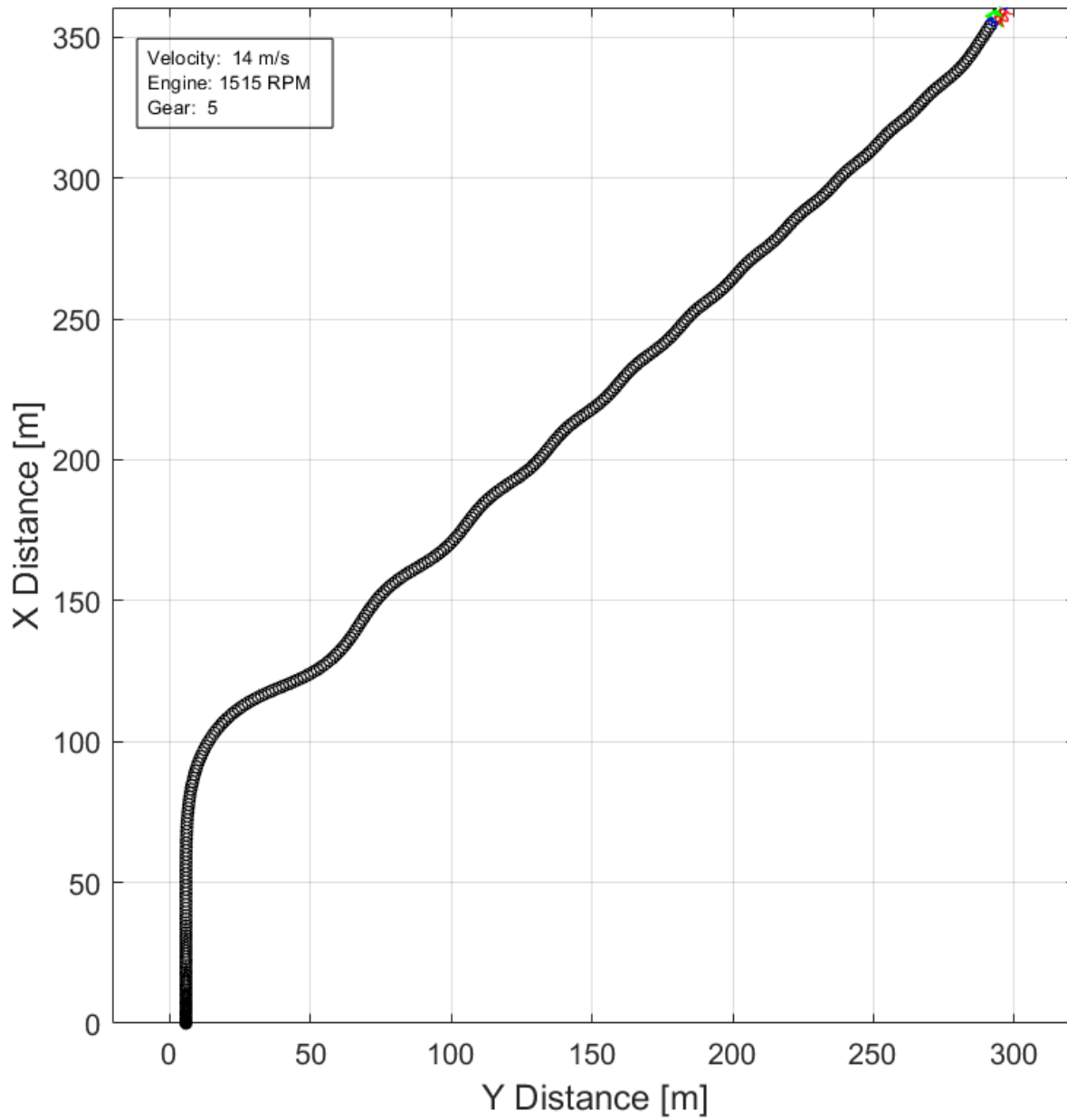
- In the Vehicle Position window, view the vehicle longitudinal distance as a function of the lateral distance. The yellow line is the yaw rate. The blue line is the steering angle.

- In the Visualization subsystem, open the Yaw Rate and Steer Scope block to display the yaw rate and steering angle versus time.
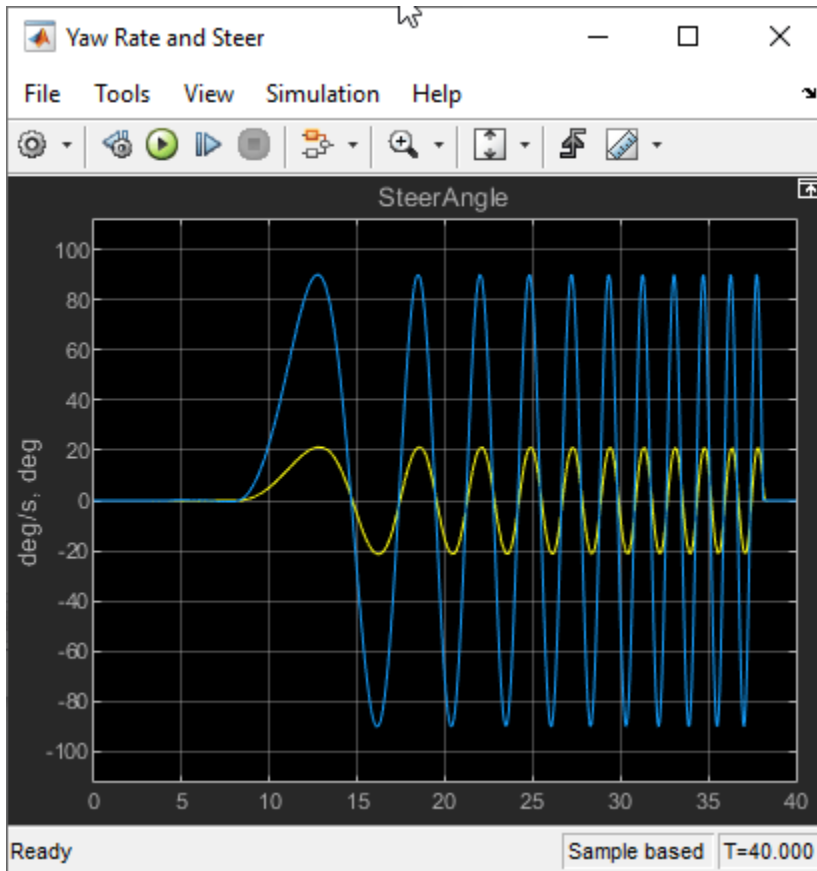
Sweep Steering

Run the reference application with three different sinusoidal wave steering amplitudes.

1. In the swept-sine steering reference application model SSSReferenceApplication, open the Swept Sine Reference Generator block. The **Steering amplitude, theta_hw** block parameter sets the amplitude. By default, the amplitude is 90 deg.

2. Enable signal logging for the velocity, lane, and ISO signals. You can use the Simulink® editor or, alternatively, these MATLAB® commands. Save the model.

• Enable signal logging for the Lane Change Reference Generator outport Lane signal.
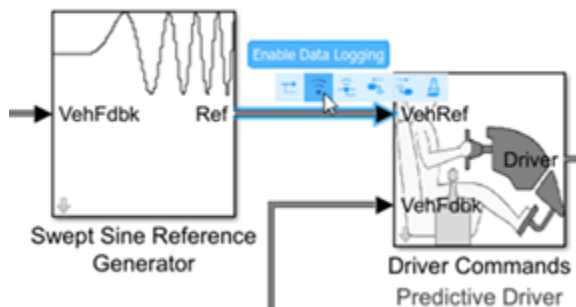
```
mdl = 'SSSReferenceApplication';
open_system(mdl);
ph=get_param('SSSReferenceApplication/Swept Sine Reference Generator','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```

- Enable signal logging for the Passenger Vehicle block outport signal.

Passenger Vehicle

```
ph=get_param('SSSReferenceApplication/Passenger Vehicle','PortHandles');
set_param(ph.Outport(1),'DataLogging','on');
```
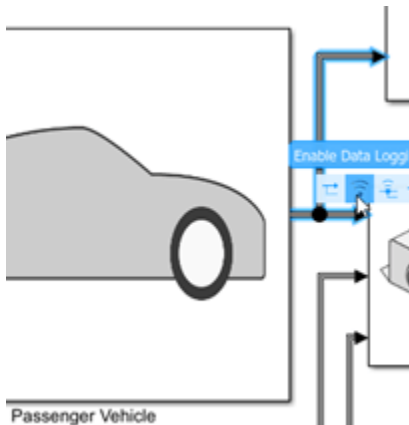
- In the Visualization subsystem, enable signal logging for the ISO block.


```
set_param([mdl '/Visualization/ISO 15037-1:2006'],'Measurement','Enable');
```

3. Set up a steering amplitude vector, `amp`, that you want to investigate. For example, at the command line, type:

```
amp = [60, 90, 120];
numExperiments = length(amp);
```

4. Create an array of simulation inputs that set the Swept Sine Reference Generator block parameter **Steering amplitude, theta_hw** equal to `amp`.

```
for idx = numExperiments:-1:1
    in(idx) = Simulink.SimulationInput(mdl);
    in(idx) = in(idx).setBlockParameter([mdl '/Swept Sine Reference Generator'],...
        'theta_hw',num2str(amp(idx)));
end
```

5. Save the model and run the simulations. If available, use parallel computing.

```
save_system(mdl)
tic;
simout = parsim(in,'ShowSimulationManager','on');
toc;
```

```
[05-Jan-2023 09:51:30] Checking for availability of parallel pool...
[05-Jan-2023 09:51:30] Starting Simulink on parallel workers...
[05-Jan-2023 09:51:32] Loading project on parallel workers...
[05-Jan-2023 09:51:32] Configuring simulation cache folder on parallel workers...
[05-Jan-2023 09:51:32] Loading model on parallel workers...
[05-Jan-2023 09:51:49] Running simulations...
[05-Jan-2023 09:53:10] Completed 1 of 3 simulation runs
[05-Jan-2023 09:53:10] Completed 2 of 3 simulation runs
[05-Jan-2023 09:53:10] Completed 3 of 3 simulation runs
[05-Jan-2023 09:53:10] Cleaning up parallel workers...
Elapsed time is 110.798751 seconds.
```

6. After the simulations complete, close the Simulation Data Inspector windows.

Use Simulation Data Inspector to Analyze Results

Use the Simulation Data Inspector to examine the results. You can use the UI or, alternatively, command-line functions.

1. Open the Simulation Data Inspector. On the Simulink Toolstrip, on the **Simulation** tab, under **Review Results**, click **Data Inspector**.

- In the Simulation Data Inspector, select **Import**.

- In the **Import** dialog box, clear `logsout`. Select `simout(1)`, `simout(2)`, and `simout(3)`. Select **Import**.

- Use the Simulation Data Inspector to examine the results.

2. Alternatively, use these MATLAB commands to plot data for each run. For example, use these commands to plot the lateral position, steering wheel angle, and lateral acceleration. The results are similar to these plots, which show the results for each run.

```matlab
for idx = 1:numExperiments
    % Create sdi run object
    simoutRun(idx)=Simulink.sdi.Run.create;
    simoutRun(idx).Name=['Amplitude = ', num2str(amp(idx))];
    add(simoutRun(idx),'vars',simout(idx));
end
sigcolor=[0 1 0;0 0 1;1 0 1];
for idx = 1:numExperiments
    % Extract the lateral acceleration, position, and steering
    ysignal(idx)=getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y');
    ysignal(idx).LineColor =sigcolor((idx),:);
    ssignal(idx)=getSignalsByName(simoutRun(idx), 'Steering-wheel angle');
    ssignal(idx).LineColor =sigcolor((idx),:);
    asignal(idx)=getSignalsByName(simoutRun(idx), 'Lateral acceleration');
    asignal(idx).LineColor =sigcolor((idx),:);
end
Simulink.sdi.view
Simulink.sdi.setSubPlotLayout(3,1);
for idx = 1:numExperiments
    % Plot the lateral position, steering angle, and lateral acceleration
    plotOnSubPlot(ysignal(idx),1,1,true);
```

```
        plotOnSubPlot(ssignal(idx),2,1,true);
        plotOnSubPlot(asignal(idx),3,1,true);
end
```

The results are similar to these plots, which indicate that the greatest lateral acceleration occurs when the steering amplitude is 120 deg.
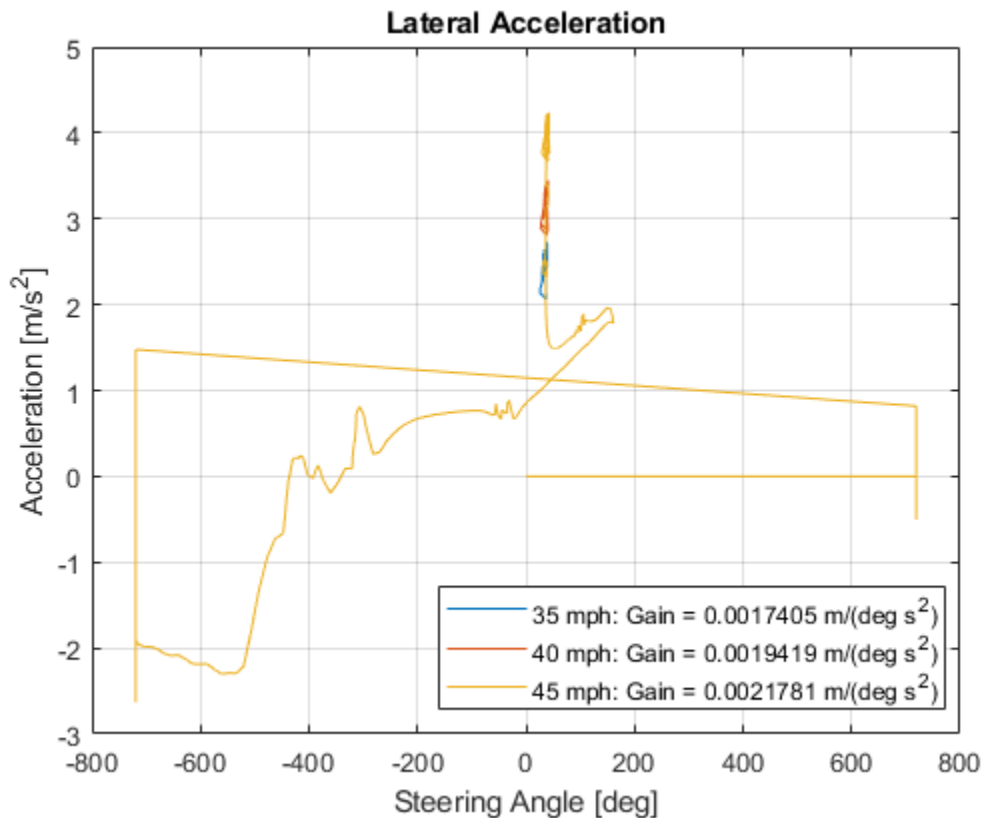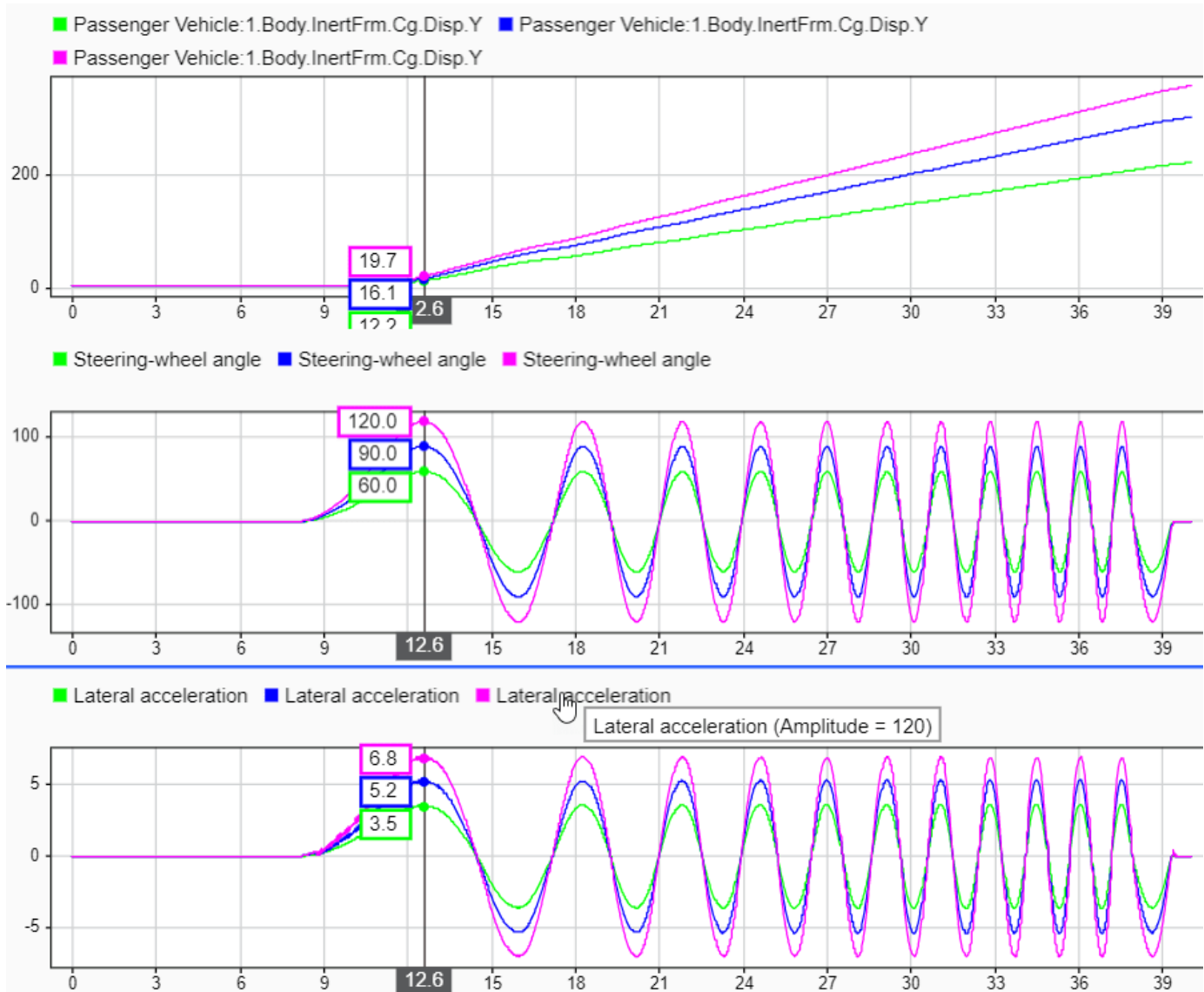

Further Analysis

To explore the results further, use these commands to extract the lateral acceleration, steering angle, and vehicle trajectory from the `simout` object.

1. Extract the lateral acceleration and steering angle. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
```

```
    % Extract Data
    log = get(simout(idx),'logsout');
    sa=log.get('Steering-wheel angle').Values;
    ay=log.get('Lateral acceleration').Values;
    legend_labels{idx} = ['amplitude = ', num2str(amp(idx)), '^{\circ}'];
    % Plot steering angle vs. lateral acceleration
    plot(sa.Data,ay.Data)
    hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Lateral Acceleration')
xlabel('Steering Angle [deg]')
ylabel('Acceleration [m/s^2]')
grid on
```


2. Extract the vehicle path. Plot the data. The results are similar to this plot.

```
figure
for idx = 1:numExperiments
    % Extract Data
    log = get(simout(idx),'logsout');
    xValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.X').Va
    yValues = getSignalsByName(simoutRun(idx), 'Passenger Vehicle:1.Body.InertFrm.Cg.Disp.Y').Va
    x = xValues.Data;
    y = yValues.Data;
    legend_labels{idx} = ['amplitude = ', num2str(amp(idx)), '^{\circ}'];
    % Plot vehicle location
```

```
        axis('equal')
        plot(y,x)
        hold on
end
% Add labels to the plots
legend(legend_labels, 'Location', 'best');
title('Vehicle Path')
xlabel('Y Position [m]')
ylabel('X Position [m]')
grid on
```
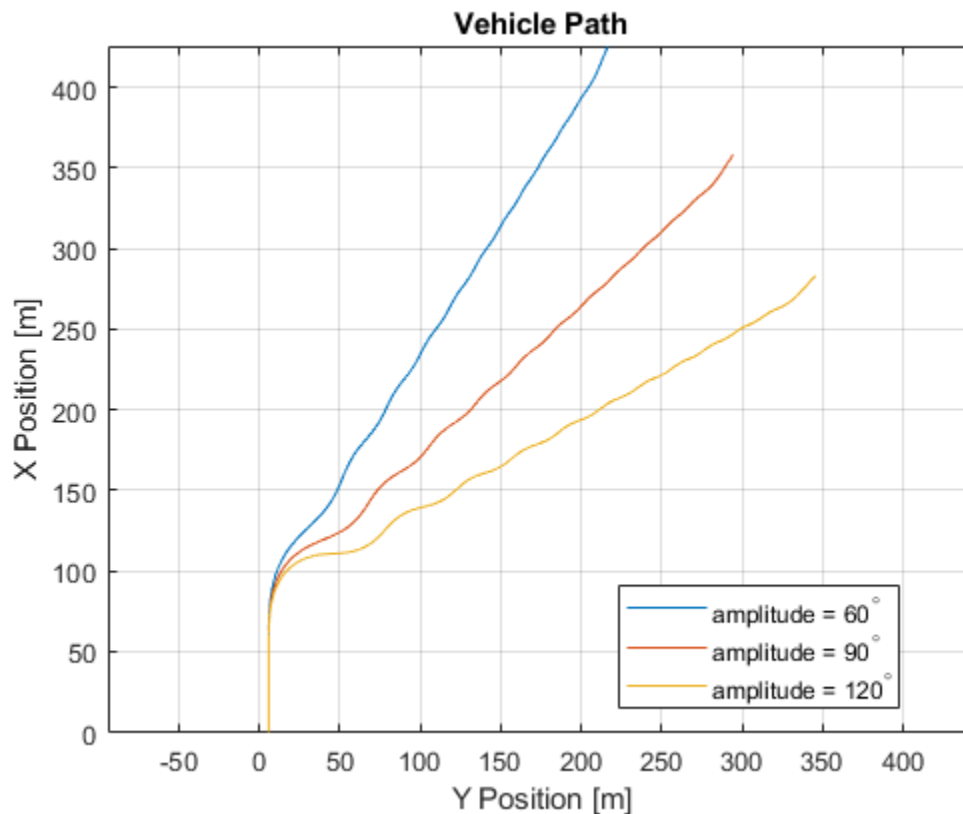

See Also

Simulink.SimulationInput | Simulink.SimulationOutput | fft

More About

- "Fourier Analysis and Filtering"
- Simulation Data Inspector
- "Swept-Sine Steering Maneuver" on page 3-41
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8

Coordinate Systems

Coordinate Systems in Vehicle Dynamics Blockset

Vehicle Dynamics Blockset uses these coordinate systems to calculate the vehicle dynamics and position objects in the 3D visualization environment.

Environment	Description	Coordinate Systems
Vehicle dynamics in Simulink	The *right-hand rule* establishes the *X-Y-Z* sequence and rotation of the coordinate axes used to calculate the vehicle dynamics. The Vehicle Dynamics Blockset 3D simulation environment uses these *right-handed* (RH) *Cartesian* coordinate systems defined in the SAE J670[2] and ISO 8855[3] standards: • Earth-fixed (inertial) • Vehicle • Tire • Wheel The coordinate systems can have either orientation: • Z-down — Defined in SAE J670[2] • Z-up — Defined in SAE J670[2] and ISO 8855[3]	"Earth-Fixed (Inertial) Coordinate System" on page 2-2 "Vehicle Coordinate System" on page 2-3 "Tire and Wheel Coordinate Systems" on page 2-3
3D visualization engine	To position objects and query the 3D visualization environment, the Vehicle Dynamics Blockset uses a world coordinate system.	"World Coordinate System" on page 2-5

Earth-Fixed (Inertial) Coordinate System

The earth-fixed coordinate system (X_E, Y_E, Z_E) axes are fixed in an inertial reference frame. The inertial reference frame has zero linear and angular acceleration and zero angular velocity. In Newtonian physics, the earth is an inertial reference.
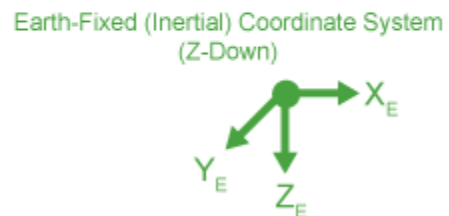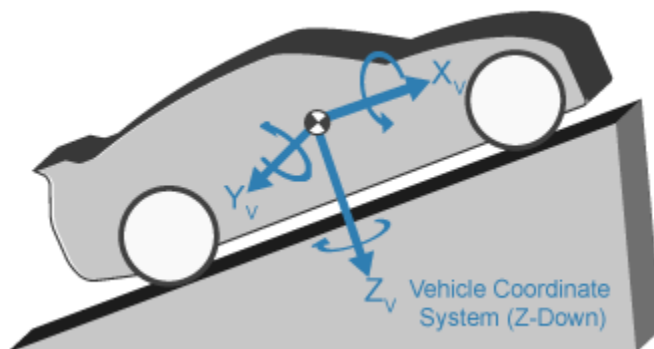
Axis	Description
X_E	The X_E axis is in the forward direction of the vehicle.
Y_E	The X_E and Y_E axes are parallel to the ground plane. The ground plane is a horizontal plane normal to the gravitational vector.
Z_E	In the Z-up orientation, the positive Z_E axis points upward.
	In the Z-down orientation, the positive Z_E axis points downward.

Vehicle Coordinate System

The vehicle coordinate system axes (X_V, Y_V, Z_V) are fixed in a reference frame attached to the vehicle. The origin is at the vehicle sprung mass.

Z-Down Orientation
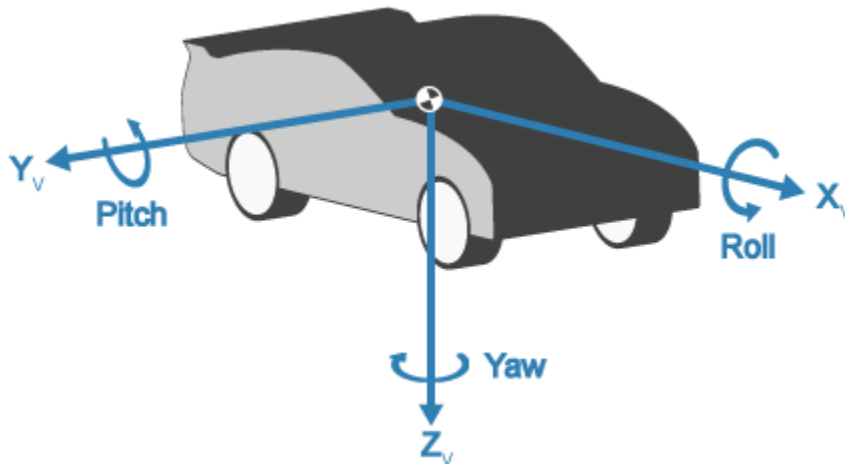
Axis	Description
X_V	The X_V axis points forward and is parallel to the vehicle plane of symmetry.
Y_V	The Y_V axis is perpendicular to the vehicle plane of symmetry.
Z_V	In the Z-down orientation:
	• Y_V axis points to the right
	• Z_V axis points downward

Tire and Wheel Coordinate Systems

The tire coordinate system axes (X_T, Y_T, Z_T) are fixed in a reference frame attached to the tire. The origin is at the tire contact with the ground.

The wheel coordinate system axes (X_W, Y_W, Z_W) are fixed in a reference frame attached to the wheel. The origin is at the wheel center.

Z-Up Orientation[1]
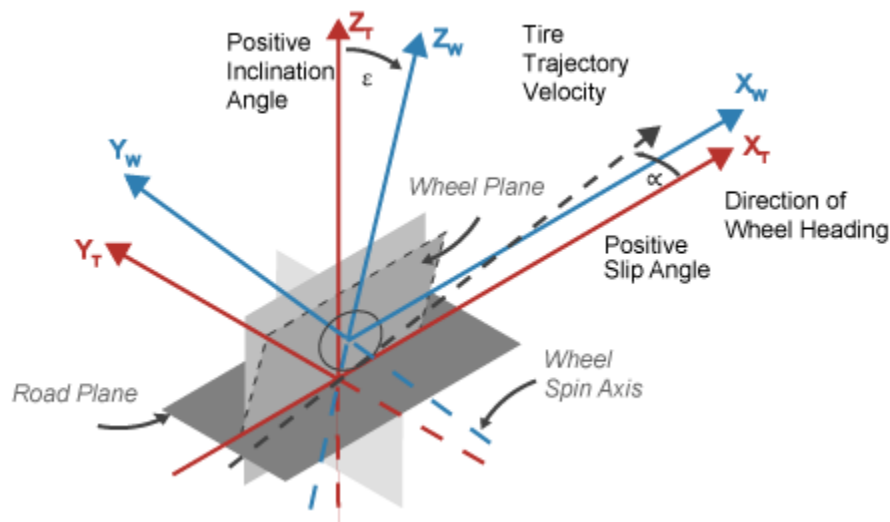
Z-Down Orientation
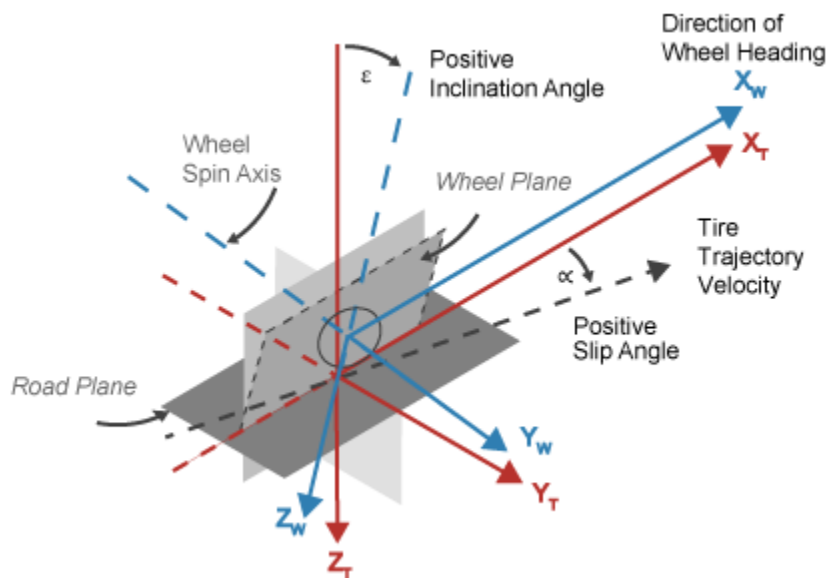
Axis	Description
X_T	X_T and Y_T are parallel to the road plane. The intersection of the wheel plane and the road plane define the orientation of the X_T axis.
Y_T	
Z_T	Z_T points: • Upward in the Z-up orientation • Downward in the Z-down orientation
X_W	X_W and Y_W are parallel to the wheel plane:

---

1   Reprinted with permission Copyright © 2008 SAE International. Further distribution of this material is not permitted without prior permission from SAE.

Axis	Description
Y_W	• X_W is parallel to the local road plane. • Y_W is parallel to the wheel-spin axis.
Z_W	Z_W points: • Upward in the Z-up orientation • Downward in the Z-down orientation

World Coordinate System

The 3D visualization environment uses a world coordinate system with axes that are fixed in the inertial reference frame.
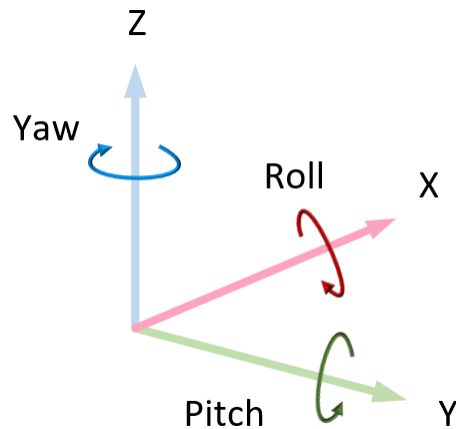
Axis	Description
X	Forward direction of the vehicle Roll — Right-handed rotation about X-axis
Y	Extends to the right of the vehicle, parallel to the ground plane Pitch — Right-handed rotation about Y-axis
Z	Extends upwards Yaw — Left-handed rotation about Z-axis

References

[1] Gillespie, Thomas. *Fundamentals of Vehicle Dynamics*. Warrendale, PA: Society of Automotive Engineers, 1992.

[2] Vehicle Dynamics Standards Committee. *Vehicle Dynamics Terminology*. SAE J670. Warrendale, PA: Society of Automotive Engineers, 2008.

[3] Technical Committee. *Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*. ISO 8855:2011. Geneva, Switzerland: International Organization for Standardization, 2011.

See Also

More About

- "Coordinate Systems in Automated Driving Toolbox" (Automated Driving Toolbox)

External Websites

- SAE International Standards
- ISO Standards

Reference Applications

Passenger Vehicle Dynamics Models

To analyze the dynamic system response in common ride and handling maneuvers, Vehicle Dynamics Blockset provides these pre-assembled vehicle dynamics models.

Vehicle Model	Description	Vehicle Body Degrees-of-Freedom (DOFs)					Wheel DOFs		
Passenger 14DOF Vehicle	• Vehicle with four wheels • Available as model variant in the maneuver reference applications	Six					Two per wheel - eight total		
		Translational		**Rotational**			**Translational**		**Rotational**
		Longitudinal	✓	Pitch	✓		Vertical	✓	Rolling ✓
		Lateral	✓	Yaw	✓				
		Vertical	✓	Roll	✓				
Passenger 7DOF Vehicle	• Vehicle with four wheels • Available as model variant in the maneuver reference applications	Three					One per wheel - four total		
		Translational		**Rotational**			**Rotational**		
		Longitudinal	✓	Pitch			Rolling	✓	
		Lateral	✓	Yaw	✓				
		Vertical		Roll					
Passenger 3DOF Vehicle	• Vehicle with ideal tire	Three					None		
		Translational		**Rotational**					
		Longitudinal	✓	Pitch					
		Lateral	✓	Yaw	✓				
		Vertical		Roll					

From the Simulink start page, you can open project files that contain the vehicle models.
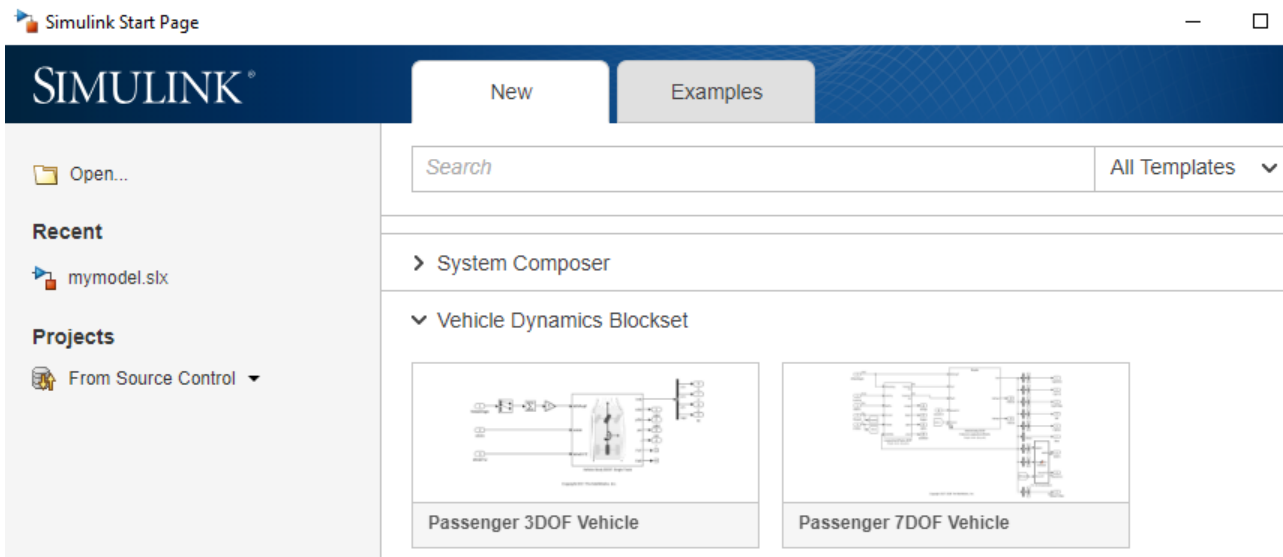
See Also

Vehicle Body 6DOF | Vehicle Body 3DOF

More About

- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "Vehicle Reference Applications"

Longitudinal Motorcycle Braking Test

This reference application represents an in-plane longitudinal motorcycle undergoing a braking test. You can create your own versions, establishing a framework to test that your motorcycle meets the design requirements under normal and extreme driving conditions. Use this reference application in ride and handling studies and chassis controls development to characterize the vehicle dynamics of a motorcycle during a braking test.

To test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software, you can run the maneuver in a 3D environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

To create and open a working copy of the longitudinal motorcycle braking test reference application, enter

```
vdynblksMotoLongBrakingStart
```

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
Straight Maneuver Reference Generator	Generates accelerator and brake commands to conduct a straight line maneuver. The acceleration begins at the specified rate until the motorcycle achieves the longitudinal velocity setpoint. The motorcycle controller maintains the longitudinal velocity setpoint for the specified time or distance. The controller then decelerates the motorcycle. Optionally, specify fault conditions before braking during a test. If the motorcycle speed, steering angle, or yaw rate is not within the allowable range before braking, the block sets a fault condition.	NA
Longitudinal Rider	Implements the rider model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Longitudinal Rider subsystem use is the Longitudinal Driver block with **Control type, cntrlType** set to `Predictive`.	NA
Environment	Implements wind and road forces, including a constant or split friction coefficient scaling factor.	✓
Controllers	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓

Reference Application Element	Description	Variants
Motorcycle Vehicle	Implements the: • Body, suspension, and wheels • Engine • Steering, transmission, driveline, and brakes	✓
Visualization	Provides the motorcycle trajectory, rider response, and 3D visualization. To enable 3D visualization, set the 3D Engine block parameter **3D Engine** parameter to **Enabled**. For the minimum 3D visualization environment hardware requirements, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Set as Label Model Active Choice**.

Straight Maneuver Reference Generator

The Straight Maneuver Reference Generator block generates accelerator and brake commands to conduct a straight line maneuver. The acceleration begins at the specified rate until the motorcycle achieves the longitudinal velocity setpoint. The motorcycle controller maintains the longitudinal velocity setpoint for the specified time or distance. The controller then decelerates the motorcycle.

Use the **Maneuver Parameters** to specify the maneuver start time, velocity setpoint, acceleration, and deceleration.

Optionally, on the **Tracking Parameters** tab, select **Enable fault tracking before braking**. Use the parameters to specify fault conditions before braking. If the motorcycle speed, steering angle, or yaw rate is not within the allowable range before braking, the block sets a fault condition.

For more information, see Straight Maneuver Reference Generator.

Longitudinal Rider

The Longitudinal Rider subsystem implements the rider model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Longitudinal Rider subsystem use is the Longitudinal Driver block with **Control type, cntrlType** set to Predictive.

Environment

The Environment subsystem implements wind and road forces. The reference application has these ground feedback variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Simulation 3D Terrain Sensor block to implement ray tracing in 3D environment.
	`Constant` (default)	Implements a constant friction coefficient scaling factor.

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Transmission Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the motorcycle acceleration, wheel speed, and engine speed.
`Driver - No Clutch` (default)	Open loop transmission control. The controller sets the gear command to the gear request.
`PRNDL Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the motorcycle acceleration, brake command, wheel speed, engine speed, and gear request.
`Paddles`	Implements a paddle controller that uses the motorcycle acceleration and engine speed to generate the gear command.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
`Bang Bang ABS`	Implements an ABS feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.

Variant	Description
`Open Loop`	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.
`Five-State ABS for Motorcycle` (default)	Five-state ABS control when you simulate the brake test. The five-state ABS controller uses logic-switching based on wheel deceleration and motorcycle acceleration to control the braking pressure at each wheel. Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver. The default ABS parameters are set to work on roads that have a constant friction coefficient scaling factor of 1.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
`Rear Diff Controller`	Implements a controller that generates the differential pressure command based on the: • Steer angle • Vehicle pitch • Brake command • Wheel speed • Gear • Vehicle acceleration
`No Control` (default)	Does not implement a controller. Sets the differential pressure command to 0.

Motorcycle Vehicle

The Motorcycle Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the motorcycle contains these subsystems.

Subsystem	Variant	Description
Body, Suspension, Wheels	`Longitudinal` (default)	Motorcycle with two wheels: • Motorcycle — Implemented with Motorcycle Body Longitudinal In-Plane block • Wheels — Implemented with Combined Slip Wheel CPI blocks

Subsystem	Variant	Description
Simscape Multibody	`Simscape Multibody`	Motorcycle with two wheels implemented with Simscape Multibody.

Engine Subsystem	Variant	Description
Engine	`Mapped` (default)	Implemented with Simple Engine block.

Steering, Transmission, Driveline, and Brakes Subsystem		Description
Two Wheels Chain Driven	Transmission	Implements an ideal fixed gear transmission.
	Motorcycle Chain	Implements the dynamic effects of a motorcycle chain on the Motorcycle Body Longitudinal In-Plane block, including dynamic tension and moment drive coupling.

Visualization

When you run the simulation, the Visualization subsystem provides rider and motorcycle response information. The reference application logs motorcycle signals during the maneuver, including steering, motorcycle and engine speed, and lateral acceleration. By default, the Yaw Rate, Brake Pressure, Velocity, Accel Scope block shows the signals as the simulation runs. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Motorcycle response: • Engine speed • Motorcycle speed • Longitudinal acceleration • ABS indicator

Element	Description

Yaw Rate, Brake Pressure, Velocity, Accel Scope block	• `<q>` — Yaw rate versus time
	• `BrkPrs` — Brake pressure versus time
	    • `BrkPrs:1` — Front wheel
	    • `BrkPrs:2` — Rear wheel
	• Signals
	    • `<xdot>` — Longitudinal vehicle velocity versus time
	    • `VehWhlSpd:1` — Front wheel velocity versus time
	    • `VehWhlSpd:2` — Rear wheel velocity versus time
	    • `LngRef` — Longitudinal reference velocity
	• `<ax>` — Longitudinal acceleration versus time

If you enable 3D visualization on the Reference Generator block **3D Engine** tab by selecting **Enabled**, you can view the vehicle response in the Simulation 3D Viewer.

To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene.
Mouse scroll wheel	Control the camera distance from the vehicle.

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation.
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle.

See Also

3D Engine | Straight Maneuver Reference Generator | Motorcycle Body Longitudinal In-Plane | Motorcycle Chain | Simulation 3D Terrain Sensor

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- Simulation Data Inspector

Braking Test

This reference application represents a full vehicle dynamics model undergoing a braking test, including a split-mu test. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use this reference application in ride and handling studies and chassis controls development to characterize the vehicle dynamics during a braking test. For information about this and similar maneuvers, see standards SAE J299_200901[4] and ISO 21994:2007[5].

To test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software, you can run the maneuver in a 3D environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

To create and open a working copy of the braking test reference application, enter

```
vdynblksBrakingStart
```

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
"Straight Maneuver Reference Generator" on page 3-12	Generates accelerator and brake commands to conduct a straight line maneuver. The acceleration begins at the specified rate until the vehicle achieves the longitudinal velocity setpoint. The vehicle controller maintains the longitudinal velocity setpoint for the specified time or distance. The controller then decelerates the vehicle. Optionally, specify fault conditions before braking during a split-mu test. If the vehicle speed, steering angle, or yaw rate is not within the allowable range before braking, the block sets a fault condition. The default values represent compliance with ISO 14512[6].	✓
"Driver Commands" on page 3-12	Implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Driver Commands subsystem variant is the Predictive Driver block.	✓
"Environment" on page 3-13	Implements wind and road forces, including a constant or split friction coefficient scaling factor.	✓
"Controllers" on page 3-13	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓

Reference Application Element	Description	Variants
"Passenger Vehicle" on page 3-15	Implements the: • Body, suspension, and wheels • Engine • Steering, transmission, driveline, and brakes	✓
"Visualization" on page 3-17	Provides the vehicle trajectory, driver response, and 3D visualization. To enable 3D visualization, set the 3D Engine block parameter **3D Engine** parameter to **Enabled**. For the minimum 3D visualization environment hardware requirements, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Override using this Choice**.

Straight Maneuver Reference Generator

The Straight Maneuver Reference Generator block generates accelerator and brake commands to conduct a straight line maneuver. The acceleration begins at the specified rate until the vehicle achieves the longitudinal velocity setpoint. The vehicle controller maintains the longitudinal velocity setpoint for the specified time or distance. The controller then decelerates the vehicle.

Use the **Maneuver Parameters** to specify the maneuver start time, velocity setpoint, acceleration, and deceleration.

Optionally, on the **Tracking Parameters** tab, select **Enable fault tracking before braking**. Use the parameters to specify fault conditions before braking during a split-mu test. If the vehicle speed, steering angle, or yaw rate is not within the allowable range before braking, the block sets a fault condition. The default values represent compliance with ISO 14512[6].

For more information, see Straight Maneuver Reference Generator.

Driver Commands

The Driver Commands block implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, if you select the Reference Generator block parameter **Use maneuver-specific driver, initial position, and scene**, the reference application selects the driver for the maneuver that you specified.

Vehicle Command Mode Setting	Implementation
`Longitudinal Driver`	Longitudinal Driver block — Longitudinal speed-tracking controller. Based on reference and feedback velocities, the block generates normalized acceleration and braking commands that can vary from 0 through 1. Use the block to model the dynamic response of a driver or to generate the commands necessary to track a longitudinal drive cycle.
`Predictive Driver` (default)	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. The controller uses a single-track (bicycle) model for optimal single-point preview control.
`Open Loop`	Implements an open-loop system so that you can configure the reference application for constant or signal-based steering, acceleration, braking, and gear command input.

Environment

The Environment subsystem implements wind and road forces, including a constant or split friction coefficient scaling factor.

Use the Road Track Friction block **Type of surface** parameter to specify the friction coefficient scaling factor:

- `Constant friction coefficient scaling factor` — Constant surface friction during the maneuver
- `Split friction coefficient scaling factor` — Two friction coefficients

  Select this option to specify the friction scaling coefficients for a split-mu braking test. Use the enabled parameters to set the ground friction and rectangular surface friction coefficient scaling factors.

For more information, see Road Track Friction.

The reference application has these ground feedback variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Vehicle Terrain Sensor block to implement ray tracing in 3D environment.
	`Constant` (default)	Implements a constant or split friction coefficient scaling factor based on the Road Track Friction block output.

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Transmission Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, wheel speed, and engine speed.
`Driver - No Clutch`	Open loop transmission control. The controller sets the gear command to the gear request.
`PRNDL Controller` (default)	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, brake command, wheel speed, engine speed, and gear request.
`Paddles`	Implements a paddle controller that uses the vehicle acceleration and engine speed to generate the gear command.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
`Bang Bang ABS`	Implements an ABS feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.
`Open Loop` (default)	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.

Variant	Description
Five-State ABS	Five-state ABS control when you simulate the maneuver.[1,2,3] The five-state ABS controller uses logic-switching based on wheel deceleration and vehicle acceleration to control the braking pressure at each wheel.
	Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver.
	The default ABS parameters are set to work on roads that have either:
	• Constant friction coefficient scaling factor of 0.6.
	• Split friction coefficient scaling factors of 0.6 and 0.8.
	To specify the road surface, see "Environment" on page 3-13.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
Rear Diff Controller	Implements a controller that generates the differential pressure command based on the:
	• Steer angle
	• Vehicle pitch, yaw, and roll
	• Brake command
	• Wheel speed
	• Gear
	• Vehicle acceleration
No Control (default)	Does not implement a controller. Sets the differential pressure command to 0.

Passenger Vehicle

The Passenger Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the vehicle contains these subsystems.

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh7DOF	`PassVeh7DOF`	Vehicle with four wheels: • Vehicle body has three degrees-of-freedom (DOFs) — Longitudinal, lateral, and yaw • Each wheel has one DOF — Rolling Subsystem has variants for the tires, including: • Fiala • Magic Formula
PassVeh14DOF	`PassVeh14DOF` (default)	Vehicle with four wheels. • Vehicle body has six DOFs — Longitudinal, lateral, vertical and pitch, yaw, and roll • Each wheel has two DOFs — Vertical and rolling Subsystem has variants for the suspension, including: • Double Wishbone • Independent Mapped Front • Kinematics and Compliance Independent Suspension Subsystem has variants for the tires, including: • Fiala • Magic Formula • Dugoff

Engine Subsystem	Variant	Description
Mapped Engine	`SiMappedEngine` (default)	Mapped spark-ignition (SI) engine

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
Driveline Ideal Fixed Gear	Driveline model	`All Wheel Drive`	Configure the driveline for all-wheel, front-wheel, rear-wheel, or rear-wheel active differential drive and specify the type of torque coupling.
		`Front Wheel Drive`	
		`Rear Wheel Drive`	
		`Rear Wheel Drive Active Differential` (default)	

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
	Transmission	Ideal (default)	Implements an ideal fixed gear transmission.
	Brake Hydraulics	NA	Implements the heuristic response of a hydraulic system when the controller applies a brake command to a cylinder. Includes front and rear wheel bias coefficients. The subsystem converts the applied pressure to a cylinder spool position. To generate the brake pressure, the spool applies a flow downstream to the cylinders.

Visualization

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Lateral acceleration • ABS indicator • Electronic stability program (ESP) indicator – Indicates when traction control system (TCS) is active
Yaw rate, Brake Pressure, Velocity, Lat Accel Scope block	• Yaw Rate — Steering angle versus time • BrkPrs — Steering angle versus time • VehWhlSpd — Longitudinal wheel speed versus time • <ay> — Lateral acceleration versus time
Vehicle XY Plotter	Vehicle longitudinal versus lateral distance
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

If you enable 3D visualization on the Reference Generator block **3D Engine** tab by selecting **Enabled**, you can view the vehicle response in the Simulation 3D Viewer.

To smoothly change the camera views, use these key commands.

Key	Camera View
1	Back left
2	Back
3	Back right
4	Left
5	Internal
6	Right
7	Front left
8	Front
9	Front right
0	Overhead

View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

References

[1] Pasillas-Lépine, William. "*Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms.*" *Vehicle System Dynamics* 44, no. 2 (2006): 173-188.

[2] Gerard, Mathieu, William Pasillas-Lépine, Edwin De Vries, and Michel Verhaegen. "*Improvements to a five-phase ABS algorithm for experimental validation.*" *Vehicle System Dynamics* 50, no. 10 (2012): 1585-1611.

[3] Bosch, R. "*Bosch Automotive Handbook*." 10th ed. Warrendale, PA: SAE International, 2018.

[4] J299_200901. *Stopping Distance Test Procedure*. Warrendale, PA: SAE International, 2009.

[5] ISO 21994:2007. *Passenger cars — Stopping distance at straight-line braking with ABS — Open-loop test method*. Geneva: ISO, 2007.

[6] ISO 14512:1999. *Passenger cars — Straight-ahead braking on surfaces with split coefficient of friction -- Open-loop test procedure*. Geneva: ISO, 2007.

See Also

3D Engine | Road Track Friction | Straight Maneuver Reference Generator | Vehicle Terrain Sensor

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2
- Simulation Data Inspector

Double-Lane Change Maneuver

This reference application represents a full vehicle dynamics model undergoing a double-lane change maneuver according to standard ISO 3888-2[4]. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application to analyze vehicle ride and handling and develop chassis controls. To perform vehicle studies, including yaw stability and lateral acceleration limits, use this reference application.

ISO 3888-2 defines the double-lane change maneuver to test the obstacle avoidance performance of a vehicle. In the test, the driver:

- Accelerates until vehicle hits a target velocity
- Releases the accelerator pedal
- Turns steering wheel to follow path into the left lane
- Turns steering wheel to follow path back into the right lane

Typically, cones mark the lane boundaries. If the vehicle and driver can negotiate the maneuver without hitting a cone, the vehicle passes the test.

To test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software, you can run the maneuver in a 3D environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

To create and open a working copy of the double-lane change reference application project, enter

`vdynblksDblLaneChangeStart`

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
"Lane Change Reference Generator" on page 3-23	Generates lane signals for the visualization subsystem and trajectory signals	
"Driver Commands" on page 3-23	Implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Driver Commands subsystem variant is the Predictive Driver block.	✓
"Environment" on page 3-67	Implements wind and ground forces	✓
"Controllers" on page 3-24	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓

Reference Application Element	Description	Variants
"Passenger Vehicle" on page 3-26	Implements the: • Body, suspension, and wheels • Engine • Steering, transmission, driveline, and brakes	✓
"Visualization" on page 3-28	Provides the vehicle trajectory, driver response, and 3D visualization	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Override using this Choice**.

Lane Change Reference Generator

Use the Lane Change Reference Generator block to generate:

• Lane signals for the Visualization subsystem — The left and right lane boundaries are a function of the **Vehicle width** parameter.
• Velocity and lateral reference signals for the Predictive Driver block — Use the **Lateral reference position breakpoints** and **Lateral reference data** parameters to specify the lateral reference trajectory as a function of the longitudinal distance.

To start simulations from a non-zero steady-state velocities, use the **Steady-state initial conditions** and **Steady-State Solver** tab parameters. For an example, see "Start Double-Lane Change Maneuver at Target Velocity" on page 3-102.

Driver Commands

The Driver Commands block implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, if you select the Reference Generator block parameter **Use maneuver-specific driver, initial position, and scene**, the reference application selects the driver for the maneuver that you specified.

Vehicle Command Mode Setting	Implementation
Longitudinal Driver	Longitudinal Driver block — Longitudinal speed-tracking controller. Based on reference and feedback velocities, the block generates normalized acceleration and braking commands that can vary from 0 through 1. Use the block to model the dynamic response of a driver or to generate the commands necessary to track a longitudinal drive cycle.
Predictive Driver	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. The controller uses a single-track (bicycle) model for optimal single-point preview control.

Vehicle Command Mode Setting	Implementation
`Predictive Stanley Driver` (default)	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. • For longitudinal control, the block uses a single-track (bicycle) model for optimal single-point preview control. • For lateral control, the block uses a Stanley controller to minimize the position and angle error of the current path with respect to a reference path.
`Open Loop`	Implements an open-loop system so that you can configure the reference application for constant or signal-based steering, acceleration, braking, and gear command input.

Environment

The Environment subsystem generates the wind and ground forces. The reference application has these environment variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Simulation 3D Terrain Sensor block to implement a multipoint terrain sensor in 3D environment
	`Constant` (default)	Implements a constant friction value

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Driver - No Clutch`	Open loop transmission control. The controller sets the gear command to the gear request.

Variant	Description
`PRNDL Controller` (default)	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, brake command, wheel speed, engine speed, and gear request.
`Paddles`	Implements a paddle controller that uses the vehicle acceleration and engine speed to generate the gear command.
`Transmission Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, wheel speed, and engine speed.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
`Bang Bang ABS`	Implements an anti-lock braking system (ABS) feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.
`Open Loop` (default)	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.
`Five-State ABS`	Five-state ABS control when you simulate the maneuver.[1,2,3] The five-state ABS controller uses logic-switching based on wheel deceleration and vehicle acceleration to control the braking pressure at each wheel. Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver. The default ABS parameters are set to work on roads that have a constant friction coefficient scaling factor of 0.6.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
`Rear Diff Controller`	Implements a controller that generates the differential pressure command based on the: • Steer angle • Vehicle pitch, yaw, and roll • Brake command • Wheel speed • Gear • Vehicle acceleration
`No Control` (default)	Does not implement a controller. Sets the differential pressure command to 0.

Passenger Vehicle

The Passenger Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the vehicle contains these subsystems.

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh7DOF	`PassVeh7DOF`	Vehicle with four wheels: • Vehicle body has three degrees-of-freedom (DOFs) — Longitudinal, lateral, and yaw • Each wheel has one DOF — Rolling Subsystem has variants for the tires, including: • Fiala • Magic Formula

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh14DOF	`PassVeh14DOF` (default)	Vehicle with four wheels. • Vehicle body has six DOFs — Longitudinal, lateral, vertical and pitch, yaw, and roll • Each wheel has two DOFs — Vertical and rolling Subsystem has variants for the suspension, including: • Double Wishbone • Independent Mapped Front • Kinematics and Compliance Independent Suspension Subsystem has variants for the tires, including: • Fiala • Magic Formula • Dugoff

Engine Subsystem	Variant	Description
Mapped Engine	`SiMappedEngine` (default)	Mapped spark-ignition (SI) engine

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
Driveline Ideal Fixed Gear	Driveline model	`All Wheel Drive`	Configure the driveline for all-wheel, front-wheel, rear-wheel, or rear-wheel active differential drive and specify the type of torque coupling.
		`Front Wheel Drive`	
		`Rear Wheel Drive`	
		`Rear Wheel Drive Active Differential` (default)	
	Transmission	`Ideal` (default)	Implements an ideal fixed gear transmission.
	Brake Hydraulics	NA	Implements the heuristic response of a hydraulic system when the controller applies a brake command to a cylinder. Includes front and rear wheel bias coefficients. The subsystem converts the applied pressure to a cylinder spool position. To generate the brake pressure, the spool applies a flow downstream to the cylinders.

Visualization

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Acceleration command
Lane Change Scope block	Lateral vehicle displacement versus time: • Red line — Cones marking right lane boundary • Orange line — Cones marking left lane boundary • Blue line — Reference trajectory • Green line — Actual trajectory
Steer, Velocity, Lat Accel Scope block	• `SteerAngle` — Steering angle versus time • `<xdot>` — Longitudinal vehicle velocity versus time • `<ay>` — Lateral acceleration versus time

Element	Description
Vehicle XY Plotter	Vehicle longitudinal versus lateral distance
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

3D Visualization

Optionally, you can enable or disable the 3D visualization environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6. After you open the reference application, in the Visualization subsystem, open the 3D Engine block. Set these parameters.

- **3D Engine** to **Enabled**.
- **Scene** to one of the scenes, for example `Straight road`.

- To position the vehicle in the scene:

  1 Select the position initialization method:

    - **Recommended for scene** — Set the initial vehicle position to values recommended for the scene
    - **User-specified** — Set your own initial vehicle position

  2 Click **Update the model workspaces with the initial values** to overwrite the initial vehicle position in the model workspaces with the applied values.

When you run the simulation, view the vehicle response in the Simulation 3D Viewer.

Note

- To open and close the Simulation 3D Viewer, use the Simulink Run and Stop buttons. If you manually close the Simulation 3D Viewer, Simulink stops the simulation with an error.
- When you enable the 3D visualization environment, you cannot step the simulation back.

To smoothly change the camera views, use these key commands.

Key	Camera View
1	Back left
2	Back
3	Back right
4	Left
5	Internal
6	Right
7	Front left
8	Front
9	Front right
0	Overhead

View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

References

[1] Pasillas-Lépine, William. "*Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms.*" *Vehicle System Dynamics* 44, no. 2 (2006): 173-188.

[2] Gerard, Mathieu, William Pasillas-Lépine, Edwin De Vries, and Michel Verhaegen. "*Improvements to a five-phase ABS algorithm for experimental validation.*" *Vehicle System Dynamics* 50, no. 10 (2012): 1585-1611.

[3] Bosch, R. "*Bosch Automotive Handbook.*" 10th ed. Warrendale, PA: SAE International, 2018.

[4] ISO 3888-2: 2011. *Passenger cars — Test track for a severe lane-change manoeuvre.*

See Also

Predictive Driver | Mapped SI Engine | 3D Engine | Lane Change Reference Generator | Simulation 3D Terrain Sensor

Related Examples

- "Send and Receive Double-Lane Change Scene Data" on page 3-93
- "Start Double-Lane Change Maneuver at Target Velocity" on page 3-102
- "Yaw Stability on Varying Road Surfaces" on page 1-16

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2
- "Passenger Vehicle Dynamics Models" on page 3-2
- "Send and Receive Double-Lane Change Scene Data" on page 3-93
- Simulation Data Inspector

Scene Interrogation in 3D Environment

The scene interrogation with camera and ray tracing reference application provides the Simulink interface with the 3D visualization environment. For the minimum hardware required to run the reference application, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

The scene interrogation with camera and ray tracing reference application contains:

- One passenger vehicle with a simple driveline, combined slip wheel, and 3DOF vehicle dynamics model.
- One camera mounted on the passenger vehicle rear-view mirror.
- Steering, acceleration, gear, and braking controls.
- Vehicle light controls.
- 3D visualization environment configured for the Virtual Mcity scene.
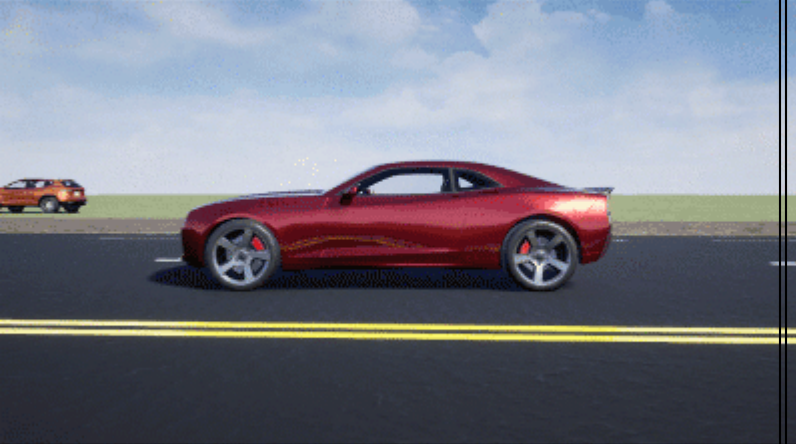
Create and open a working copy of the camera and ray tracing reference application project.
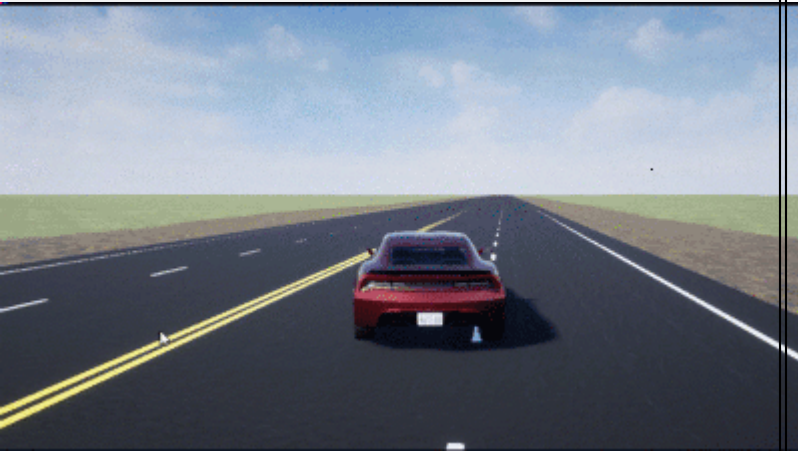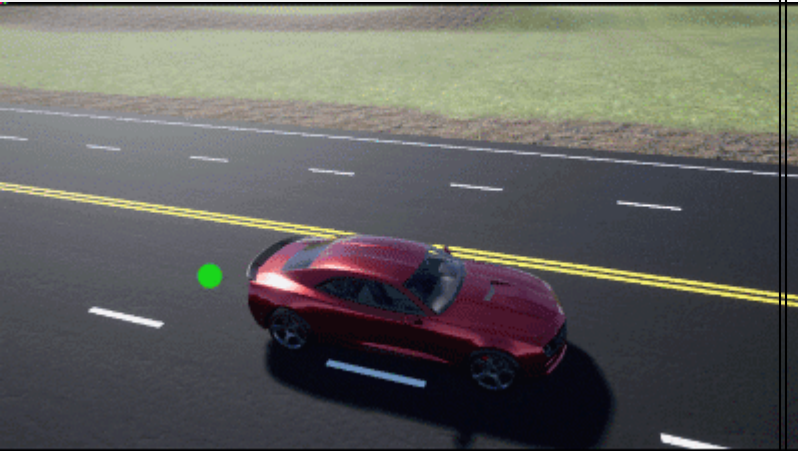
`vdynblksSceneCameraRayStart`
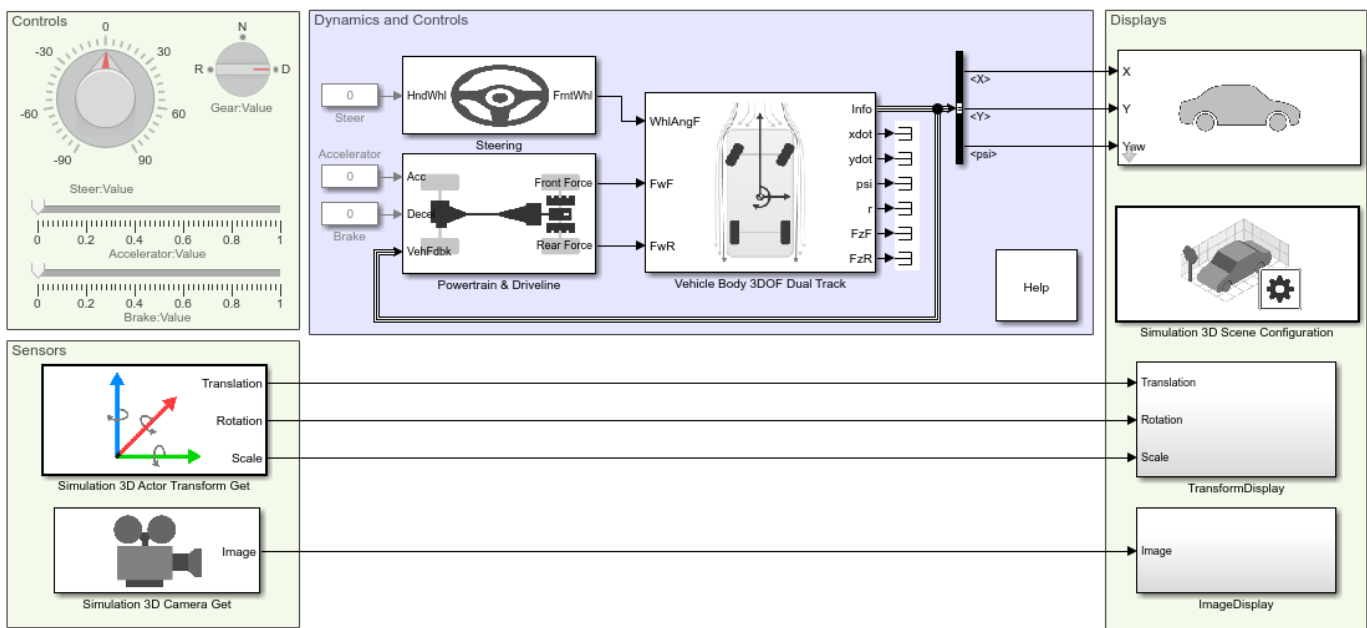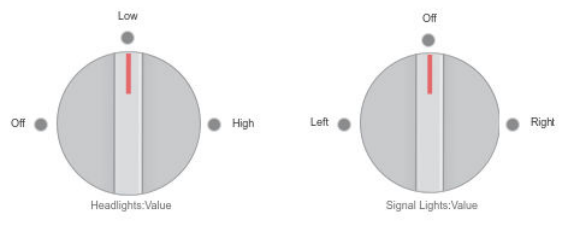
When you run the simulation, the reference application provides this vehicle and scene information.

Window	Description
Simulation 3D Viewer	Video output of the Unreal Engine 3D visualization environment image feedback. By default, the display shows the view from the Simulation 3D Scene Configuration block **Scene view** parameter `SimulinkVehicle1`. To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	

View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene.

Window	Description	
	Key	**Camera Control**
		View Animated GIF
	Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**
	L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation.

Window	Description	
	Key	**Camera Control**
		View Animated GIF
	F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**
SDL Video Display	Video image output of Simulation 3D Camera Get block. By default, the display shows the view specified by these parameter settings: • **Vehicle name** — `SimulinkVehicle1` • **Vehicle mounting location** — `Rearview mirror`	

This table summarizes the parts of the reference application.

Name	Description
Controls	Dials and gauges that control the vehicle steering, gear, acceleration, and braking. The braking control turns on the vehicle brake lights. Setting the gear control to R turns on the vehicle reverse lights.
Sensors	The Simulation 3D Actor Transform Get block returns the translation, rotation, and scale for the vehicle passenger vehicle and four wheels from the 3D visualization environment. The Simulation 3D Camera Get block returns the camera image from the 3D visualization environment. By default, the block returns image data for a camera location specified by these parameter settings: • **Vehicle name** — `SimulinkVehicle1` • **Vehicle mounting location** — `Rearview mirror`
Dynamics and Controls	Interfaces with Simulink to calculate the dynamic response of the vehicle plant and controller. By default, the subsystem contains a simple driveline and the Vehicle 3DOF Dual Track block vehicle dynamics model. Implements a Light Controls subsystem that you can use to control the headlights and signal lights.
Displays	The Simulation 3D Vehicle with Ground Following block implements a passenger vehicle in the 3D visualization environment. The block uses the vehicle position to adjust the vehicle elevation, roll, and pitch so that the vehicle follows the ground terrain. By default, the block has these parameter settings: • **Type** — `Muscle car` • **Color** — `Red` • **Name** — `SimulinkVehicle1` • **Enable light controls** — `On` The Simulation 3D Scene Configuration block configures the Unreal Engine 3D visualization environment. By default, the block has these parameter settings: • **Scene name** — `Virtual Mcity` • **Scene view** — `SimulinkVehicle1` The TransformDisplay subsystem displays the translation, rotation, and scale of the `SimulinkVehicle1` vehicle body and four wheels. The ImageDisplay subsystem displays the video image output of Simulation 3D Camera Get block in the SDL Video Display window.

Displays Subsystems

TransformDisplay Subsystem

In the TransformDisplay subsystem, the Display block provides the translation, rotation, and scale of the vehicle body and four wheels. Use the Constant block value to control the display.

- `1` — Translation
- `2` — Rotation
- `3` — Scale

For example, to display translation information, set the value to `1`.
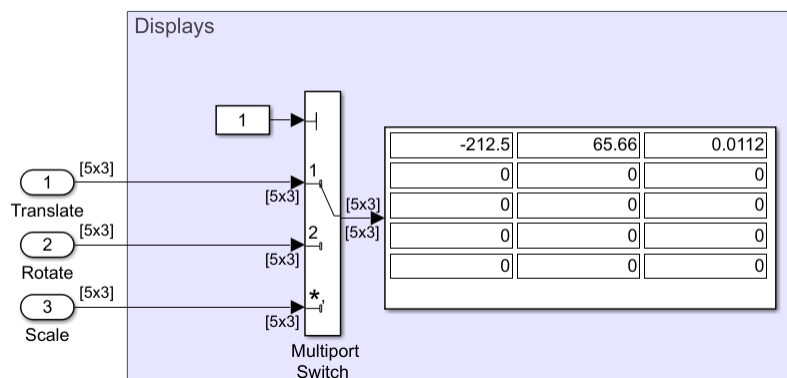
The display indicates that the:

- Vehicle body is at `-212.5` m, `65.66` m, and `0.0112` m along the world *X*-, *Y*-, and *Z*- axes, respectively.
- Wheels are at their initial positions along the world *X*-, *Y*-, and *Z*- axes, respectively.

The Display block provides an array of the vehicle and wheel locations.

$$\begin{bmatrix} Vehicle_X & Vehicle_Y & Vehicle_Z \\ FrontLeft_X & FrontLeft_Y & FrontLeft_Z \\ FrontRight_X & FrontRight_Y & FrontRight_Z \\ RearLeft_X & RearLeft_Y & RearLeft_Z \\ RearRear_X & RearRear_Y & RearRear_Z \end{bmatrix}$$

- Vehicle translation and rotation are along the world coordinate system axes.
- Wheel translations and rotations are with respect to their initial positions, along the world coordinate system axes.

ImageDisplay Subsystem

In the ImageDisplay subsystem, the Level-2 MATLAB S-Function block uses the `VideoDisplayMSfcnWin` function to display the video image output of Simulation 3D Camera Get block.

See Also

Simulation 3D Actor Transform Get | Simulation 3D Camera Get | Simulation 3D Scene Configuration | Virtual Mcity | Simulation 3D Vehicle with Ground Following

Related Examples

- "Send and Receive Double-Lane Change Scene Data" on page 3-93

More About

- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8

External Websites

- Unreal Engine

Swept-Sine Steering Maneuver

This reference application represents a full vehicle dynamics model undergoing a swept-sine steering maneuver. You can create your own versions, providing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application to analyze vehicle ride and handling and develop chassis controls. To analyze the dynamic steering response, use this reference application.

The swept-sine steering maneuver tests the vehicle frequency response to steering inputs. In the test, the driver:

- Accelerates until the vehicle hits a target velocity.
- Commands a sinusoidal steering wheel input.
- Linearly increase the frequency of the sinusoidal wave.

To test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software, you can run the maneuver in a 3D environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

To create and open a working copy of the swept-sine steering reference application project, enter

```
vdynblksSweptSineSteeringStart
```

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
"Swept Sine Reference Generator" on page 3-42	Generate the sinusoidal steering commands for a swept-sine steering maneuver.	
"Driver Commands" on page 3-42	Implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Driver Commands subsystem variant is the Predictive Driver block.	✓
"Environment" on page 3-43	Implements wind and road forces.	✓
"Controllers" on page 3-43	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓
"Passenger Vehicle" on page 3-44	Implements the: - Body, suspension, and wheels - Engine - Steering, transmission, driveline, and brakes	✓

Reference Application Element	Description	Variants
"Visualization" on page 3-46	Provides the vehicle trajectory, driver response, and 3D visualization.	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Override using this Choice**.

Swept Sine Reference Generator

Use the Swept Sine Reference Generator block to generate the sinusoidal steering commands for a swept-sine steering maneuver.

- **Longitudinal velocity setpoint** — Target velocity
- **Steering amplitude** — Sinusoidal wave amplitude
- **Final frequency** — Cut off frequency to stop the maneuver

To start simulations from a steady-state condition, use the **Steady-state initial conditions** and **Steady-State Solver** tab parameters.

For more information, see Swept Sine Reference Generator.

Driver Commands

The Driver Commands block implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, if you select the Reference Generator block parameter **Use maneuver-specific driver, initial position, and scene**, the reference application selects the driver for the maneuver that you specified.

Vehicle Command Mode Setting	Implementation
Longitudinal Driver	Longitudinal Driver block — Longitudinal speed-tracking controller. Based on reference and feedback velocities, the block generates normalized acceleration and braking commands that can vary from 0 through 1. Use the block to model the dynamic response of a driver or to generate the commands necessary to track a longitudinal drive cycle.
Predictive Driver (default)	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. The controller uses a single-track (bicycle) model for optimal single-point preview control.
Open Loop	Implements an open-loop system so that you can configure the reference application for constant or signal-based steering, acceleration, braking, and gear command input.

Environment

The Environment subsystem generates the wind and ground forces. The reference application has these environment variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Simulation 3D Terrain Sensor block to implement a multipoint terrain sensor in 3D environment
	`Constant` (default)	Implements a constant friction value

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Driver - No Clutch`	Open loop transmission control. The controller sets the gear command to the gear request.
`PRNDL Controller` (default)	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, brake command, wheel speed, engine speed, and gear request.
`Paddles`	Implements a paddle controller that uses the vehicle acceleration and engine speed to generate the gear command.
`Transmission Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, wheel speed, and engine speed.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
`Bang Bang ABS`	Implements an anti-lock braking system (ABS) feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.
`Open Loop` (default)	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.
`Five-State ABS`	Five-state ABS control when you simulate the maneuver.[1,2,3] The five-state ABS controller uses logic-switching based on wheel deceleration and vehicle acceleration to control the braking pressure at each wheel. Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver. The default ABS parameters are set to work on roads that have a constant friction coefficient scaling factor of 0.6.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
`Rear Diff Controller`	Implements a controller that generates the differential pressure command based on the: • Steer angle • Vehicle pitch, yaw, and roll • Brake command • Wheel speed • Gear • Vehicle acceleration
`No Control` (default)	Does not implement a controller. Sets the differential pressure command to 0.

Passenger Vehicle

The Passenger Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the vehicle contains these subsystems.

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh7DOF	`PassVeh7DOF`	Vehicle with four wheels: • Vehicle body has three degrees-of-freedom (DOFs) — Longitudinal, lateral, and yaw • Each wheel has one DOF — Rolling Subsystem has variants for the tires, including: • Fiala • Magic Formula
PassVeh14DOF	`PassVeh14DOF` (default)	Vehicle with four wheels. • Vehicle body has six DOFs — Longitudinal, lateral, vertical and pitch, yaw, and roll • Each wheel has two DOFs — Vertical and rolling Subsystem has variants for the suspension, including: • Double Wishbone • Independent Mapped Front • Kinematics and Compliance Independent Suspension Subsystem has variants for the tires, including: • Fiala • Magic Formula • Dugoff

Engine Subsystem	Variant	Description
Mapped Engine	`SiMappedEngine` (default)	Mapped spark-ignition (SI) engine

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
Driveline Ideal Fixed Gear	Driveline model	`All Wheel Drive`	Configure the driveline for all-wheel, front-wheel, rear-wheel, or rear-wheel active differential drive and specify the type of torque coupling.
		`Front Wheel Drive`	
		`Rear Wheel Drive`	
		`Rear Wheel Drive Active Differential` (default)	

Steering, Transmission, Driveline, and Brakes Subsystem	Variant	Description
Transmission	`Ideal` (default)	Implements an ideal fixed gear transmission.
Brake Hydraulics	NA	Implements the heuristic response of a hydraulic system when the controller applies a brake command to a cylinder. Includes front and rear wheel bias coefficients. The subsystem converts the applied pressure to a cylinder spool position. To generate the brake pressure, the spool applies a flow downstream to the cylinders.

Visualization

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Lateral acceleration

Element	Description
Yaw Rate and Steer Scope block	Yaw rate and steering angle versus time: • Yellow line — Yaw rate • Blue lines — Steering angle
Steer vs Ay Scope block	Steering angle versus lateral acceleration
Steer, Velocity, Lat Accel Scope block	• `SteerAngle` — Steering angle versus time • `<xdot>` — Longitudinal vehicle velocity versus time • `<ay>` — Lateral acceleration versus time
Vehicle XY Plotter	Plot of vehicle longitudinal versus lateral distance
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

3D Visualization

Optionally, you can enable or disable the 3D visualization environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6. After you open the reference application, in the Visualization subsystem, open the 3D Engine block. Set these parameters.

• **3D Engine** to **Enabled**.

• **Scene** to one of the scenes, for example `Straight road`.

• To position the vehicle in the scene:

   1 Select the position initialization method:

   • **Recommended for scene** — Set the initial vehicle position to values recommended for the scene

   • **User-specified** — Set your own initial vehicle position

   2 Click **Update the model workspaces with the initial values** to overwrite the initial vehicle position in the model workspaces with the applied values.

When you run the simulation, view the vehicle response in the Simulation 3D Viewer.

___
Note

- To open and close the Simulation 3D Viewer, use the Simulink Run and Stop buttons. If you manually close the Simulation 3D Viewer, Simulink stops the simulation with an error.
- When you enable the 3D visualization environment, you cannot step the simulation back.

To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	**View Animated GIF**

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

References

[1] Pasillas-Lépine, William. "*Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms.*" *Vehicle System Dynamics* 44, no. 2 (2006): 173-188.

[2] Gerard, Mathieu, William Pasillas-Lépine, Edwin De Vries, and Michel Verhaegen. "*Improvements to a five-phase ABS algorithm for experimental validation*." *Vehicle System Dynamics* 50, no. 10 (2012): 1585-1611.

[3] Bosch, R. "*Bosch Automotive Handbook*." 10th ed. Warrendale, PA: SAE International, 2018.

See Also

Longitudinal Driver | Mapped SI Engine | 3D Engine | Swept Sine Reference Generator | Simulation 3D Terrain Sensor

Related Examples

- "Frequency Response to Steering Angle Input" on page 1-46

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2
- "Passenger Vehicle Dynamics Models" on page 3-2
- Simulation Data Inspector

Slowly Increasing Steering Maneuver

This reference application represents a full vehicle dynamics model undergoing a slowly increasing steering maneuver according to standard SAE J266[4]. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application to analyze vehicle ride and handling and develop chassis controls. To characterize the steering and lateral vehicle dynamics, use this reference application.

Based on the constant speed, variable steer test defined in SAE J266[4], the slowly increasing steering maneuver helps characterize the lateral dynamics of the vehicle. In the test, the driver:

- Accelerates until vehicle hits a target velocity.
- Maintains a target velocity.
- Linearly increases the steering wheel angle from 0 degrees to a maximum angle.
- Maintains the steering wheel angle for a specified time.
- Linearly decreases the steering wheel angle from maximum angle to 0 degrees.

To test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software, you can run the maneuver in a 3D environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

To create and open a working copy of the increasing steering reference application project, enter

```
vdynblksIncreasingSteeringStart
```

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
"Slowly Increasing Steer Block" on page 3-54	Generates steering, accelerator, and brake commands.	
"Driver Commands" on page 3-54	Implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Driver Commands subsystem variant is the Predictive Driver block.	✓
"Environment" on page 3-55	Implements wind and road forces.	✓
"Controllers" on page 3-55	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓

Reference Application Element	Description	Variants
"Passenger Vehicle" on page 3-56	Implements the: • Body, suspension, and wheels • Engine • Steering, transmission, driveline, and brakes	✓
"Visualization" on page 3-58	Provides the vehicle trajectory, driver response, and 3D visualization	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Override using this Choice**.

Slowly Increasing Steer Block

Use the Slowly Increasing Steering block to generate steering, accelerator, and brake commands for a slowly increasing steering maneuver.

• **Longitudinal speed setpoint** — Target velocity setpoint
• **Handwheel rate** — Linear rate to increase steering wheel angle
• **Maximum handwheel angle** — Maximum steering wheel angle

To start simulations from a steady-state condition, use the **Steady-state initial conditions** and **Steady-State Solver** tab parameters.

For more information, see Slowly Increasing Steer Reference Generator.

Driver Commands

The Driver Commands block implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, if you select the Reference Generator block parameter **Use maneuver-specific driver, initial position, and scene**, the reference application selects the driver for the maneuver that you specified.

Vehicle Command Mode Setting	Implementation
`Longitudinal Driver`	Longitudinal Driver block — Longitudinal speed-tracking controller. Based on reference and feedback velocities, the block generates normalized acceleration and braking commands that can vary from 0 through 1. Use the block to model the dynamic response of a driver or to generate the commands necessary to track a longitudinal drive cycle.
`Predictive Driver` (default)	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. The controller uses a single-track (bicycle) model for optimal single-point preview control.

Vehicle Command Mode Setting	Implementation
`Open Loop`	Implements an open-loop system so that you can configure the reference application for constant or signal-based steering, acceleration, braking, and gear command input.

Environment

The Environment subsystem generates the wind and ground forces. The reference application has these environment variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Simulation 3D Terrain Sensor block to implement a multipoint terrain sensor in 3D environment
	`Constant` (default)	Implements a constant friction value

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Driver - No Clutch`	Open loop transmission control. The controller sets the gear command to the gear request.
`PRNDL Controller` (default)	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, brake command, wheel speed, engine speed, and gear request.
`Paddles`	Implements a paddle controller that uses the vehicle acceleration and engine speed to generate the gear command.
`Transmission Controller`	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, wheel speed, and engine speed.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
Bang Bang ABS	Implements an anti-lock braking system (ABS) feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.
Open Loop (default)	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.
Five-State ABS	Five-state ABS control when you simulate the maneuver.[1,2,3] The five-state ABS controller uses logic-switching based on wheel deceleration and vehicle acceleration to control the braking pressure at each wheel. Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver. The default ABS parameters are set to work on roads that have a constant friction coefficient scaling factor of 0.6.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
Rear Diff Controller	Implements a controller that generates the differential pressure command based on the: • Steer angle • Vehicle pitch, yaw, and roll • Brake command • Wheel speed • Gear • Vehicle acceleration
No Control (default)	Does not implement a controller. Sets the differential pressure command to 0.

Passenger Vehicle

The Passenger Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the vehicle contains these subsystems.

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh7DOF	PassVeh7DOF	Vehicle with four wheels: • Vehicle body has three degrees-of-freedom (DOFs) — Longitudinal, lateral, and yaw • Each wheel has one DOF — Rolling Subsystem has variants for the tires, including: • Fiala • Magic Formula
PassVeh14DOF	PassVeh14DOF (default)	Vehicle with four wheels. • Vehicle body has six DOFs — Longitudinal, lateral, vertical and pitch, yaw, and roll • Each wheel has two DOFs — Vertical and rolling Subsystem has variants for the suspension, including: • Double Wishbone • Independent Mapped Front • Kinematics and Compliance Independent Suspension Subsystem has variants for the tires, including: • Fiala • Magic Formula • Dugoff

Engine Subsystem	Variant	Description
Mapped Engine	SiMappedEngine (default)	Mapped spark-ignition (SI) engine

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
Driveline Ideal Fixed Gear	Driveline model	All Wheel Drive	Configure the driveline for all-wheel, front-wheel, rear-wheel, or rear-wheel active differential drive and specify the type of torque coupling.
		Front Wheel Drive	
		Rear Wheel Drive	
		Rear Wheel Drive Active Differential (default)	

Steering, Transmission, Driveline, and Brakes Subsystem	Variant	Description
Transmission	`Ideal` (default)	Implements an ideal fixed gear transmission.
Brake Hydraulics	NA	Implements the heuristic response of a hydraulic system when the controller applies a brake command to a cylinder. Includes front and rear wheel bias coefficients. The subsystem converts the applied pressure to a cylinder spool position. To generate the brake pressure, the spool applies a flow downstream to the cylinders.

Visualization

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Lateral acceleration

Element	Description
Yaw Rate and Steer Scope block	Yaw rate and steering angle versus time: • Yellow line — Yaw rate • Blue lines — Steering angle
Steer, Velocity, Lat Accel Scope block	• `SteerAngle` — Steering angle versus time • `<xdot>` — Longitudinal vehicle velocity versus time • `<ay>` — Lateral acceleration versus time
Vehicle XY Plotter	Plot of vehicle longitudinal versus lateral distance
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

3D Visualization

Optionally, you can enable or disable the 3D visualization environment. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6. After you open the reference application, in the Visualization subsystem, open the 3D Engine block. Set these parameters.

• **3D Engine** to **Enabled**.

• **Scene** to one of the scenes, for example `Straight road`.

• To position the vehicle in the scene:

   1 Select the position initialization method:

      • **Recommended for scene** — Set the initial vehicle position to values recommended for the scene

      • **User-specified** — Set your own initial vehicle position

   2 Click **Update the model workspaces with the initial values** to overwrite the initial vehicle position in the model workspaces with the applied values.

When you run the simulation, view the vehicle response in the Simulation 3D Viewer.

Note

- To open and close the Simulation 3D Viewer, use the Simulink Run and Stop buttons. If you manually close the Simulation 3D Viewer, Simulink stops the simulation with an error.
- When you enable the 3D visualization environment, you cannot step the simulation back.

To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	
		View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

References

[1] Pasillas-Lépine, William. "*Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms.*" *Vehicle System Dynamics* 44, no. 2 (2006): 173-188.

[2] Gerard, Mathieu, William Pasillas-Lépine, Edwin De Vries, and Michel Verhaegen. "*Improvements to a five-phase ABS algorithm for experimental validation*." *Vehicle System Dynamics* 50, no. 10 (2012): 1585-1611.

[3] Bosch, R. "*Bosch Automotive Handbook*." 10th ed. Warrendale, PA: SAE International, 2018.

[4] SAE J266. *Steady-State Directional Control Test Procedures For Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

See Also

Longitudinal Driver | Mapped SI Engine | 3D Engine | Slowly Increasing Steer Reference Generator | Simulation 3D Terrain Sensor

Related Examples

- "Vehicle Steering Gain at Different Speeds" on page 1-26

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2
- "Passenger Vehicle Dynamics Models" on page 3-2
- Simulation Data Inspector

Constant Radius Maneuver

This reference application represents a full vehicle dynamics model undergoing a constant radius test maneuver. For information about similar maneuvers, see standards SAE J266_199601[4] and ISO 4138:2012[5]. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use this reference application in ride and handling studies and chassis controls development to characterize the steering and lateral vehicle dynamics.

You can configure the reference application for open-loop and closed-loop tests:

- Open-loop — Maintain the target velocity and steering wheel angle to determine the lateral acceleration, side-slip characteristics, and steering angles for specific accelerations and subsequent test maneuvers. For the open-loop test, set the Reference Generator block **Maneuver** parameter to `Increasing Steer`.
- Closed-loop — Use the predictive driver to maintain a prespecified turn radius at different velocities for drivability and handling performance studies. For the closed-loop test, set the Reference Generator block **Maneuver** parameter to `Constant radius`.

To create and open a working copy of the constant radius reference application, enter

`vdynblksConstRadiusStart`

This table summarizes the blocks and subsystems in the reference application. Some subsystems contain variants.

Reference Application Element	Description	Variants
"Reference Generator" on page 3-66	Sets the parameters that configure the maneuver and 3D visualization environment. By default, the block is set for the constant radius maneuver with the 3D simulation engine environment disabled. For the minimum 3D visualization environment hardware requirements, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6. To enable 3D visualization, on the **3D Engine** tab, select **Enabled**.	✓
"Driver Commands" on page 3-66	Implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, Driver Commands subsystem variant is the Predictive Driver block.	✓
"Environment" on page 3-67	Implements wind and road forces.	✓
"Controllers" on page 3-67	Implements controllers for engine control units (ECUs), transmissions, anti-lock braking systems (ABS), and active differentials.	✓

Reference Application Element	Description	Variants
"Passenger Vehicle" on page 3-69	Implements the: • Body, suspension, and wheels • Engine • Steering, transmission, driveline, and brakes	✓
"Visualization" on page 3-71	Provides the vehicle trajectory and driver response	✓

To override the default variant, on the **Modeling** tab, in the **Design** section, click the drop-down. In the **General** section, select **Variant Manager**. In the Variant Manager, navigate to the variant that you want to use. Right-click and select **Override using this Choice**.

Reference Generator

The Reference Generator block sets the parameters that configure the maneuver and 3D simulation environment. By default, the block is set for the constant radius maneuver with the 3D simulation engine environment disabled.

Use the **Maneuver** parameter to specify the type of maneuver. You can specify the double lane change, swept sine, sine with dwell, and slowly increasing maneuvers.

If you select the **Use maneuver-specific driver, initial position, and scene** parameter, the reference application sets the driver, initial position, and scene for the maneuver that you specified.

For more information, see Reference Generator.

Driver Commands

The Driver Commands block implements the driver model that the reference application uses to generate acceleration, braking, gear, and steering commands. By default, if you select the Reference Generator block parameter **Use maneuver-specific driver, initial position, and scene**, the reference application selects the driver for the maneuver that you specified.

Vehicle Command Mode Setting	Implementation
Longitudinal Driver	Longitudinal Driver block — Longitudinal speed-tracking controller. Based on reference and feedback velocities, the block generates normalized acceleration and braking commands that can vary from 0 through 1. Use the block to model the dynamic response of a driver or to generate the commands necessary to track a longitudinal drive cycle.
Predictive Driver	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. The controller uses a single-track (bicycle) model for optimal single-point preview control.

Vehicle Command Mode Setting	Implementation
`Predictive Stanley Driver` (default)	Predictive Driver block — Controller that generates normalized steering, acceleration, and braking commands to track longitudinal velocity and a lateral reference displacement. The normalized commands can vary between -1 to 1. • For longitudinal control, the block uses a single-track (bicycle) model for optimal single-point preview control. • For lateral control, the block uses a Stanley controller to minimize the position and angle error of the current path with respect to a reference path.
`Open Loop`	Implements an open-loop system so that you can configure the reference application for constant or signal-based steering, acceleration, braking, and gear command input.

Environment

The Environment subsystem generates the wind and ground forces. The reference application has these environment variants.

Environment	Variant	Description
Ground Feedback	`3D Engine`	Uses Simulation 3D Terrain Sensor block to implement a multipoint terrain sensor in 3D environment
	`Constant` (default)	Implements a constant friction value

Controllers

The Controllers subsystem generates engine torque, transmission gear, brake pressure, and differential pressure commands.

ECU

The ECU controller generates the engine torque command. The controller prevents over-revving the engine by limiting the engine torque command to the value specified by model workspace variable `EngRevLim`. By default, the value is 7000 rpm. If the differential torque command exceeds the limited engine torque command, the ECU sets the engine torque command to the commanded differential torque.

Transmission Control

The Transmission Controller subsystem generates the transmission gear command. The controller includes these variants.

Variant	Description
`Driver - No Clutch`	Open loop transmission control. The controller sets the gear command to the gear request.

Variant	Description
PRNDL Controller (default)	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, brake command, wheel speed, engine speed, and gear request.
Paddles	Implements a paddle controller that uses the vehicle acceleration and engine speed to generate the gear command.
Transmission Controller	Implements a transmission control module (TCM) that uses Stateflow logic to generate the gear command based on the vehicle acceleration, wheel speed, and engine speed.

Brake Pressure Control

The Brake Controller subsystem implements a Brake Pressure Control subsystem to generate the brake pressure command. The Brake Pressure Control subsystem has these variants.

Variant	Description
Bang Bang ABS	Implements an anti-lock braking system (ABS) feedback controller that switches between two states to regulate wheel slip. The bang-bang control minimizes the error between the actual slip and desired slip. For the desired slip, the controller uses the slip value at which the mu-slip curve reaches a peak value. This desired slip value is optimal for minimum braking distance.
Open Loop (default)	Open loop brake control. The controller sets the brake pressure command to a reference brake pressure based on the brake command.
Five-State ABS	Five-state ABS control when you simulate the maneuver.[1,2,3] The five-state ABS controller uses logic-switching based on wheel deceleration and vehicle acceleration to control the braking pressure at each wheel. Consider using five-state ABS control to prevent wheel lock-up, decrease braking distance, or maintain yaw stability during the maneuver. The default ABS parameters are set to work on roads that have a constant friction coefficient scaling factor of 0.6.

Active Differential Control

The Active Differential Control subsystem generates the differential pressure command. To calculate the command, the subsystem has these variants.

Variant	Description
`Rear Diff Controller`	Implements a controller that generates the differential pressure command based on the: • Steer angle • Vehicle pitch, yaw, and roll • Brake command • Wheel speed • Gear • Vehicle acceleration
`No Control` (default)	Does not implement a controller. Sets the differential pressure command to 0.

Passenger Vehicle

The Passenger Vehicle subsystem has an engine, controllers, and a vehicle body with four wheels. Specifically, the vehicle contains these subsystems.

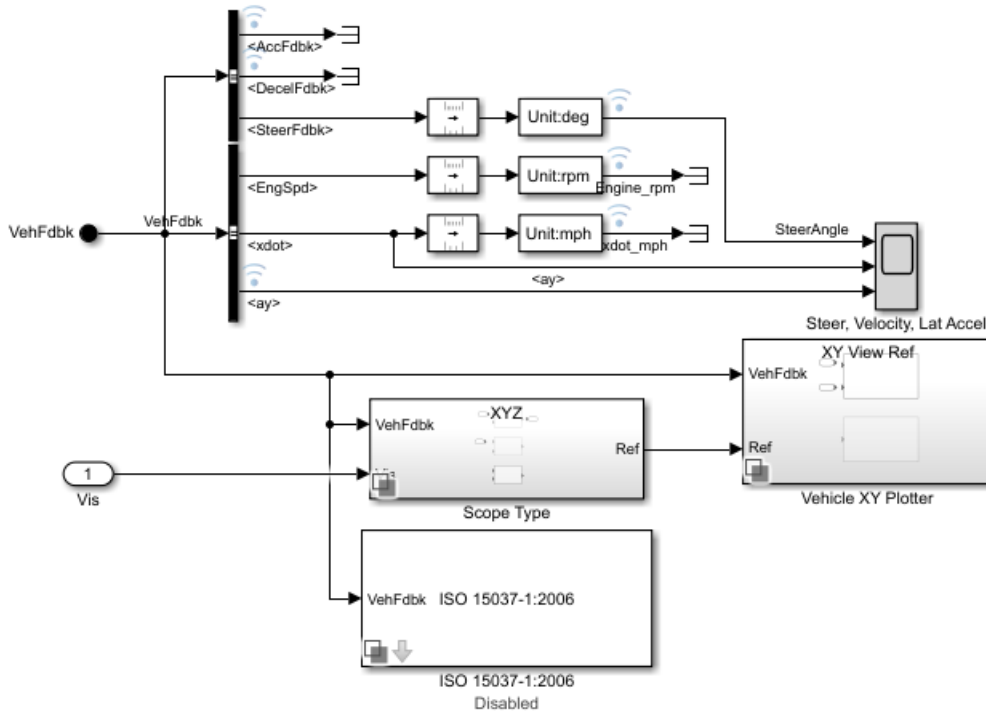| Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh7DOF	`PassVeh7DOF`	Vehicle with four wheels: • Vehicle body has three degrees-of-freedom (DOFs) — Longitudinal, lateral, and yaw • Each wheel has one DOF — Rolling Subsystem has variants for the tires, including: • Fiala • Magic Formula

Body, Suspension, Wheels Subsystem	Variant	Description
PassVeh14DOF	`PassVeh14DOF` (default)	Vehicle with four wheels. • Vehicle body has six DOFs — Longitudinal, lateral, vertical and pitch, yaw, and roll • Each wheel has two DOFs — Vertical and rolling Subsystem has variants for the suspension, including: • Double Wishbone • Independent Mapped Front • Kinematics and Compliance Independent Suspension Subsystem has variants for the tires, including: • Fiala • Magic Formula • Dugoff

Engine Subsystem	Variant	Description
Mapped Engine	`SiMappedEngine` (default)	Mapped spark-ignition (SI) engine

Steering, Transmission, Driveline, and Brakes Subsystem		Variant	Description
Driveline Ideal Fixed Gear	Driveline model	`All Wheel Drive`	Configure the driveline for all-wheel, front-wheel, rear-wheel, or rear-wheel active differential drive and specify the type of torque coupling.
		`Front Wheel Drive`	
		`Rear Wheel Drive`	
		`Rear Wheel Drive Active Differential` (default)	
	Transmission	`Ideal` (default)	Implements an ideal fixed gear transmission.
	Brake Hydraulics	NA	Implements the heuristic response of a hydraulic system when the controller applies a brake command to a cylinder. Includes front and rear wheel bias coefficients. The subsystem converts the applied pressure to a cylinder spool position. To generate the brake pressure, the spool applies a flow downstream to the cylinders.

Visualization

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.

Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Lateral acceleration
Steer, Velocity, Lat Accel Scope block	• `SteerAngle` — Steering angle versus time • `<xdot>` — Longitudinal vehicle velocity versus time • `<ay>` — Lateral acceleration versus time
Vehicle XY Plotter	Vehicle longitudinal versus lateral distance

Element	Description
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

If you enable 3D visualization on the Reference Generator block **3D Engine** tab by selecting **Enabled**, you can view the vehicle response in the Simulation 3D Viewer.

To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	

View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

References

[1] Pasillas-Lépine, William. "*Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms.*" *Vehicle System Dynamics* 44, no. 2 (2006): 173-188.

[2] Gerard, Mathieu, William Pasillas-Lépine, Edwin De Vries, and Michel Verhaegen. "*Improvements to a five-phase ABS algorithm for experimental validation*." *Vehicle System Dynamics* 50, no. 10 (2012): 1585-1611.

[3] Bosch, R. "*Bosch Automotive Handbook*." 10th ed. Warrendale, PA: SAE International, 2018.

[4] J266_199601. *Steady-State Directional Control Test Procedures for Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

[5] ISO 4138:2012. *Passenger cars — Steady-state circular driving behaviour — Open-loop test methods*. Geneva: ISO, 2012.

See Also

3D Engine | Driver Commands | Reference Generator | Simulation 3D Terrain Sensor

Related Examples

- "Vehicle Lateral Acceleration at Different Speeds" on page 1-36

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2
- Simulation Data Inspector
- "Slowly Increasing Steering Maneuver" on page 3-53

Kinematics and Compliance Virtual Test Laboratory

Model-Based Calibration Toolbox allows you to generate optimized suspension parameters for the Independent Suspension - Mapped and Solid Axle Suspension - Mapped blocks by using the kinematics (K) and compliance (C) virtual test laboratory.

To create and open a working copy of the K and C virtual test laboratory reference application, enter

```
vdynblksKandCTestLabStart
```

The K and C virtual test laboratory contains vehicle, test system, and test control subsystems. The vehicle system has two variants:

- `Simscape Multibody Vehicle` — Vehicle with a Simscape Multibody suspension system
- `VDBS Vehicle` — Vehicle with an Independent Suspension - Mapped block
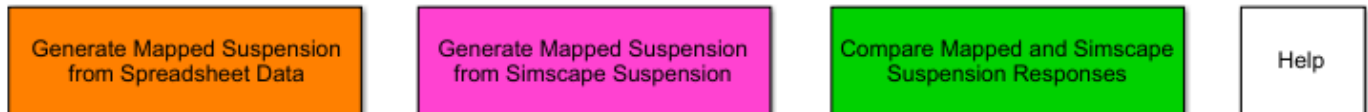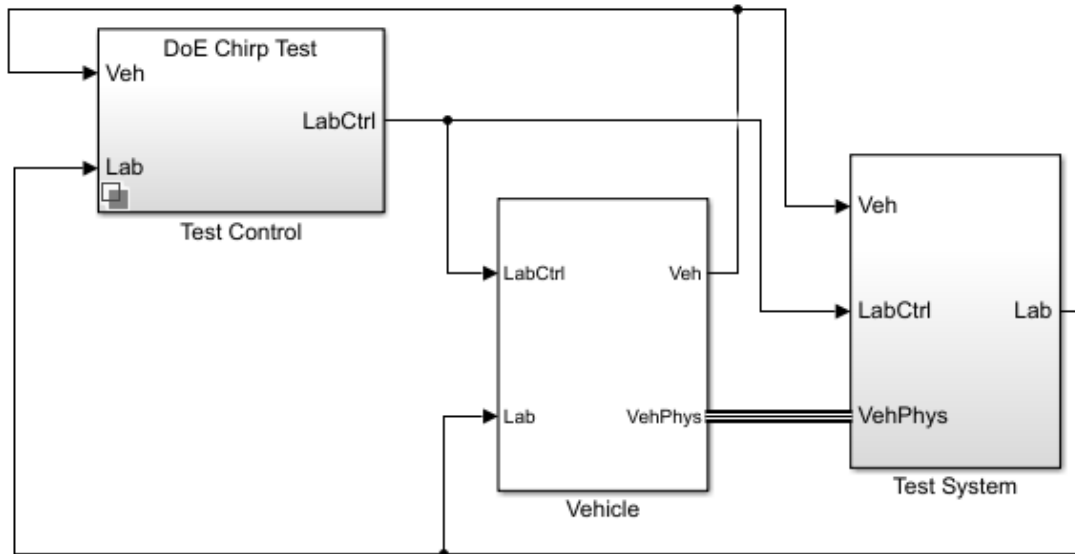
This table summarizes the virtual test laboratory tests.

Test	Objective	Method
Generate Mapped Suspension from Spreadsheet Data	Use measured vertical force and suspension geometry data to generate calibrated suspension parameters for the mapped suspension blocks. **Note** You can use a third-party simulation model to generate the measured suspension data.	The virtual test lab uses Model-Based Calibration Toolbox to fit camber angle, toe angle, and vertical force response models for the data. The virtual test lab then uses the response models to generate suspension parameters for the suspension blocks.
Generate Mapped Suspension from Simscape Suspension	Use a Simscape Multibody suspension system to generate calibrated suspension parameters for the suspension mapped blocks.	The virtual test lab uses Model-Based Calibration Toolbox to perform a Sobol sequence design of experiments (DoE) on the suspension height and handwheel angle operating points. At each operating point, the reference application stimulates the Simscape Multibody suspension system with a chirp signal over a frequency range of 0.1 to 2 Hz. The virtual test lab then uses the data to fit the suspension vertical force, camber angle, and toe angle with a Gaussian process model (GPM) as a function of the suspension state. Finally, the reference application uses the GPM to generate suspension parameters for the suspension blocks.
Compare Mapped and Simscape Suspension Responses	Compare the mapped suspension with the Simscape Multibody suspension results.	The virtual test laboratory stimulates the Simscape Multibody suspension at one operating point and then compares the response to the mapped suspension.

Generate Mapped Suspension from Spreadsheet Data

The virtual test lab uses Model-Based Calibration Toolbox to fit camber angle, toe angle, and vertical force response models for the data. The virtual test lab then uses the response models to generate suspension parameters for the suspension blocks.
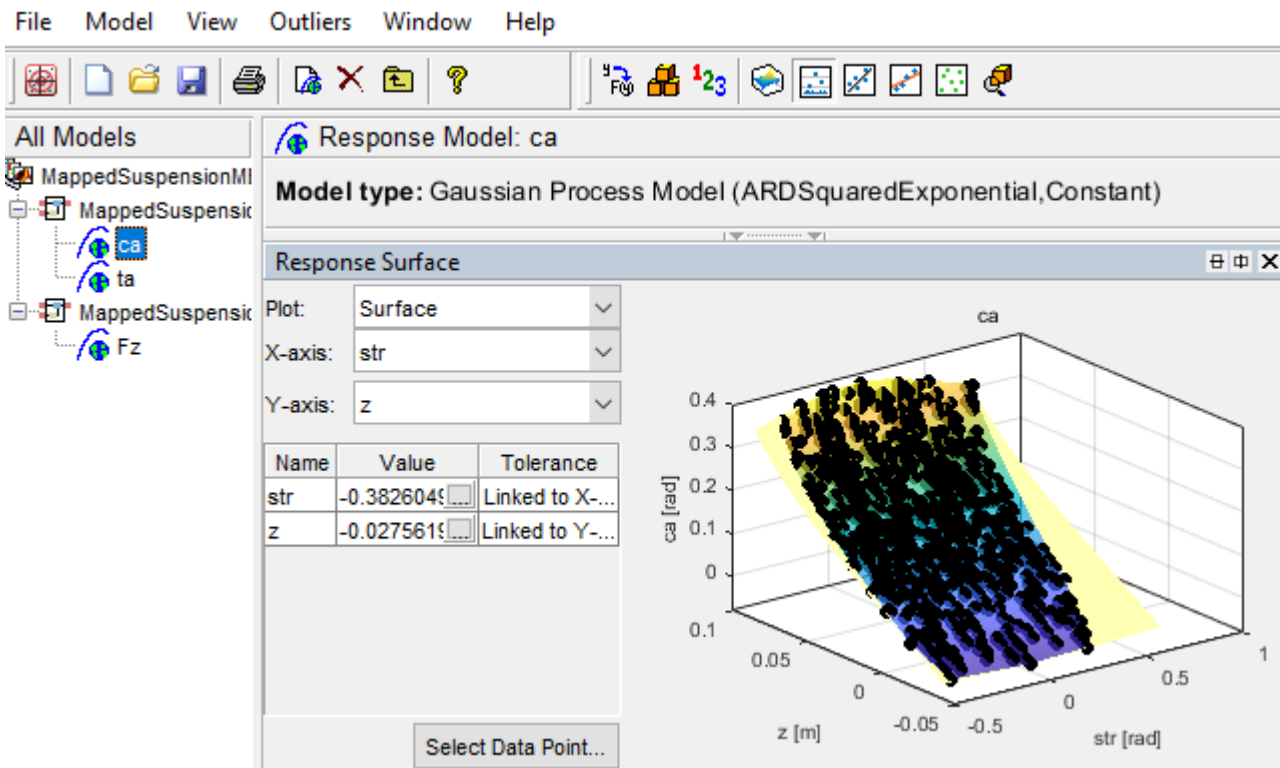
Generate Mapped Suspension Calibration

1   Use the **Spreadsheet file** field to provide a data file. By default, the reference application has KandCTestData.xlsx containing required data. The table summarizes the data file requirements for generating calibrated tables.

Data	Description	Data Requirements for Generating Mapped Suspension Tables
z	Vertical axis suspension height, in m	*Required*
zdot	Vertical axis suspension height velocity breakpoints, in m/s	*Required*
str	Steering angle, in rad	*Required*
Fz	Vertical axis suspension force, in N	*Required*
ca	Camber angle, in rad	*Required*
ta	Toe angle, in rad	*Required*

2   Click **Generate mapped suspension calibration** to generate response surface models in Model-Based Calibration Toolbox.

The model browser opens when the process completes. You can view the camber angle, *ca*, toe angle, *ta*, and vertical force, *Fz*, response model fits for the data.

Apply Calibration to Mapped Suspension Model

1   Click **Apply calibration to mapped suspension model**. The virtual test lab uses the response models to generate calibrated suspension and breakpoint data.

2   Click **OK** to update the model workspace and suspension blocks.

In the Model Explorer, you can view the generated suspension parameters.

Parameter	Model Workspace Variable	Description
Axle breakpoints, f_susp_axl_bp	`f_susp_axl_bp`	Axle breakpoints, P, dimensionless.
Vertical axis suspension height breakpoints, f_susp_dz_bp	`f_susp_dz_bp`	Vertical axis suspension height breakpoints, M, in m.
Vertical axis suspension height velocity breakpoints, f_susp_dzdot_bp	`f_susp_dzdot_bp`	Vertical axis suspension height velocity breakpoints, N, in m/s.

Vertical axis suspension force and moment responses, f_susp_fmz	`f_susp_fmz`	M-by-N-by-O-by-P-by-4 array of output values as a function of:
		
		 • Vertical suspension height, M
		 • Vertical suspension height velocity, N
		 • Steering angle, O
		 • Axle, P
		 • 4 output types
		
		    • 1 — Vertical force, in N·m
		    • 2 — User-defined
		    • 3 — Stored energy, in J
		    • 4 — Absorbed power, in W

Parameter	Model Workspace Variable	Description

Suspension geometry responses, f_susp_geom	`f_susp_geom`	M-by-O-by-P-by-3 array of geometric suspension values as a function of:
		
		• Vertical suspension height, *M*
		• Steering angle, *O*
		• Axle, *P*
		• 3 output types
		
		   • 1 — Camber angle, in rad
		   • 2 — Caster angle, in rad
		   • 3 — Toe angle, in rad

Steering angle breakpoints, f_susp_strgdelta_bp	`f_susp_strgdelta_bp`	Steering angle breakpoints, *O*, in rad.

Generate Mapped Suspension from Simscape Suspension

The virtual test lab uses Model-Based Calibration Toolbox to perform a Sobol sequence design of experiments (DoE) on the suspension height and handwheel angle operating points. At each operating point, the reference application stimulates the Simscape Multibody suspension system with a chirp signal over a frequency range of 0.1 to 2 Hz. The virtual test lab then uses the data to fit the suspension vertical force, camber angle, and toe angle with a Gaussian process model (GPM) as a function of the suspension state. Finally, the reference application uses the GPM to generate suspension parameters for the suspension blocks.

The test laboratory exercises the suspension system with the DOE settings contained in the DOEMAT array. To view the DOEMAT array values, open the Model Explorer.

Element	Description
`DOEMAT(1,1)`	Suspension height
`DOEMAT(1,2)`	Handwheel angle
`DOEMAT(1,3)`	Chirp signal amplitude
`DOEMAT(1,4)`	Starting chirp frequency
`DOEMAT(1,5)`	Ending chirp frequency
`DOEMAT(1,6)`	Simulation time to complete chirp signal frequency range

The generation can take time to run and slow other computer processes. View progress in the MATLAB® window.

In the Model Explorer, you can view the generated suspension parameters.

Parameter	Model Workspace Variable	Description
Axle breakpoints, f_susp_axl_bp	`f_susp_axl_bp`	Axle breakpoints, P, dimensionless.
Vertical axis suspension height breakpoints, f_susp_dz_bp	`f_susp_dz_bp`	Vertical axis suspension height breakpoints, M, in m.
Vertical axis suspension height velocity breakpoints, f_susp_dzdot_bp	`f_susp_dzdot_bp`	Vertical axis suspension height velocity breakpoints, N, in m/s.

Vertical axis suspension force and moment responses, f_susp_fmz	`f_susp_fmz`	M-by-N-by-O-by-P-by-4 array of output values as a function of:
		
		• Vertical suspension height, M
		• Vertical suspension height velocity, N
		• Steering angle, O
		• Axle, P
		• 4 output types
		
		   • 1 — Vertical force, in N·m
		   • 2 — User-defined
		   • 3 — Stored energy, in J
		   • 4 — Absorbed power, in W

Parameter	Model Workspace Variable	Description

Suspension geometry responses, f_susp_geom	`f_susp_geom`	M-by-O-by-P-by-3 array of geometric suspension values as a function of:
		
		• Vertical suspension height, *M*
		
		• Steering angle, *O*
		
		• Axle, *P*
		
		• 3 output types
		
		   • 1 — Camber angle, in rad
		   • 2 — Caster angle, in rad
		   • 3 — Toe angle, in rad

Steering angle breakpoints, f_susp_strgdelta_bp	`f_susp_strgdelta_bp`	Steering angle breakpoints, *O*, in rad.

Compare Mapped and Simscape Suspension Responses

The virtual test laboratory stimulates the Simscape Multibody suspension at one operating point and then compares the response to the mapped suspension.

• To stimulate the Simscape Multibody suspension model, the test laboratory uses with the DOE settings contained in the `DOEMAT` array.

During the simulation, to view the suspension system, select the **Mechanics Explorers** tab.
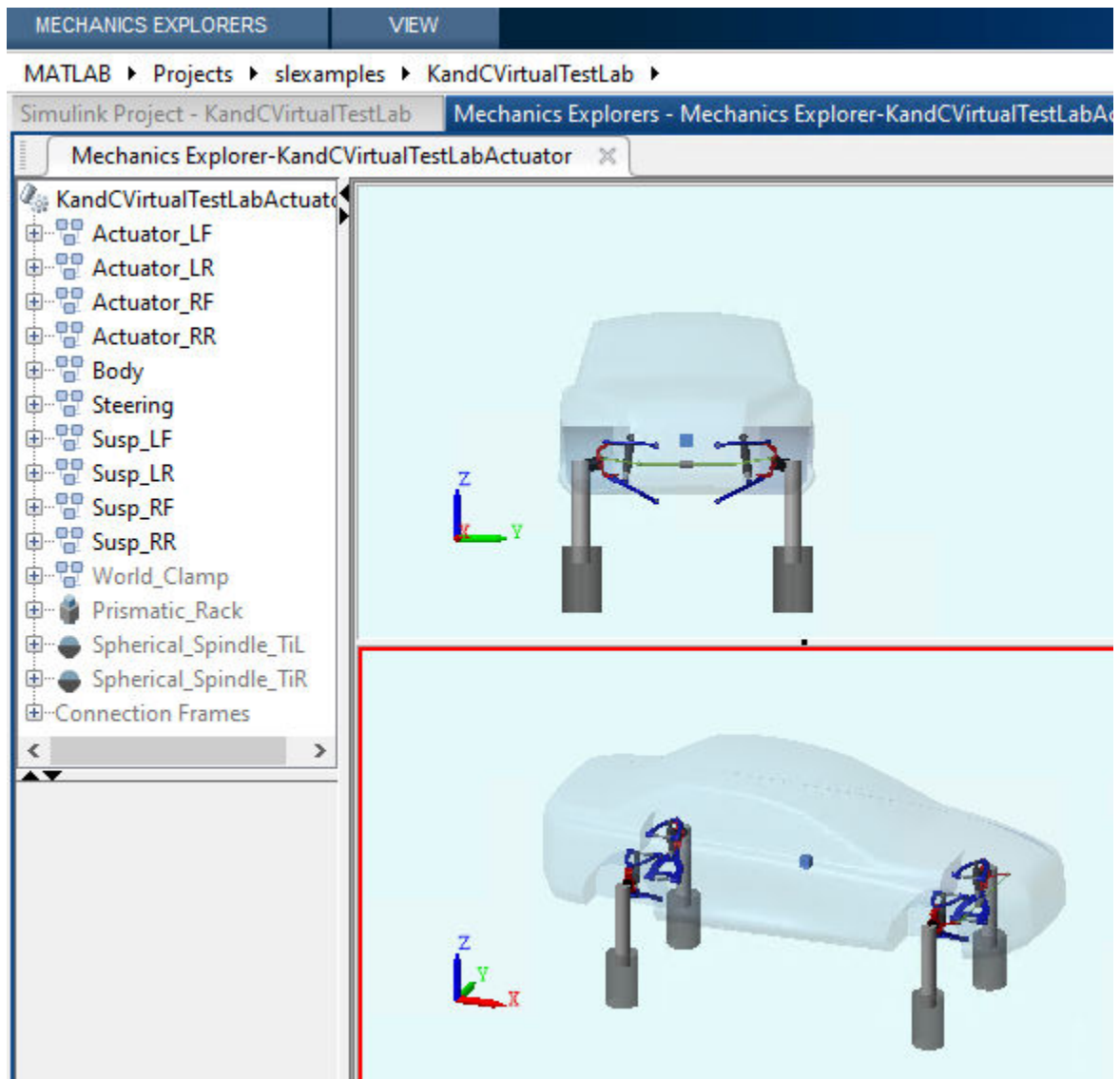
- After the simulation completes, use the Simulation Data Inspector to compare the suspension system response for the mapped suspension and Simscape Multibody suspension model. For example, compare the vertical force, camber angle, and toe angle responses.
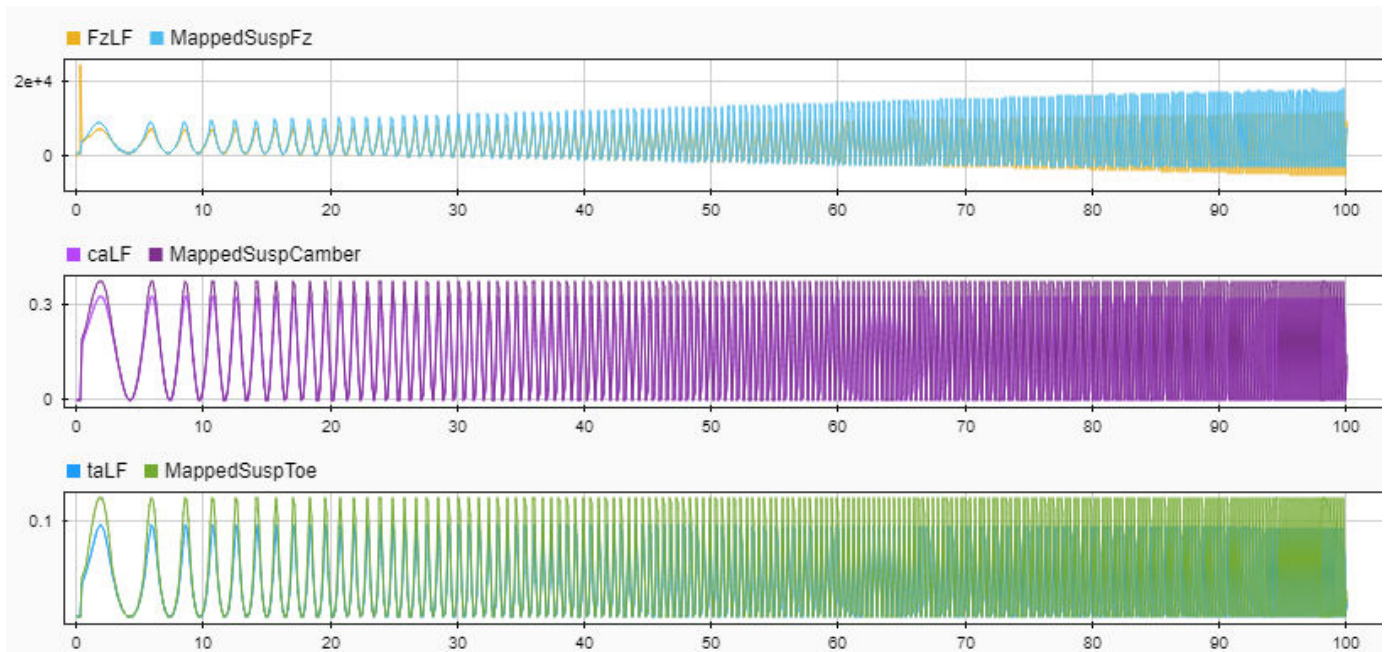
See Also

Independent Suspension - Mapped | Solid Axle Suspension - Mapped

More About

- Simulation Data Inspector

Run a Vehicle Dynamics Maneuver in 3D Environment

This example shows how to run a vehicle dynamics maneuver in a 3D environment. By integrating vehicle dynamics models with a 3D environment, you can test advanced driver assistance systems (ADAS) and automated driving (AD) perception, planning, and control software. For the 3D visualization engine platform requirements and hardware recommendations, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

1 Create and open a working copy of a maneuver reference application. For example, open the double-lane change reference application.

vdynblksDblLaneChangeStart

2 Run the maneuver simulation. By default, the 3D environment is disabled.

When you run the simulation, the Visualization subsystem provides driver, vehicle, and response information. The reference application logs vehicle signals during the maneuver, including steering, vehicle and engine speed, and lateral acceleration. You can use the Simulation Data Inspector to import the logged signals and examine the data.
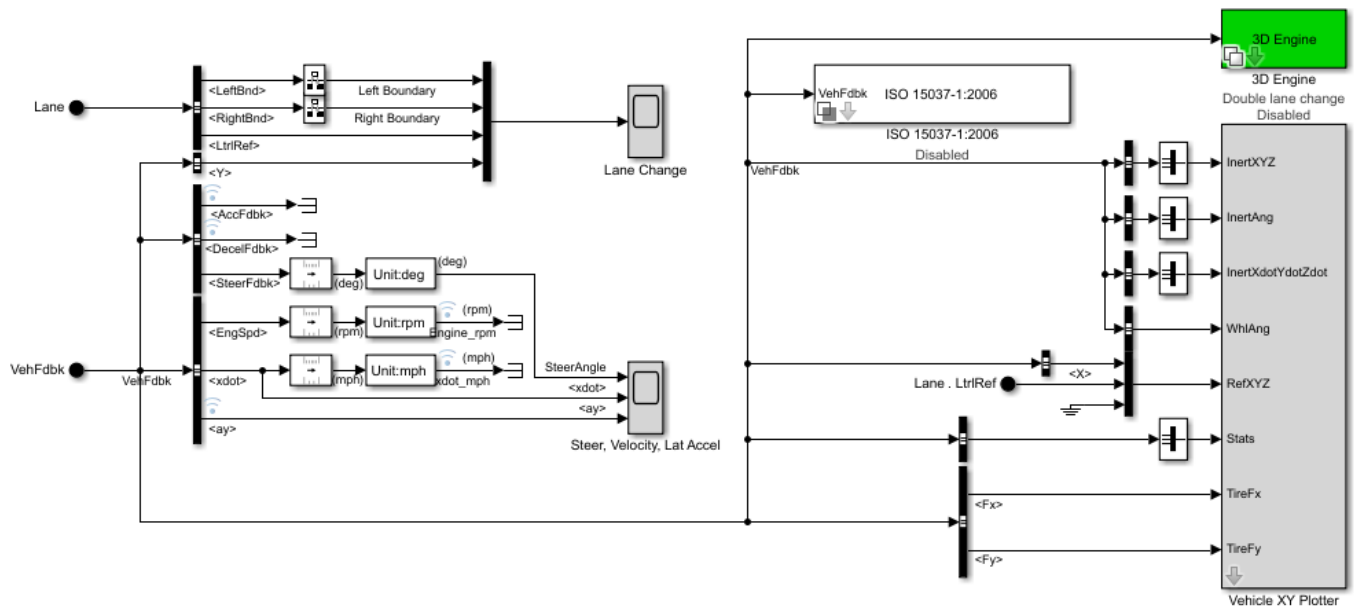
Element	Description
Driver Commands	Driver commands: • Handwheel angle • Acceleration command • Brake command
Vehicle Response	Vehicle response: • Engine speed • Vehicle speed • Acceleration command

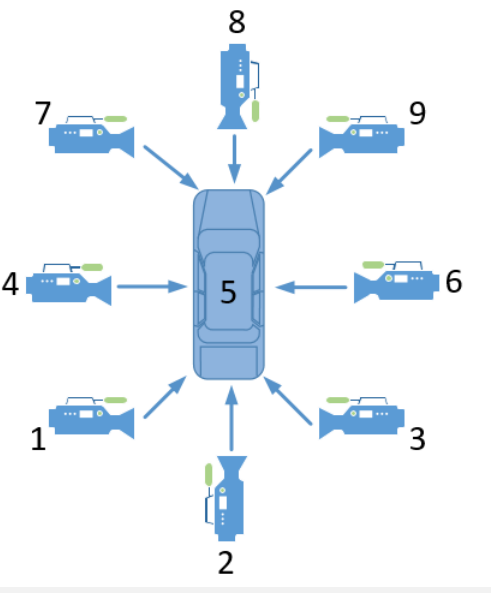| Element	Description
Lane Change Scope block	Lateral vehicle displacement versus time: • Red line — Cones marking right lane boundary • Orange line — Cones marking left lane boundary • Blue line — Reference trajectory • Green line — Actual trajectory
Steer, Velocity, Lat Accel Scope block	• `SteerAngle` — Steering angle versus time • `<xdot>` — Longitudinal vehicle velocity versus time • `<ay>` — Lateral acceleration versus time
Vehicle XY Plotter	Vehicle longitudinal versus lateral distance
ISO 15037-1:2006 block	Display ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle

3   Enable the 3D visualization environment. In the Visualization subsystem, open the 3D Engine block. Set these parameters.

- **3D Engine** to **Enabled**.
- **Scene name** to one of the scenes, for example `Double lane change`.
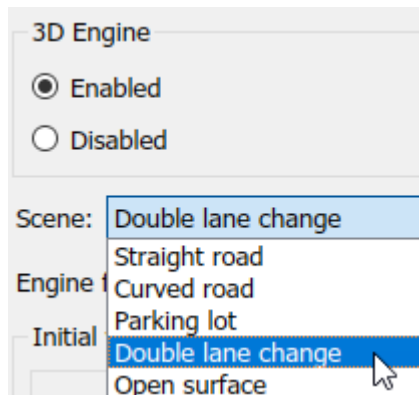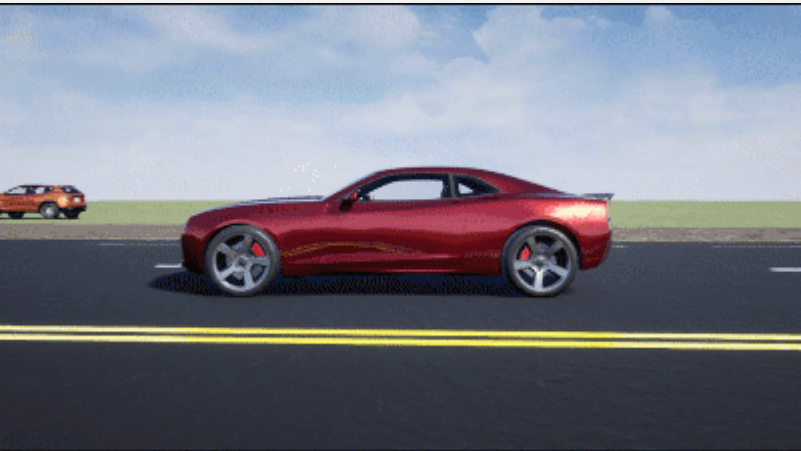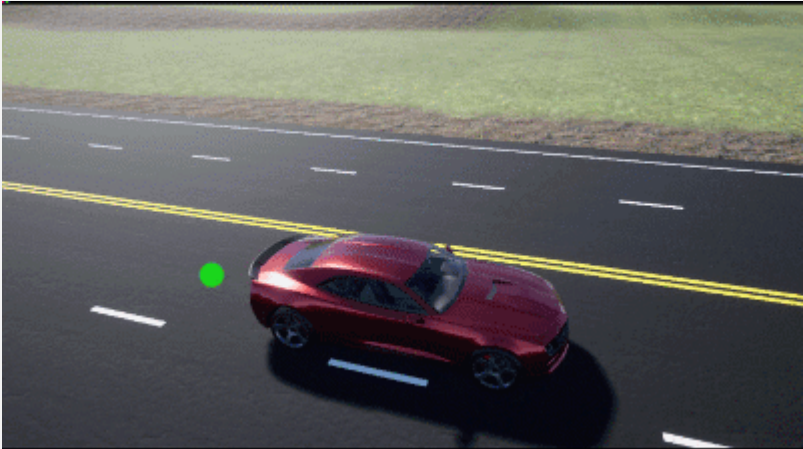
- To position the vehicle in the scene:

    a   Select the position initialization method:

    - **Recommended for scene** — Set the initial vehicle position to values recommended for the scene
    - **User-specified** — Set your own initial vehicle position

    b   Click **Update the model workspaces with the initial values** to overwrite the initial vehicle position in the model workspaces with the applied values.

4   Rerun the reference application. As the simulation runs, in the Simulation 3D Viewer, view the vehicle response.

To smoothly change the camera views, use these key commands.

Key	Camera View
1	Back left
2	Back
3	Back right
4	Left
5	Internal
6	Right
7	Front left
8	Front
9	Front right
0	Overhead

View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

For example, when you run the double-lane change maneuver, use the cameras to visualize the vehicle as it changes lanes.

- Back

- Front left

- Internal

Note

- To open and close the Simulation 3D Viewer, use the Simulink Run and Stop buttons. If you manually close the Simulation 3D Viewer, Simulink stops the simulation with an error.
- When you enable the 3D visualization environment, you cannot step the simulation back.

See Also

More About

- "Double-Lane Change Maneuver" on page 3-22
- "Slowly Increasing Steering Maneuver" on page 3-53
- "Swept-Sine Steering Maneuver" on page 3-41
- Simulation Data Inspector
- "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8

Send and Receive Double-Lane Change Scene Data

This example shows you how to use the Simulation 3D Message Set and Simulation 3D Message Get blocks to communicate with the 3D visualization environment when you run the double-lane change maneuver. Specifically, you use the:

- Simulation 3D Message Get block to retrieve which barrels the vehicle hits during the maneuver.
- Simulation 3D Message Set block to control the traffic signal light.

For the minimum hardware required to run the example, see the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

Run a Double-Lane Change Maneuver

With the 3D visualization environment enabled, run a double-lane change maneuver.

1   Create and open a working copy of the double-lane change reference application project.

```
vdynblksDblLaneChangeStart
```

2   Enable the 3D visualization environment. In the Visualization subsystem, open the 3D Engine block mask and select **Enabled**. Apply the changes and save the model.

Alternatively, at the MATLAB command prompt, enter this code.

See Code That Enables 3D Environment

```
% Enable the 3D visualization environment
mdl = 'DLCReferenceApplication';
set_param([mdl '/Visualization/3D Engine'],'engine3D','Enabled');
save_system(mdl)
```

3   Run the maneuver for 25 seconds. View the simulation in the Simulation 3D Viewer. As the vehicle completes the maneuver, it hits the barrel.

See Code That Runs Simulation

```
% Run simulation for 25s.
mdl = 'DLCReferenceApplication';
set_param(mdl,'StopTime','25');
```

```
save(mdl);
sim(mdl);
```

Use Simulation 3D Message Get Block to Retrieve Barrel Data

Use the Simulation 3D Message Get block to retrieve which barrels the vehicle hits during the maneuver. By default, the maneuver uses the double-lane change scene.

1   Navigate to the Visualization > 3D Engine subsystem. Right-click the 3D Engine block and select
    Mask > Look Under Mask. In the Visualization > 3D Engine > 3D Engine subsystem, insert
    these blocks:

    • Simulation 3D Message Get
    • Display

2   Set the Simulation 3D Message Get block parameters so that the block retrieves barrel data from
    the double-lane change scene. Set these block parameters, apply the changes, and save the
    model.

    • **Signal name, SigName** to `NumOfBarricadesHit`
    • **Data type, DataType** to `boolean`
    • **Message size, MsgSize** to `[1 8]`
    • **Sample time** to `-1`
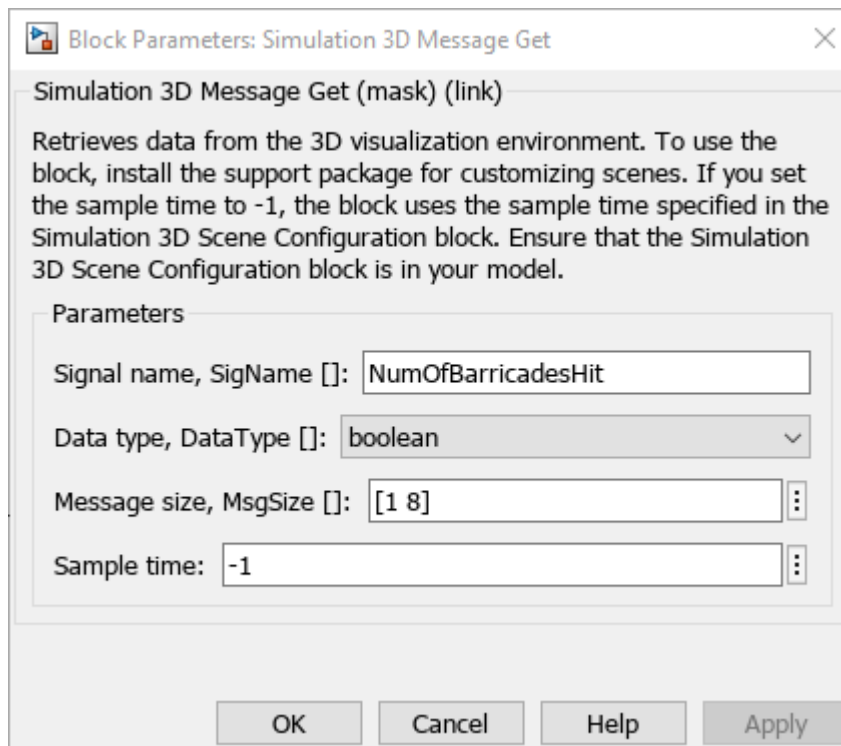
Alternatively, at the MATLAB command prompt, enter this code.

See Code That Sets Parameters

```
% Set these Simulation 3D Message Get block parameters
visualss='DLCReferenceApplication/Visualization/3D Engine/3D Engine';
```

```
set_param([visualss '/Simulation 3D Message Get'],'SigName','NumOfBarricadesHit');
set_param([visualss '/Simulation 3D Message Get'],'DataType','boolean');
set_param([visualss '/Simulation 3D Message Get'],'MsgSize','[1 8]');
set_param([visualss '/Simulation 3D Message Get'],'Ts','-1');
save_system(mdl)
```

3   Connect the Simulation 3D Message Get and Display blocks as shown. Confirm the block parameters. Save the model.

Simulation 3D Message Get

Simulation 3D Scene Configuration
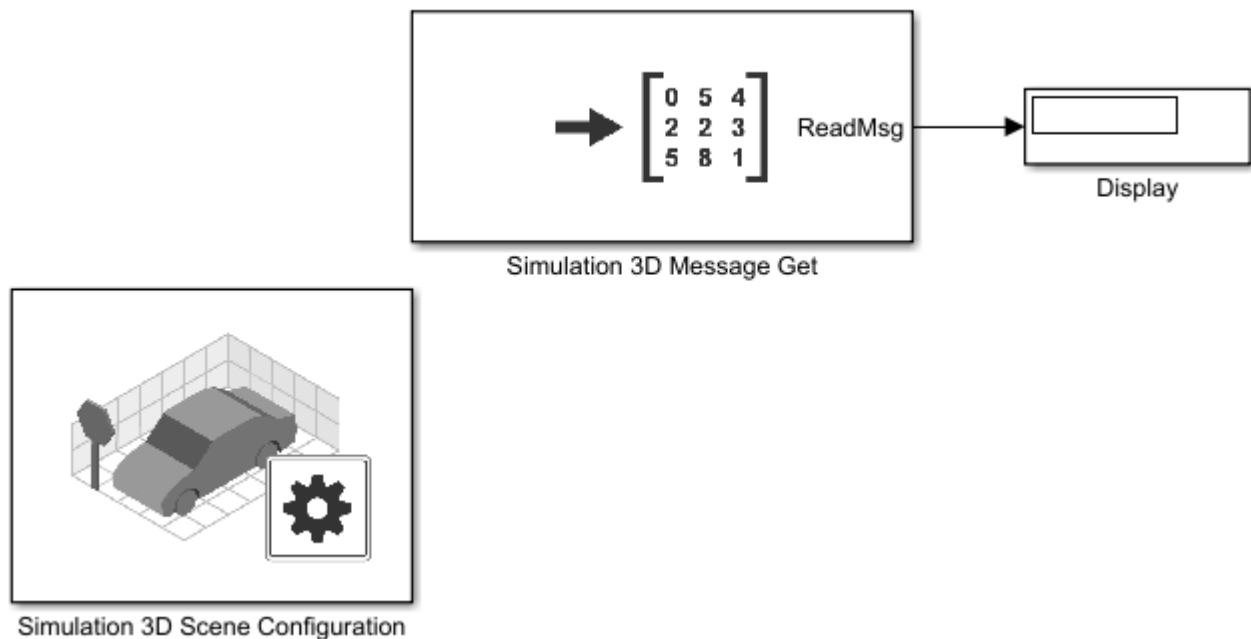
4   Verify that the Simulation 3D Scene Configuration block executes before the Simulation 3D Message Get block. That way, the Unreal Engine 3D visualization environment prepares the data before the Simulation 3D Message Get block receives it. To check the block execution order, right-click the blocks and select **Properties**. On the **General** tab, confirm these **Priority** settings:

- Simulation 3D Scene Configuration — `0`
- Simulation 3D Message Get — `1`

For more information about execution order, see "Control and Display Execution Order".

5   Run the maneuver. As the simulation runs, the display block updates with the `ReadMsg` boolean value `1` when the vehicle hits the corresponding barrel.

Display
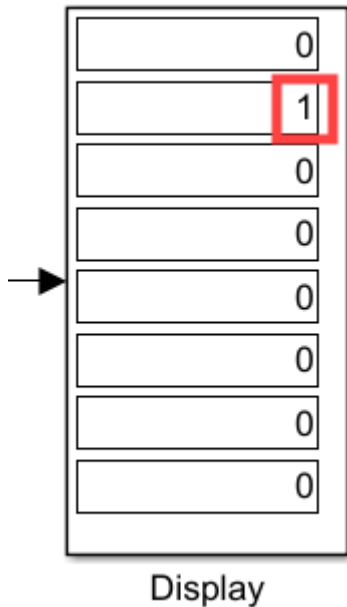
The results indicate that the vehicle hits `Drum01Node67` at the end of the maneuver.

This table provides the Double Lane Change scene barrel name that corresponds to the `ReadMsg` array element.

Simulation 3D Message Get Block ReadMsg Value	Unreal® Editor Barrel Name
ReadMsg(1,1)	Drum01Node
ReadMsg(1,2)	Drum01Node67
ReadMsg(1,3)	Drum01Node68
ReadMsg(1,4)	Drum01Node69
ReadMsg(1,5)	Drum01Node70
ReadMsg(1,6)	Drum01Node71
ReadMsg(1,7)	Drum01Node72
ReadMsg(1,8)	Drum01Node73

Use Simulation 3D Message Set Block to Control Traffic Signal Light

1 Start the maneuver at 5 seconds. In the Lane Change Reference Generator block, set **Maneuver start time** to 5.

See Code That Sets Parameters

```
% Start simulation at 5s.
mdl = 'DLCReferenceApplication';
set_param([mdl '/Lane Change Reference Generator'],'t_start','5');
save(mdl);
```

2 Navigate to the Visualization > 3D Engine subsystem. Right-click the 3D Engine block and select **Mask > Look Under Mask**. In the Visualization > 3D Engine > 3D Engine subsystem, insert these blocks:

- Simulation 3D Message Set
- Repeating Sequence Stair

3 Set the Simulation 3D Message Set block parameters so that the block sends traffic signal data to the double-lane change scene. Set these block parameters, apply the changes, and save the model.

- **Signal name, SigName** to `TrafficLight1`
- **Sample time** to `-1`

This table provides the scene traffic signal light color that corresponds to the `WriteMsg` value in the Double Lane Change scene.

Simulation 3D Message Set Block WriteMsg Value	TrafficLight1 Color
0	Red
1	Yellow
2	Green

Alternatively, at the MATLAB command prompt, enter this code.

See Code That Sets Parameters

```
% Set Simulation 3D Message Set block parameters
visualss='DLCReferenceApplication/Visualization/3D Engine/3D Engine';
set_param([visualss  '/Simulation 3D Message Set'],'SigName','TrafficLight1');
set_param([visualss  '/Simulation 3D Message Set'],'Ts','-1');
save_system(mdl)
```

4 Set the Repeating Sequence Stair block parameters to send a command that corresponds to red, yellow, and green traffic light signals. Set these block parameters, apply the changes, and save the model.

- **Vector of output values:** to [0 0 0 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2].'
- **Sample time** to 1

- **Output data type** to `int32`

Alternatively, at the MATLAB command prompt, enter this code. Apply the block changes and save the model.

See Code That Sets Parameters

```
% Set Repeating Sequence Stair block parameters
visualss='DLCReferenceApplication/Visualization/3D Engine/3D Engine';
open_system([visualss  '/Repeating Sequence Stair']);
set_param([visualss  '/Repeating Sequence Stair'],'OutValues',"[0 0 0 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
set_param([visualss  '/Repeating Sequence Stair'],'tsamp','1');
set_param([visualss  '/Repeating Sequence Stair'],'OutDataTypeStr','int32');
```

5 Connect the blocks as shown. Confirm the block parameters and signal connections. Save the model.

Simulation 3D Message Set

6   Verify that the Simulation 3D Message Set block executes before the Simulation 3D Scene
     Configuration block. That way, Simulation 3D Message Set prepares the signal data before the
     Unreal Engine 3D visualization environment receives it. To check the block execution order,
     right-click the blocks and select **Properties**. On the **General** tab, confirm these **Priority**
     settings:

     • Simulation 3D Scene Configuration — `0`
     • Simulation 3D Message Set — `-1`

     For more information about execution order, see "Control and Display Execution Order".

7   Run the maneuver. As the simulation runs, in the Simulation 3D Viewer, you can see the
     `TrafficLight1` light change from red to yellow to green.

3-99

Time Range (s)	WriteMsg Value	TrafficLight1 Color
0–3	0	Red
3–5	1	Yellow
5–30	2	Green

See Also

Double Lane Change | Simulation 3D Message Get | Simulation 3D Message Set | 3D Engine | Simulation 3D Scene Configuration

Related Examples

- "Double Lane Change Reference Application" on page 7-7
- "Yaw Stability on Varying Road Surfaces" on page 1-16

More About

- "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8

- "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25
- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

Start Double-Lane Change Maneuver at Target Velocity

This example shows you how to use the steady-state operating points to *start* the maneuver at the target velocity set point. When you start the simulation with the vehicle at rest, the vehicle accelerates until it achieves the target velocity before it starts the maneuver. The simulation run-time includes the time for getting the vehicle up-to-speed. For example, with the default double-lane change maneuver settings, the simulation takes ~11 s to achieve the target velocity and ~17 s to start the maneuver. The maneuver takes ~5 s of 25 s of simulation time.
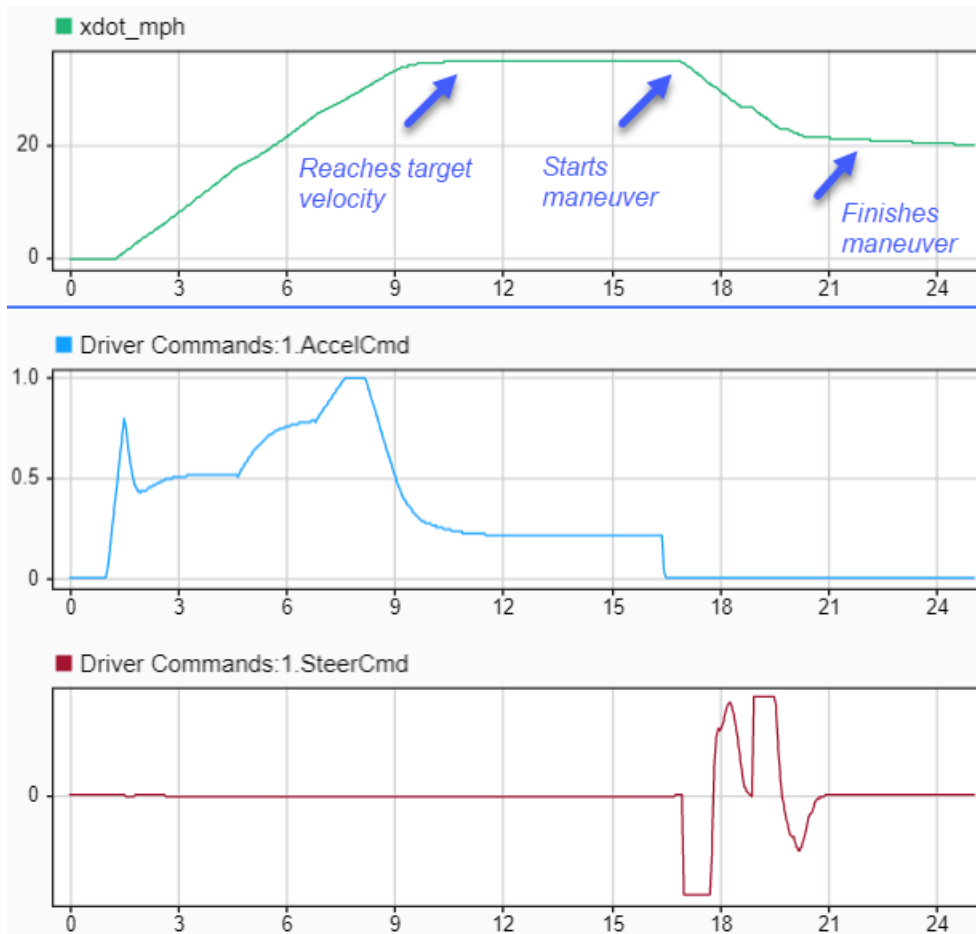

To save simulation time, you can start the simulation at the target velocity. First, you find the steady-state conditions when the vehicle is operating at the target velocity. Once you have the steady-state solution, you can use it to initialize the vehicle and start the maneuver at the target velocity.

Follow these steps.

1   Create and open a working copy of the double-lane change reference application project.

    vdynblksDblLaneChangeStart

2   On the Lane Change Reference Generator block, set the **Steady-state initial conditions** parameter to Solve using block parameters.

    The block **Longitudinal entrance velocity setpoint, xdot_r** parameter specifies a target velocity of 35 mph.
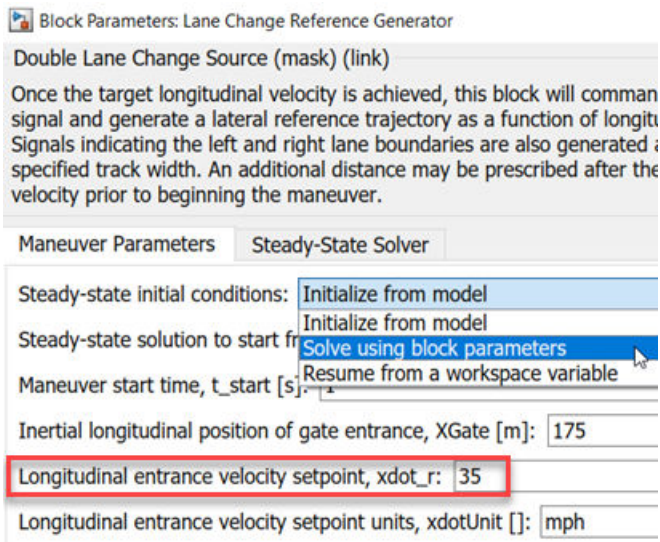
3 On the **Steady-State Solver** tab, verify the initial conditions, workspace variable, and solver setting parameters. For this example, set **Workspace variable name to generate, ssWSName** to `dlcSS35mph`.
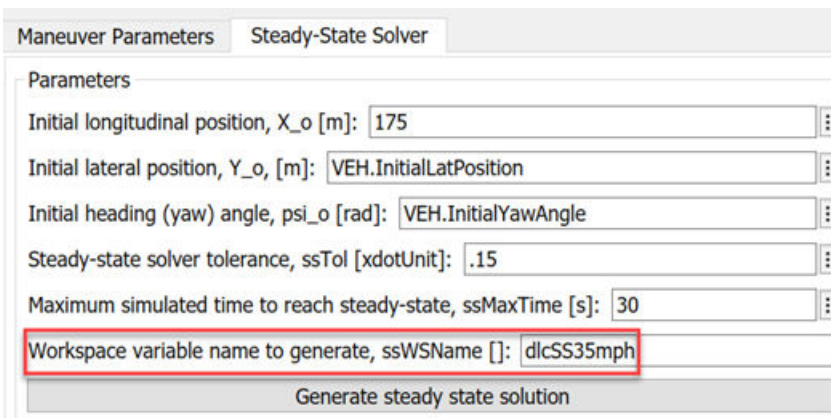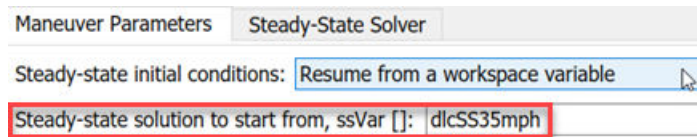

Click **Apply**.

4 Click **Generate steady state solution**. After the simulation completes, examine the `dlcSS35mph` workspace variable. It contains the logged states for approximately 40 model state variables at the steady-state operating points, including the suspension.

Note Verify that generating the steady-state solution created or updated the workspace. If the model cannot find a steady-state solution, try different parameter or solver settings.

5   On the Lane Change Reference Generator block, set:

- **Steady-state initial conditions** to `Resume from a workspace variable`.
- **Steady-state solution to start from, ssVar** to the workspace variable that you specified in step 3. For this example, set it to `dlcSS35mph`.

Click **Apply**.

6   Run the simulation.

7   Examine the results. The simulation starts at the steady-state operating point with the vehicle at the target velocity of 35 mph. The vehicle maneuver takes ~5 s of 7 s of simulation time. This is 18 s less than the original simulation time.
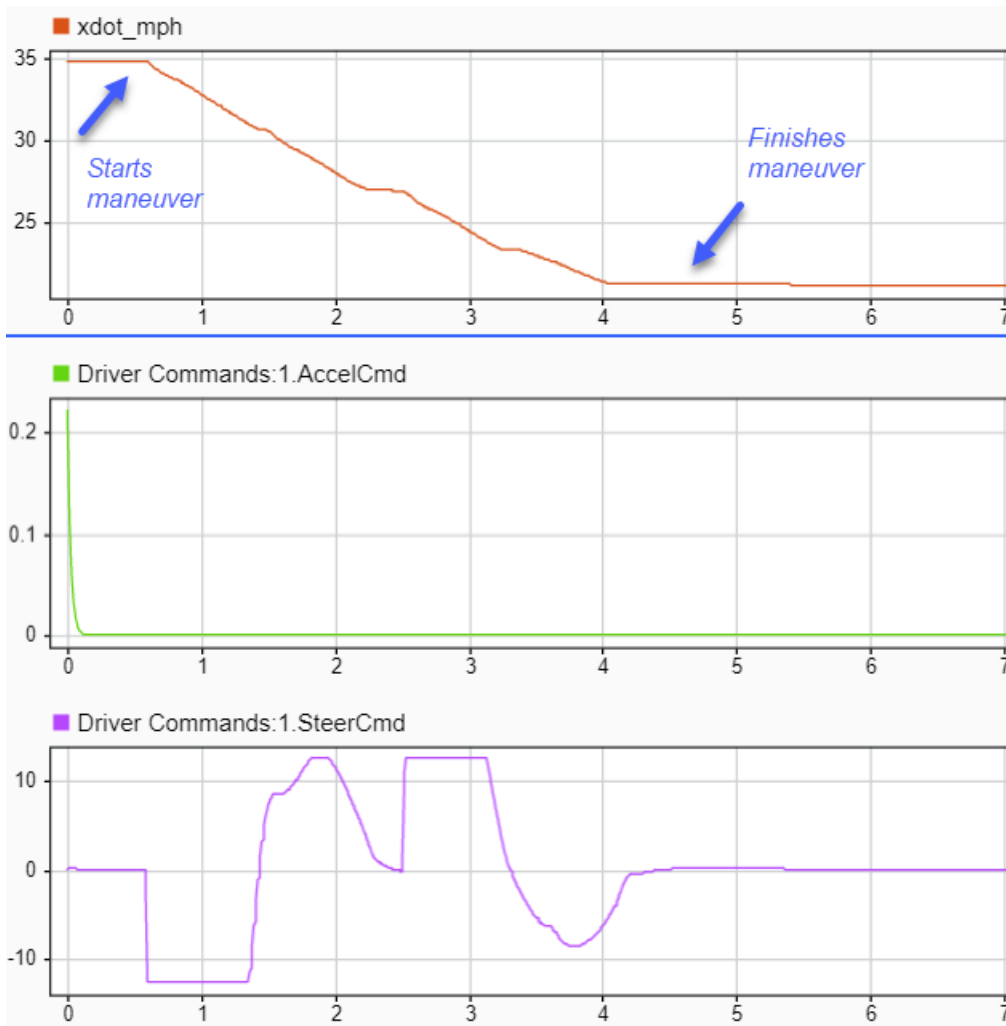
See Also

Lane Change Reference Generator

Related Examples

- "Double-Lane Change Maneuver" on page 3-22

Project Templates

Vehicle Dynamics Blockset Project Templates

Vehicle Dynamics Blockset provides preassembled vehicle dynamics models that you can use to analyze the dynamic system response to common ride and handling tests. Use the templates to create vehicle dynamic model variants for the maneuver reference applications. Open project files that contain the vehicle models from the Simulink start page.

1 In Simulink, on the **Simulation** tab, select **New > Project > New Project**.

In the Simulink start page, browse to Vehicle Dynamics Blockset and select **Passenger 3DOF Vehicle**, **Passenger 7DOF Vehicle**, or **Passenger 14DOF Vehicle**.

2 In the Create Project dialog box, in **Project name**, enter a project name.

3 In **Folder**, enter a project folder or browse to the folder to save the project.

4 Click **OK**.

If the folder does not exist, the dialog box prompts you to create it. Click **Yes**.

The software compiles the project and populates the project folders. All models and supporting files are in place for you to customize your vehicle dynamics model.

This table summarizes the vehicle dynamics project templates.

Vehicle Model	Description	Vehicle Body Degrees-of-Freedom (DOFs)				Wheel DOFs			
Passenger 14DOF Vehicle	• Vehicle with four wheels • Available as model variant in the maneuver reference applications	Six				Two per wheel - eight total			
		Translational		**Rotational**		**Translational**		**Rotational**	
		Longitudinal	✓	Pitch	✓	Vertical	✓	Rolling	✓
		Lateral	✓	Yaw	✓				
		Vertical	✓	Roll	✓				
Passenger 7DOF Vehicle	• Vehicle with four wheels • Available as model variant in the maneuver reference applications	Three				One per wheel - four total			
		Translational		**Rotational**		**Rotational**			
		Longitudinal	✓	Pitch		Rolling		✓	
		Lateral	✓	Yaw	✓				
		Vertical		Roll					
Passenger 3DOF Vehicle	• Vehicle with ideal tire	Three				None			
		Translational		**Rotational**					
		Longitudinal	✓	Pitch					
		Lateral	✓	Yaw	✓				
		Vertical		Roll					

See Also

More About

- "Double-Lane Change Maneuver" on page 3-22
- "Slowly Increasing Steering Maneuver" on page 3-53
- "Swept-Sine Steering Maneuver" on page 3-41

Maneuver Standards

ISO 15037-1:2006 Standard Measurement Signals

You can configure the maneuver reference applications to display ISO 15037-1:2006[1] standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle.

To configure the ISO signal display, in the reference application Visualization subsystem, open the ISO 15037-1:2006 block. Select **Enabled**. After you run the maneuver, the Simulation Data Inspector opens with standard measurements.

For example, to display the ISO signals when you run the double lane change maneuver:

1 Create and open a working copy of the double-lane change reference application project.

   `vdynblksDblLaneChangeStart`

2 In the Visualization subsystem, open the ISO 15037-1:2006 block. Select **Enabled**. Save the reference application.

3 Run the maneuver. As the simulation runs, view the ISO standard measurement signals in the Simulation Data Inspector, including steering wheel angle and torque, longitudinal and lateral velocity, and sideslip angle.
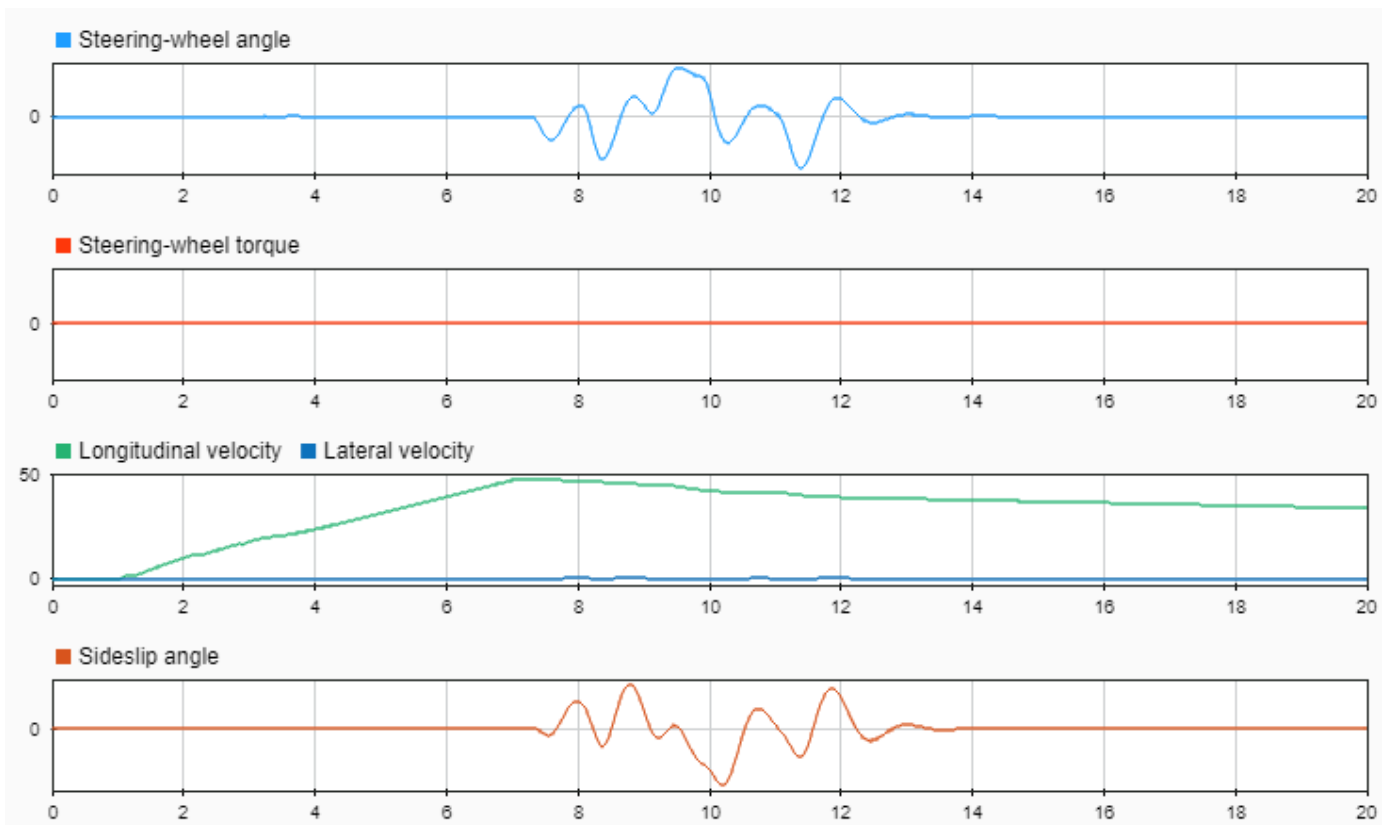
References

[1] ISO 15037-1:2006. *Road vehicles -- Vehicle dynamics test methods -- Part 1: General conditions for passenger cars*. ISO (International Organization for Standardization), 2014.

See Also

More About

- "Double-Lane Change Maneuver" on page 3-22
- "Slowly Increasing Steering Maneuver" on page 3-53
- "Swept-Sine Steering Maneuver" on page 3-41
- Simulation Data Inspector

External Websites

- International Organization for Standardization

Supporting Data

Support Package for Maneuver and Drive Cycle Data

This example shows how to install additional maneuver and drive cycle data from a support package. By default, the Drive Cycle Source block has the FTP-75 drive cycle data. The support package has drive cycles that include the gear shift schedules, for example `JC08` and `CUEDC`.

1   In the Drive Cycle Source block, click **Install additional drive cycles** to start the installer.
2   Follow the instructions and default settings provided by the installer to complete the installation.
3   On the **Select a support package** screen, select the data you want to add:

Accept or change the **Installation folder** and click **Next**.

Note You must have write permission for the Installation folder.

See Also
Drive Cycle Source

Prepare Custom Vehicle Mesh for the Unreal Editor

This example shows you how to create a vehicle mesh that is compatible with the project in the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. You can specify the mesh in the Simulation 3D Vehicle or Simulation 3D Vehicle with Ground Following block to visualize the vehicle in the Unreal Editor when you run a simulation.

Before you start, install the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. See "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8.

To create a compatible custom vehicle mesh, follow these workflow steps.

Step	Description
"Step 1: Setup Bone Hierarchy" on page 6-3	In a 3D creation environment, setup the vehicle mesh bone hierarchy and specify part names.
"Step 2: Assign Materials" on page 6-4	Optionally, assign materials to the vehicle parts.
"Step 3: Export Mesh and Armature" on page 6-4	Export the vehicle mesh and armature in .fbx file format.
"Step 4: Import Mesh to Unreal Editor" on page 6-5	Import the vehicle mesh into the Unreal Editor.
"Step 5: Set Block Parameters" on page 6-5	Set up the Simulation 3D Vehicle or Simulation 3D Vehicle with Ground Following block parameters.

Note To create the mesh, this example uses the 3D creation software Blender® Version 2.80.

Step 1: Setup Bone Hierarchy

1 Import a vehicle mesh into a 3D modeling tool, for example Blender.
2 To ensure that this mesh is compatible with the animation components in the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, use this naming convention for the vehicle parts in the mesh.

Vehicle Part	Name
Chassis	VehicleBody
Front left wheel	Wheel_FL
Front right wheel	Wheel_FR
Rear left wheel	Wheel_RL
Rear right wheel	Wheel_RR
Steering wheel	Wheel_Steering
Left headlight	Lights_Headlight_Left
Right headlight	Lights_Headlight_Right

Vehicle Part	Name
Left indicator light	Indicator_L
Right indicator light	Indicator_R
Number plate	Vehicle_Plate
Brake lights	Lights_Brake
Reverse lights	Lights_Reverse
Front left brake caliper	BrakeCaliper_FL
Front right brake caliper	BrakeCaliper_FR
Rear left brake caliper	BrakeCaliper_RL
Rear right brake caliper	BrakeCaliper_RR

3   Set the vehicle body object, **VehicleBody** as the parent of the wheel objects and other vehicle objects.

Step 2: Assign Materials

Optionally, assign material slots to vehicle parts. In this example, the mesh uses one material for the chassis and one for the four wheels.

1   Create and assign material slots to the vehicle chassis. Confirm that the first vehicle slot corresponds to the vehicle body.

2   Create and assign material slots to the wheels.

Step 3: Export Mesh and Armature

Export the mesh and armature in the .fbx file format. For example, in Blender:

1   On the **Object Types** pane, select **Armature** and **Mesh**.

2   On the **Transform** pane, set:

- **Scale** to 1.00
- **Apply Scalings** to All Local
- **Forward** to X Forward
- **Up** to Z Up

Select **Apply Unit**.

3   On the **Geometry** pane:

- Set **Smoothing** to Face
- Select **Apply Modifiers**

4   On the **Armature** pane, set:

- **Primary Bone Axis** to X Axis
- **Secondary Bone Axis** to Z Axis

Select **Export FBX**.

Step 4: Import Mesh to Unreal Editor

1   Open the Unreal Engine `AutoVrtlEnv.uproject` project in the Unreal Editor.

2   In the editor, import the FBX® file as a skeletal mesh. Assign the **Skeleton** to the
`SK_PassengenerVehicle_Skeleton` asset.
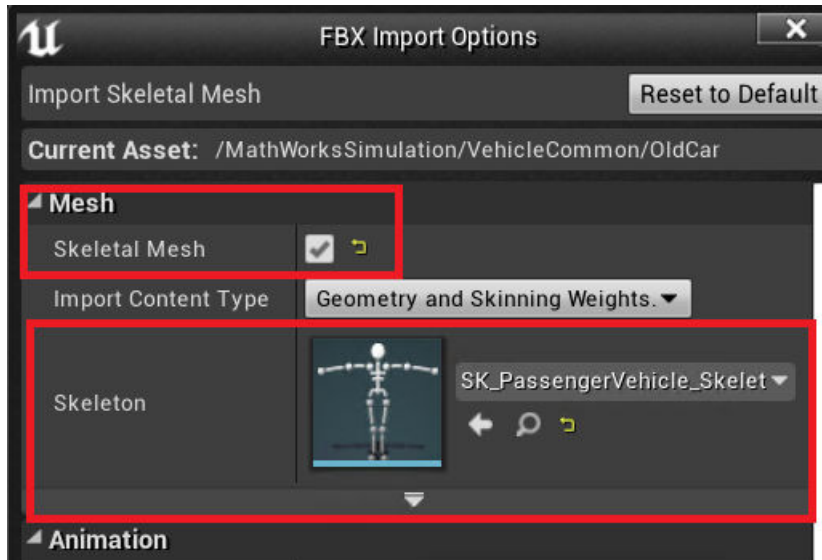
Step 5: Set Block Parameters

In your Simulink model, set these Simulation 3D Vehicle or Simulation 3D Vehicle with Ground
Following block parameters:

- **Type** to `Custom`.
- **Path** to the path in the Unreal Engine project that contains the imported mesh.

See Also
Simulation 3D Scene Configuration | Simulation 3D Vehicle with Ground Following | Simulation 3D
Vehicle

More About

- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8
- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

External Websites

- Blender

Build Light in Unreal Editor

Follow these steps to build light in the Unreal Editor. You can also use the `AutoVrtlEnv` project lighting in a custom scene.

1   In the editor, from the **Main Toolbar**, click the down-arrow next to **Build** to expand the options.

2   Under **Build**, select **Lighting Quality > Production** to rebuild production quality maps. Rebuilding complex maps can be time-intensive.
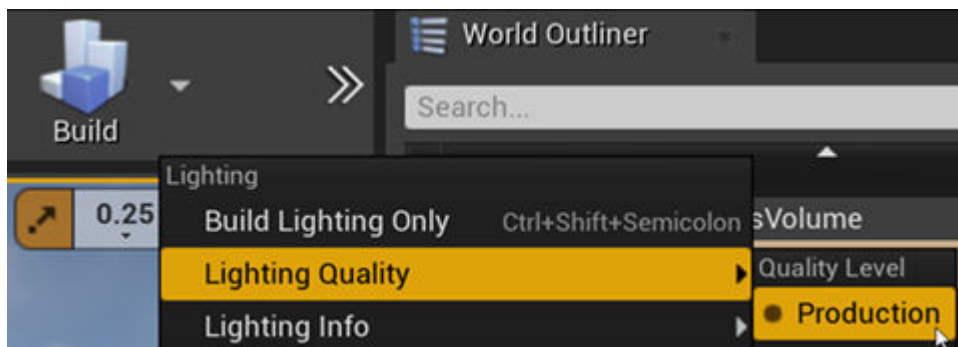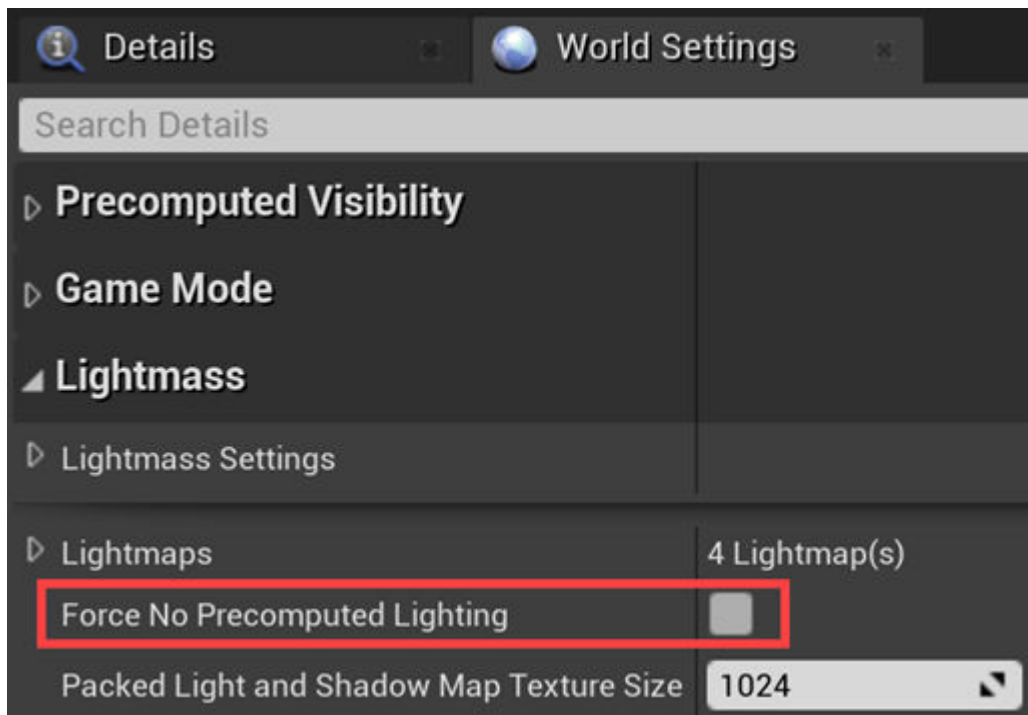

3   Click the **Build** icon to build the game. Production-quality lighting takes the a long time to build.

Use AutoVrtlEnv Project Lighting in Custom Scene

To use the lighting that comes installed with the `AutoVrtlEnv` project in Vehicle Dynamics Blockset, follow these steps.

1   On the **World Settings** tab, clear **Force no precomputed lighting**.

2 Under **Build**, select **Lighting Quality > Production** to rebuild the maps with production quality. Rebuilding complex maps can be time-intensive.
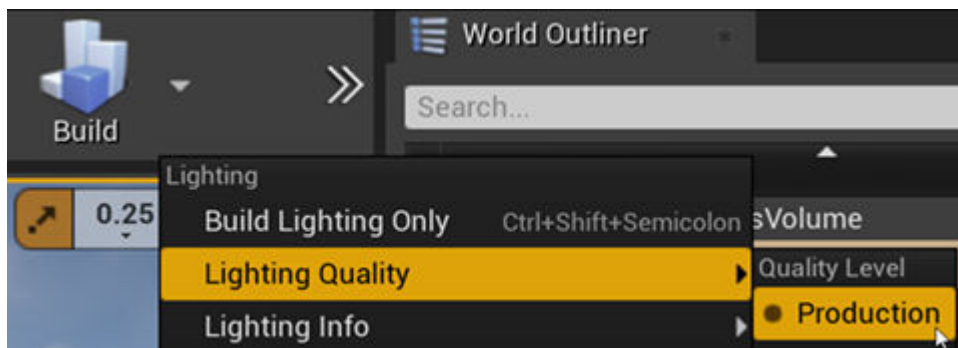
See Also

Simulation 3D Scene Configuration

More About

- "Animate Custom Actors in the Unreal Editor" on page 8-21
- "Create Empty Project in Unreal Engine" on page 6-49
- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

External Websites

- Unreal Engine

Customize 3D Scenes for Vehicle Dynamics Simulations

Vehicle Dynamics Blockset contains prebuilt scenes in which to simulate and visualize the performance of vehicles modeled in Simulink. These scenes are visualized using a standalone Unreal Engine executable within the toolbox. If you have the Unreal from Epic Games and the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects installed, you can customize these scenes. You can also use the Unreal Editor and the support package to simulate within scenes from your own custom project.

With custom scenes, you can co-simulate in both Simulink and the Unreal Editor so that you can modify your scenes between simulation runs. To customize scenes, you should be familiar with creating and modifying scenes in the Unreal Editor.
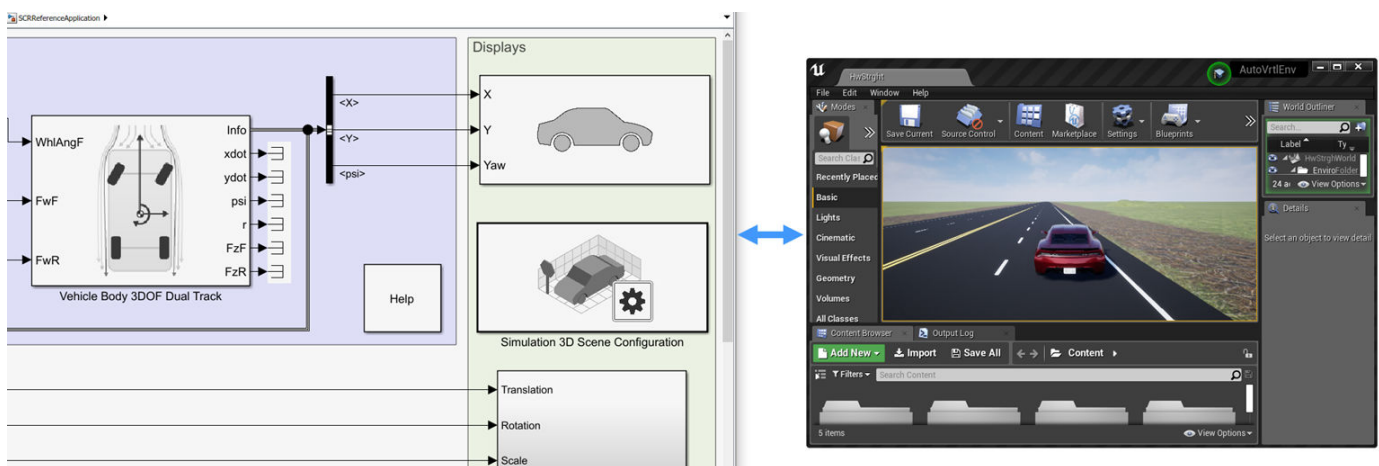
To customize 3D scenes, follow these steps:

1 "Install Support Package and Configure Environment" on page 6-10
2 "Migrate Projects Developed Using Prior Support Packages" on page 6-12
3 "Customize Scenes Using Simulink and Unreal Editor" on page 6-14
4 "Package Custom Scenes into Executable" on page 6-23

See Also
Simulation 3D Scene Configuration

Related Examples
• "Send and Receive Double-Lane Change Scene Data" on page 3-93

More About
• "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8
• "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

External Websites
• Unreal Engine

- Unreal Engine 4 Documentation
- Using Unreal Engine with Simulink

Install Support Package and Configure Environment

To customize scenes in your installation of the Unreal Editor and simulate within these scenes in Simulink, you must first install and configure the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package.

Note These installation instructions apply to **R**2023**a**. If you are using a previous release, see the documentation for Other Releases.


Verify Software and Hardware Requirements

Before installing the support package, make sure that your environment meets the minimum software and hardware requirements described in "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

Install Support Package

To install the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, follow these steps:

1 On the MATLAB **Home** tab, in the **Environment** section, select **Add-Ons > Get Add-Ons**.
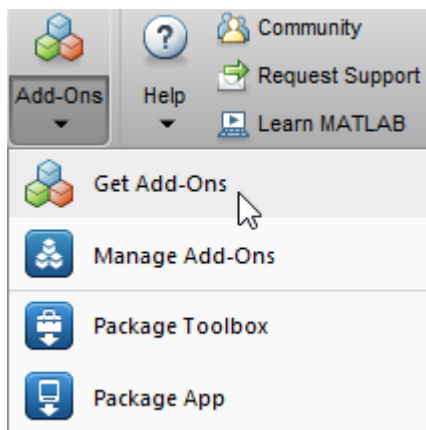
2 In the Add-On Explorer window, search for the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. Click **Install**.

Note You must have write permission for the installation folder.

Configure Environment

The Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package includes these components.

- An Unreal project, defined in `AutoVrtlEnv.uproject`, and its associated files. The project includes editable versions of the prebuilt 3D scenes that you can select from the **Scene name** parameter of the Simulation 3D Scene Configuration block.

- Three plugins, `MathWorkSimulation`: `RoadRunnerMaterials`, and `MathWorksAutomotiveContent`. These plugins establish the connection between MATLAB and the Unreal Editor and are required for co-simulation.

To configure your environment so that you can customize scenes, use `copyExampleSim3dProject` to copy the support package components to a folder on your local machine. For example, this code copies the files to `C:\project`.

```
sim3d.utils.copyExampleSim3dProject("C:\project");
```

If you want to use a project developed using a prior release of the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, you must migrate the project to make it compatible with Unreal Editor 4.27. See "Migrate Projects Developed Using Prior Support Packages" on page 6-12. Otherwise, you can "Customize Scenes Using Simulink and Unreal Editor" on page 6-14.

Note If you want to use the plugins to co-simulate with more than one Unreal project, see Unreal Engine 4.27 Plugins.

See Also

Simulation 3D Scene Configuration | `copyExampleSim3dProject`

More About

- "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8

Migrate Projects Developed Using Prior Support Packages

After you install the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package as described in "Install Support Package and Configure Environment" on page 6-10, you may need to migrate your project. If your Simulink model uses an Unreal Engine executable or project developed using a prior release of the support package, you must migrate the project to make it compatible with Unreal Editor 4.27. Follow these steps:

1   Open Unreal Engine 4.27. For example, navigate to `C:\Program Files\Epic Games \UE_4.27\Engine\Binaries\Win64` and open `UE4Editor.exe`.

2   Use the Unreal Project Browser to open the project that you want to migrate.

3   Follow the prompts to open a copy of the project. The editor creates a new project folder in the same location as the original, appended with 4.27. Close the editor.
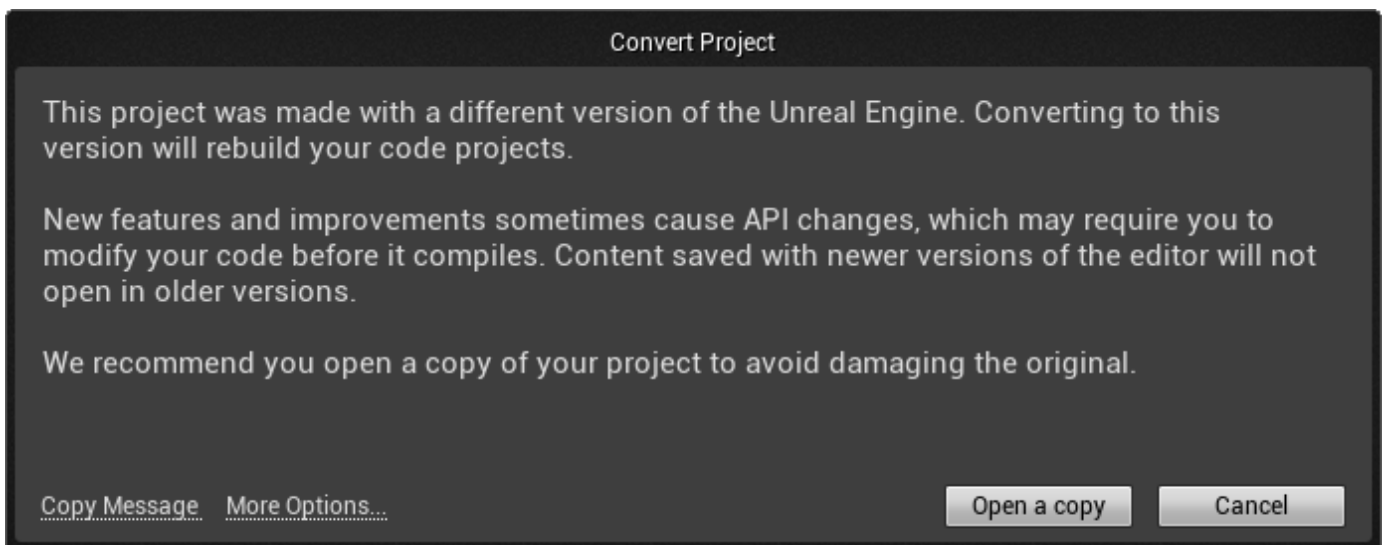
4   In a file explorer, remove any spaces in the migrated project folder name. For example, rename `MyProject 4.27` to `MyProject4.27`.

5   Use MATLAB to open the migrated project in Unreal Editor 4.27. For example, if you have a migrated project saved to the `C:/Local` folder, use this MATLAB code:

```
path = fullfile('C:','Local','MyProject4.27','MyProject.uproject');
editor = sim3d.Editor(path);
open(editor);
```

Note The support package may includes changes in the implementation of some actors. Therefore, if the original project contains actors that are placed in the scene, some of them might not fully migrate to Unreal Editor 4.27. To check, examine the Output Log.
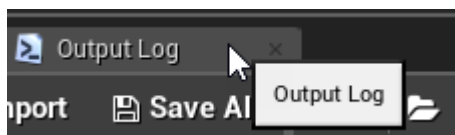
The log might contain error messages. For more information, see the Unreal Engine 4 Documentation or contact MathWorks Technical Support.

6 Optionally, after you migrate the project, you can use the project to create an Unreal Engine executable. See "Package Custom Scenes into Executable" on page 6-23.

After you migrate the project, you can create custom scenes. See "Customize Scenes Using Simulink and Unreal Editor" on page 6-14.

Tip If your project cannot locate the support package plugins, you may need to copy the plugins to the Unreal plugin folder or the Unreal project folder.

See Also
Simulation 3D Scene Configuration

More About
• "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8

Customize Scenes Using Simulink and Unreal Editor

After you install the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package as described in "Install Support Package and Configure Environment" on page 6-10, you can simulate in custom scenes simultaneously from both the Unreal Editor and Simulink. By using this co-simulation framework, you can add vehicles and sensors to a Simulink model and then run this simulation in your custom scene.

To use a project that you developed using a prior release of the support package, first migrate the project to be compatible with Unreal Engine 4.27. See "Migrate Projects Developed Using Prior Support Packages" on page 6-12.

Open Unreal Editor

Simulink will fail to establish a connection with the editor if you open the Unreal Editor outside MATLAB or Simulink. To establish this connection, you must open your project from a Simulink model or use a MATLAB function.

The first time that you open the Unreal Editor, you might be asked to rebuild `UE4Editor` DLL files or the `AutoVrtlEnv` module. Click **Yes** to rebuild these files or modules. The editor also prompts you that new plugins are available. Click **Manage Plugins** and verify that the **MathWorks Interface** plugin is installed. This plugin is the `MathWorksSimulation.uplugin` file that you copied into your Unreal Editor installation in "Install Support Package and Configure Environment" on page 6-10.

Messages about files with the name `'_BuiltData'` indicate missing lighting data for the associated level. You should rebuild these levels' lighting before shipping an executable

If you receive a warning that the lighting needs to be rebuilt, from the toolbar above the editor window, select **Build > Build Lighting Only**. The editor issues this warning the first time you open a scene or when you add new elements to a scene. To use the lighting that comes installed with `AutoVrtlEnv` in Vehicle Dynamics Blockset, see "Use AutoVrtlEnv Project Lighting in Custom Scene" on page 6-17.

Open Unreal Editor from Simulink

1   Open a Simulink model configured to simulate in the 3D environment. At a minimum, the model must contain a Simulation 3D Scene Configuration block.

2   In the Simulation 3D Scene Configuration block of this model, set the **Scene source** parameter to `Unreal Editor`.

3   In the **Project** parameter, browse for the project file that contains the scenes that you want to customize.

    For example, this sample path specifies the `AutoVrtlEnv` project that comes installed with the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package.

    `C:\Local\AutoVrtlEnv\AutoVrtlEnv.uproject`

    This sample path specifies a custom project.

    `Z:\UnrealProjects\myProject\myProject.uproject`

4   Click **Open Unreal Editor**. The Unreal Editor opens and loads a scene from your project.

Open Unreal Editor Using Command-Line Function

To open the `AutoVrtlEnv.uproject` file that was copied from the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, specify the path to where you copied this project. For example, if you copied the `AutoVrtlEnv.uproject` to `C:/Local/AutoVrtlEnv`, use this code:

```
path = fullfile('C:','Local','AutoVrtlEnv','AutoVrtlEnv.uproject');
editor = sim3d.Editor(path);
open(editor);
```

The editor opens the `AutoVrtlEnv.uproject` file.

To open your own project, use the same commands used to open the `AutoVrtlEnv.uproject` file. Update the `path` variable with the path to your `.uproject` file. For example, if you have a project saved to the `C:/Local` folder, use this code:

```
path = fullfile('C:','Local','myProject','myProject.uproject');
editor = sim3d.Editor(path);
open(editor);
```

Reparent Actor Blueprint

Note If you are using a scene from the `AutoVtrlEnv` project that comes installed with the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, skip this section. However, if you create a new scene based off of one of the scenes in this project, then you must complete this section.

The first time that you open a custom scene from Simulink, you need to associate, or reparent, this project with the **Sim3dLevelScriptActor** level blueprint used in Vehicle Dynamics Blockset. The level blueprint controls how objects interact with the 3D environment once they are placed in it. Simulink returns an error at the start of simulation if the project is not reparented. You must reparent each scene in a custom project separately.

To reparent the level blueprint, follow these steps:

1 In the Unreal Editor toolbar, select **Blueprints** > **Open Level Blueprint**.

2 In the Level Blueprint window, select **File** > **Reparent Blueprint**.

3 Click the **Sim3dLevelScriptActor** blueprint. If you do not see the **Sim3dLevelScriptActor** blueprint listed, use these steps to check that you have the `MathWorksSimulation` plugin installed and enabled:

    a In the Unreal Editor toolbar, select **Settings** > **Plugins**.

    b In the Plugins window, verify that the **MathWorks Interface** plugin is listed in the installed window. If the plugin is not already enabled, select the **Enabled** check box.

        If you do not see the **MathWorks Interface** plugin in this window, repeat step 3 in "Configure Environment" on page 6-10 and reopen the editor from Simulink.

    c Close the editor and reopen it from Simulink.

4 Close the Level Blueprint window.

Create or Modify Scenes in Unreal Editor

After you open the editor, you can modify the scenes in your project or create new scenes.

Open Scene

In the Unreal Editor, scenes within a project are referred to as levels. Levels come in several types, and scenes have a level type of map.

To open a prebuilt scene from the `AutoVrtlEnv.uproject` file, in the **Content Browser** pane below the editor window, navigate to the **Content > Maps** folder. Then, select the map that corresponds to the scene you want to modify.

Unreal Editor Map	Vehicle Dynamics Blockset Scene
HwCurve	Curved Road
DblLnChng	Double Lane Change
BlackLake	Open Surface
LargeParkingLot	Large Parking Lot
SimpleLot	Parking Lot
HwStrght	Straight Road
USCityBlock	US City Block
USHighway	US Highway

Note The `AutoVrtlEnv.uproject` file does not include the Virtual Mcity scene.

To open a scene within your own project, in the **Content Browser** pane, navigate to the folder that contains your scenes.

Send Data to Scene

The Simulation 3D Message Get block retrieves data from the Unreal Engine 3D visualization environment. To use the block, you must configure scenes in the Unreal Engine environment to send data to the Simulink model.

For detailed information about using the block to send data to the scenes, see "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25.

Receive Data from Scene

The Simulation 3D Message Set block sends data to the Unreal Engine 3D visualization environment. To use the block, you must configure scenes in the Unreal Engine environment to receive data from the Simulink model.

For detailed information about using the block to receive data from the scene, see "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25.

Create New Scene

To create a new scene in your project, from the top-left menu of the editor, select **File > New Level**.

Alternatively, you can create a new scene from an existing one. This technique is useful if you want to use one of the prebuilt scenes in the `AutoVtrlEnv` project as a starting point for creating your own scene. To save a version of the currently opened scene to your project, from the top-left menu of the editor, select **File > Save Current As**. The new scene is saved to the same location as the existing scene.

Add Assets to Scene

In the Unreal Editor, elements within a scene are referred to as assets. To add assets to a scene, you can browse or search for them in the **Content Browser** pane at the bottom and drag them into the editor window.

When adding assets to a scene that is in the `AutoVrtlEnv` project, you can choose from a library of driving-related assets. These assets are built as static meshes and begin with the prefix `SM_`. Search for these objects in the **Content Browser** pane.

For example, to add a traffic cone to a scene in the `AutoVrtlEnv` project:

1 In the **Content Browser** pane at the bottom of the editor, navigate to the **Content** folder.

2 In the search bar, search for `SM_Cone`. Drag the cone from the **Content Browser** into the editing window. You can then change the position of the cone in the editing window or on the **Details** pane on the right, in the **Transform** section.

If you want to override the default weather or use enhanced fog conditions in the scene, add the **Exponential Height Fog** actor.

The Unreal Editor uses a left-hand *Z*-up coordinate system, where the *Y*-axis points to the right. The vehicle blocks in Vehicle Dynamics Blockset uses a right-hand *Z*-down coordinate system, where the *Y*-axis points to the right. When positioning objects in a scene, keep this coordinate system difference in mind.

For more information on modifying scenes and adding assets, see Unreal Engine 4 Documentation.

To migrate assets from the `AutoVrtlEnv` project into your own project file, see the Unreal Engine documentation.

Use AutoVrtlEnv Project Lighting in Custom Scene

To use the lighting that comes installed with the `AutoVrtlEnv` project in Vehicle Dynamics Blockset, follow these steps.

1 On the **World Settings** tab, clear **Force no precomputed lighting**.
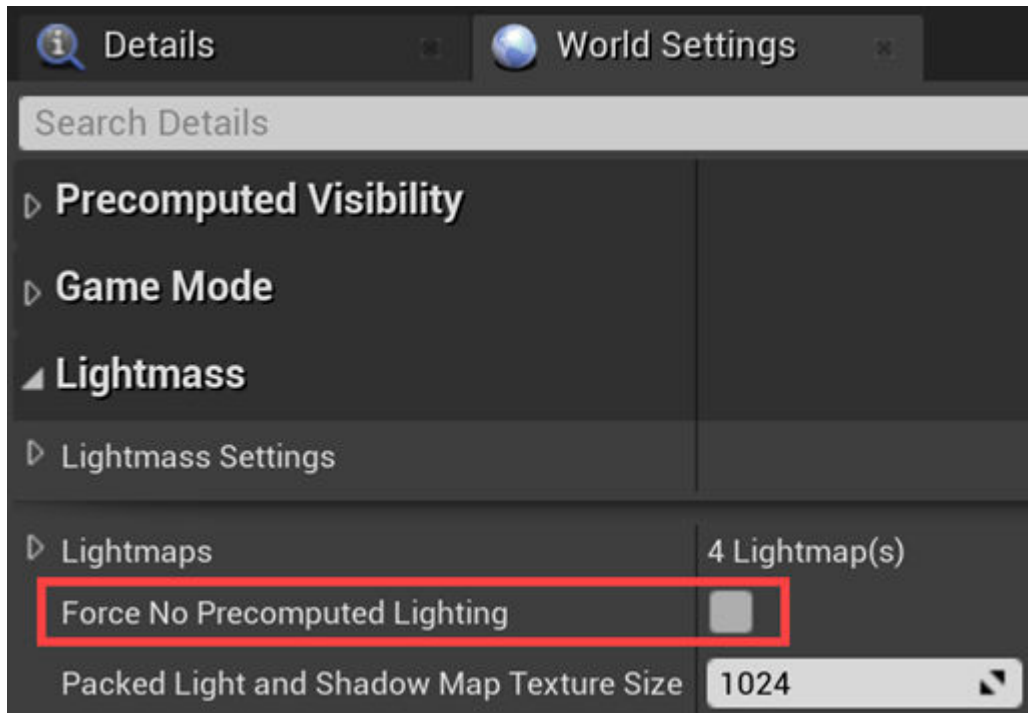
2   Under **Build**, select **Lighting Quality > Production** to rebuild the maps with production quality. Rebuilding complex maps can be time-intensive.
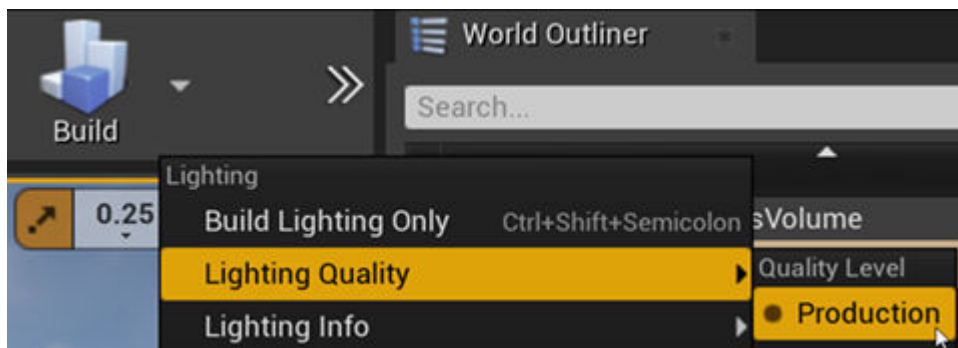
Run Simulation

Verify that the Simulink model and Unreal Editor are configured to co-simulate by running a test simulation.

1   In the Simulink model, click **Run**.

Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start. Instead, you must start the simulation from the editor.

2   Verify that the Diagnostic Viewer window in Simulink displays this message:

```
In the Simulation 3D Scene Configuration block, you set the scene source
to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the scene.
```

This message confirms that Simulink has instantiated vehicles and other assets in the Unreal Engine 3D environment.
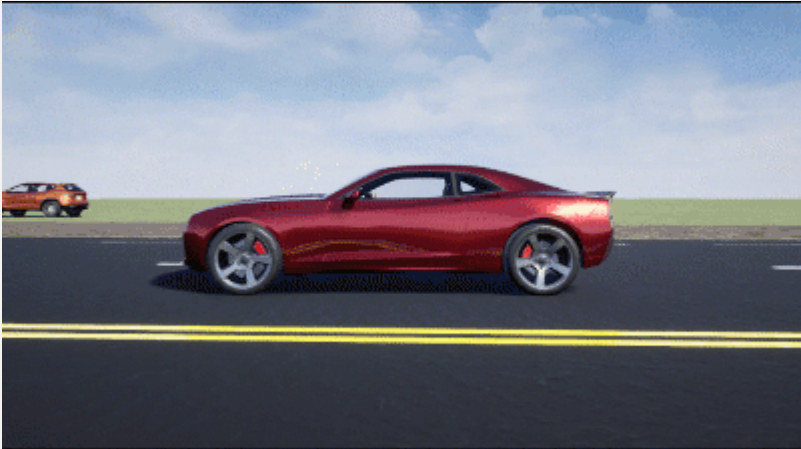
3 In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor. If your Simulink model contains vehicles, these vehicles drive through the scene that is open in the editor.

To control the view of the scene during simulation, in the Simulation 3D Scene Configuration block, select the vehicle name from the **Scene view** parameter. To change the scene view as the simulation runs, use the numeric keypad in the editor. The table shows the position of the camera displaying the scene, relative to the vehicle selected in the **Scene view** parameter.

To smoothly change the camera views, use these key commands.

Key	Camera View	
1	Back left	
2	Back	
3	Back right	
4	Left	
5	Internal	
6	Right	
7	Front left	
8	Front	
9	Front right	
0	Overhead	
		View Animated GIF

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF**
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF**

Key	Camera Control
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF**
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF**

To restart a simulation, click **Run** in the Simulink model, wait until the Diagnostic Viewer displays the confirmation message, and then click **Play** in the editor. If you click **Play** before starting the simulation in your model, the connection between Simulink and the Unreal Editor is not established, and the editor displays an empty scene.

If you are co-simulating a custom project, to enable the numeric keypad, copy the `DefaultInput.ini` file from the support package installation folder to your custom project folder. For example, copy `DefaultInput.ini` from:

`C:\ProgramData\MATLAB\SupportPackages\<MATLABRelease>\toolbox\shared\sim3dprojects\spkg\project\`

to:

`C:\<yourproject>.project\Config`

After tuning your custom scene based on simulation results, you can then package the scene into an executable. For more details, see "Package Custom Scenes into Executable" on page 6-23.

See Also

Simulation 3D Scene Configuration | `sim3d.Editor`

External Websites

- Unreal Engine
- Unreal Engine 4 Documentation

Package Custom Scenes into Executable

When you finish modifying a custom scene as described in "Customize Scenes Using Simulink and Unreal Editor" on page 6-14, you can package the project file containing this scene into an executable. You can then configure your model to simulate from this executable by using the Simulation 3D Scene Configuration block. Executable files can improve simulation performance and do not require opening the Unreal Editor to simulate your scene. Instead, the scene runs by using the Unreal Engine that comes installed with Vehicle Dynamics Blockset.

Package Scene into Executable Using Unreal Editor

Before packaging the custom scenes into an executable, make sure that the plugins are:

- Located in the Unreal Engine installation area, for example, `C:\Program Files\Epic Games \UE_4.27\Engine\Plugins\Marketplace\Mathworks`.

- Deleted from your project area, for example, `C:\project\AutoVrtlEnv\Plugins`.

Then, follow these steps.

1 Open the project containing the scene in the Unreal Editor. You must open the project from a Simulink model that is configured to co-simulate with the Unreal Editor.

   For more details on how to package projects, see "Packaging Projects" under Unreal Engine 4 Documentation.

2 Rebuild the lighting in your scenes. If you do not rebuild the lighting, the shadows from the light source in your executable file are incorrect and a warning about rebuilding the lighting displays during simulation. In the Unreal Editor toolbar, select **Build > Build Lighting Only**.

3 Close the **Project Settings** window.

4 In the top-left menu of the editor, select **File > Package Project > Windows > Windows (64-bit)**. Select a local folder in which to save the executable, such as to the root of the project file (for example, `C:/Local/myProject`).

Note Packaging a project into an executable can take several minutes. The more scenes that you include in the executable, the longer the packaging takes.

Once packaging is complete, the folder where you saved the package contains a `WindowsNoEditor` folder that includes the executable file. This file has the same name as the project file.

Note If you repackage a project into the same folder, the new executable folder overwrites the old one.

Suppose you package a scene that is from the `myProject.uproject` file and save the executable to the `C:/Local/myProject` folder. The editor creates a file named `myProject.exe` with this path:

`C:/Local/myProject/WindowsNoEditor/myProject.exe`

6-23

Simulate Scene from Executable in Simulink

To improve co-simulation performance, consider configuring the Simulation 3D Scene Configuration block to co-simulate with the project executable.

1   In the Simulation 3D Scene Configuration block of your Simulink model, set the **Scene source** parameter to `Unreal Executable`.

2   Set the **File name** parameter to the name of your Unreal Editor executable file. You can either browse for the file or specify the full path to the file by using backslashes. For example:

   `C:\Local\myProject\WindowsNoEditor\myProject.exe`

3   Set the **Scene** parameter to the name of a scene from within the executable file. For example:

   `/Game/Maps/myScene`

4   Run the simulation. The model simulates in the custom scene that you created.

If you are simulating a scene from a project that is not based on the `AutoVtrlEnv` project, then the scene simulates in full screen mode. To use the same window size as the default scenes, copy the `DefaultGameUserSettings.ini` file from the support package installation folder to your custom project folder. For example, copy `DefaultGameUserSettings.ini` from:

`C:\ProgramData\MATLAB\SupportPackages\<MATLABrelease>\toolbox\shared\sim3dprojects\automotive\Au`

to:

`C:\<yourproject>.project\Config`

Then, package scenes from the project into an executable again and retry the simulation.

See Also
Simulation 3D Scene Configuration

Get Started Communicating with the Unreal Engine Visualization Environment

You can set up communication with Unreal Engine by using the Simulation 3D Message Get and Simulation 3D Message Set blocks:

- Simulation 3D Message Get receives data from the Unreal Engine environment.
- Simulation 3D Message Set sends data to the Unreal Engine environment.

To use the blocks and communicate with Unreal Engine, make sure you install the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. For more information, see "Install Support Package and Configure Environment" on page 6-10.

Next, follow these workflow steps to set up the Simulink model and the Unreal Engine environment and run a simulation.

Workflow		Description
"Set Up Simulink Model to Send and Receive Data" on page 6-26		Configure the Simulation 3D Message Get and Simulation 3D Message Set blocks in Simulink to send and receive the cone location from Unreal Editor. The steps provides the general workflow for communicating with the editor.
		The Simulation 3D Message Get and Simulation 3D Message Set blocks can send and receive these data types: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `Boolean`. The Simulation 3D Actor Transform Set and Simulation 3D Actor Transform Get blocks can send and receive only the `single` data type.
Set Up Unreal Engine to Send and Receive Data	"C++ Workflow: Set Up Unreal Engine to Send and Receive Data" on page 6-27	Specific Unreal C++ workflow to send and receive Simulink cone location data.
		- Simulation 3D Message Get receives data from an Unreal Engine environment C++ actor class. In this example workflow, you use the block to receive the cone location from Unreal Editor.
		- Simulation 3D Message Set sends data to an Unreal Engine C++ actor class. In this example, you use the block to set the initial cone location in the Unreal Editor.
		To follow this workflow, you should be comfortable coding with C++ in Unreal Engine. Make sure that your environment meets the minimum software requirements described in "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.
	"Blueprint Workflow: Set Up Unreal Engine to Send and Receive Data" on page 6-37	Generalized Unreal Editor blueprint workflow to send and receive Simulink data.

Workflow	Description
"Run Simulation" on page 6-42	After you set up the Simulink model and Unreal Editor environment, run a simulation.

Set Up Simulink Model to Send and Receive Data

Step 1: Install Support Package

If you have already downloaded and installed Unreal Engine and the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, go to the next step.

To install and configure the support package, see "Install Support Package and Configure Environment" on page 6-10.

Before installing the support package, make sure that your environment meets the minimum software and hardware requirements described in "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

Note Make sure to launch Unreal Engine from Simulink.

Step 2: Set Up Simulink Model

Open a new Simulink model. Connect the blocks as shown.
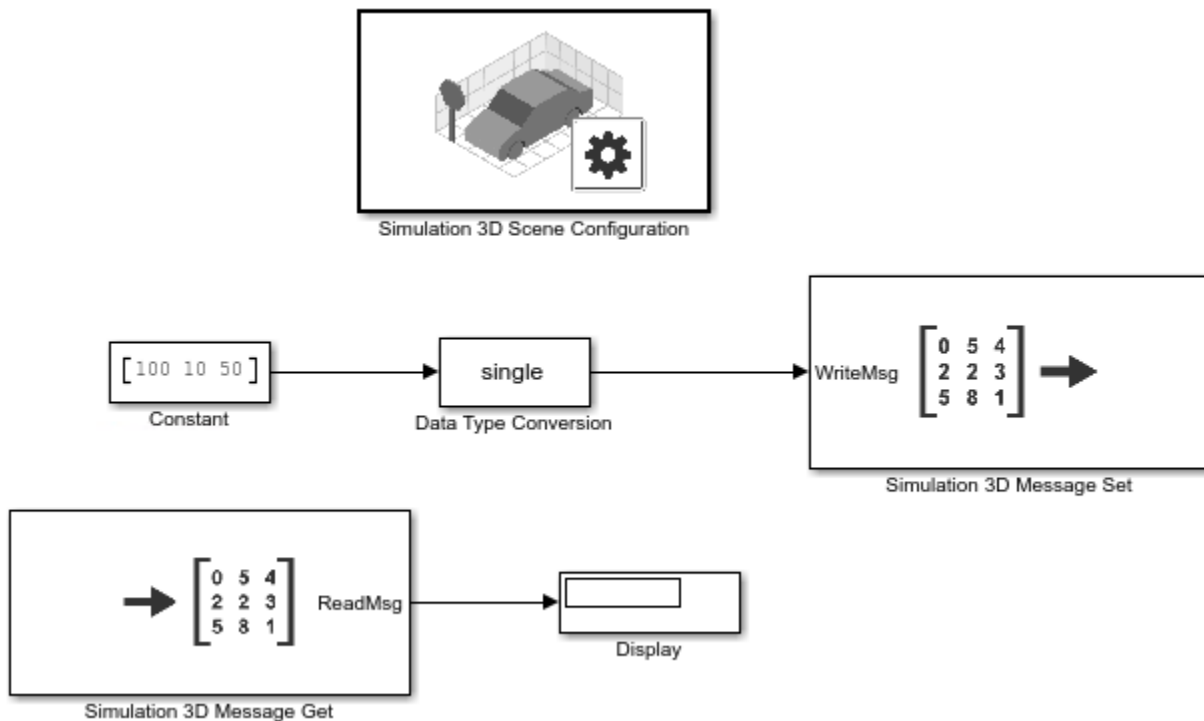
Step 3: Configure Blocks

Use these block settings to configure blocks to send and receive cone data from the Unreal Editor.

Block	Parameter Settings
Constant	• **Constant value** — `[100,10,50]` Sets the initial cone location in the Unreal Editor coordinate system (in cm, left-handed, in Z-up coordinate system) • **Interpret vector parameters as 1-D** — `off` • **Output data type** — `single`
Data Type Conversion	• **Output data type** — `single`
Simulation 3D Scene Configuration	• **Scene Source** — `Unreal Editor` • **Project** — Project path Path to project, for example the support package project.`C:\Local\AutoVrtlEnv\AutoVrtlEnv.uproject` • **Open Unreal Editor** — Select to open the editor
Simulation 3D Message Get	• **Signal name, SigName** — `ConeLocGet` • **Data type, DataType** — `single` • **Message size, MsgSize** — `[1 3]` • **Sample time** — `-1`
Simulation 3D Message Set	• **Signal name, SigName** — `ConeLocSet` • **Sample time** — `-1`

C++ Workflow: Set Up Unreal Engine to Send and Receive Data

Step 4: Open Unreal Editor in Editor Mode

1 In your model, open the Simulation 3D Scene Configuration block. Select **Open Unreal Editor**.

2 Create an Unreal Engine C++ project. Name it `TestSim3dGetSet`. For steps on creating C++ projects, see "Create Empty Project in Unreal Engine" on page 6-49.

3 In the Unreal Editor, on the **Edit** tab, select **Plugins**. Make sure that the `MathWorks Interface` plugin is enabled. If it is disabled, enable it.

4 Close the Unreal Editor.

5 If Visual Studio® is not open, open it.

6 In Visual Studio, add the MathWorksSimulation dependency to the `TestSim3dGetSet` project build file.

- The project build file, `TestSim3dGetSet.Build.cs`, is located in this folder: `...\TestSim3dGetSet\Source\TestSim3dGetSet`.

- In the build file, edit line 11 to add the `MathWorksSimulation` dependency:

```
PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
"Engine", "InputCore", "MathWorksSimulation"});
```

7 Save the change and close the `TestSim3dGetSet` project.

8 Close Visual Studio.

9 In your model, open the Simulation 3D Scene Configuration block.

   a Set **Project** to *Your_Project_path*\TestSim3dGetSet.uproject.

   b Select **Open Unreal Editor**.

Step 5: Create Actor Class

1 In the Unreal Editor, on the **Content Browser** tab, under **View Options**, select **Show Engine Content** and **Show Plugin Content**.
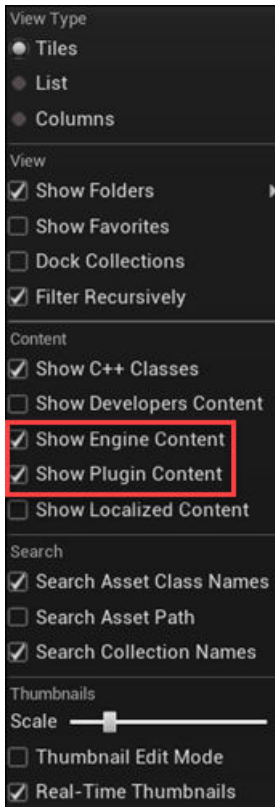
2 In the Unreal Editor, from the MathWorksSimulation C++ classes directory, select **Sim3dActor**.
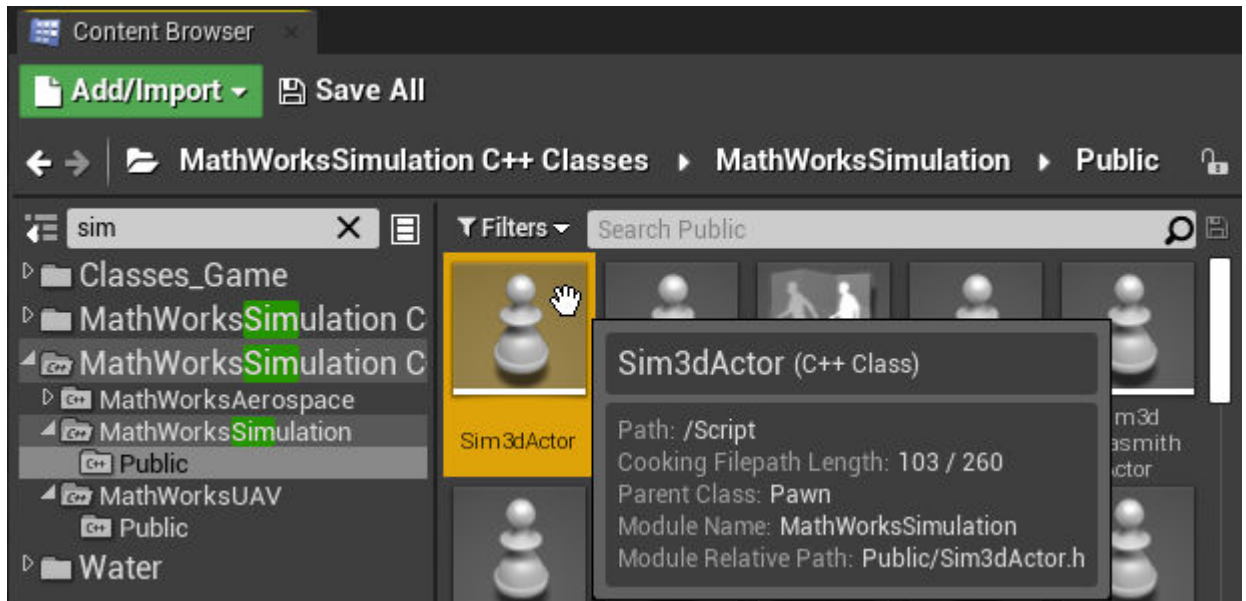
Right-click and select **Create C++ class derived from Sim3dActor**.
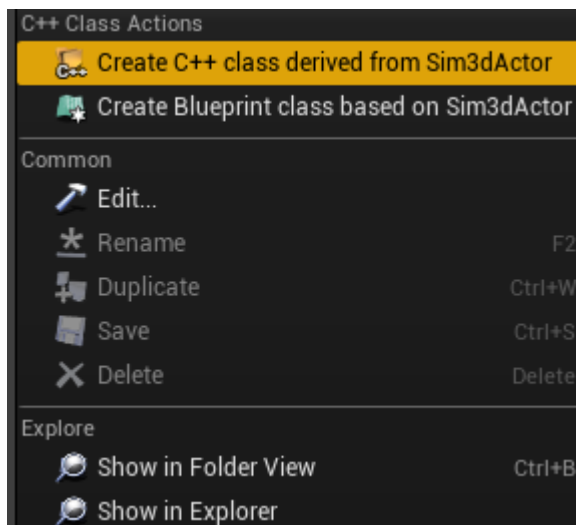
3   Name the new Sim3dActor `SetGetActorLocation`. Select **Public**. Click **Create Class**.

4   Close the Unreal Editor.

Step 6: Open SetGetActorLocation.h

Visual Studio opens with new C++ files in the project folder:

- `SetGetActorLocation.h`
- `SetGetActorLocation.cpp`

Make sure you close the Unreal Editor.

In Visual Studio, build the solution `TestSim3dGetSet`:

1. In the Solution Explorer, right-click **Solution 'TestSim3dGetSet' (2 projects)**.
2. Select **Build Solution**.
3. After the solution builds, open `SetGetActorLocation.h`. Edit the file as shown.

Replacement Code: `SetGetActorLocation.h`

This is the replacement code for `SetGetActorLocation.h`.

```
// Copyright 2019-2022 The MathWorks, Inc.

#pragma once

#include "CoreMinimal.h"
#include "Sim3dActor.h"
#include "SetGetActorLocation.generated.h"

UCLASS()
class TESTSIM3DGETSET_API ASetGetActorLocation : public ASim3dActor
{
    GENERATED_BODY()

    void *SignalReader;
    void *SignalWriter;

public:
    // Sets default values for this actor's properties
    ASetGetActorLocation();

    virtual void Sim3dSetup() override;
    virtual void Sim3dRelease() override;
    virtual void Sim3dStep(float DeltaSeconds) override;
};
```

Step 7: Open SetGetActorLocation.cpp

Open `SetGetActorLocation.cpp` and replace the block of code.

Replacement Code: Set Pointer to Parameter

This code allows you to set a pointer to the parameter Signal Name parameter for the Simulink blocks Simulation 3D Message Set and Simulation 3D Message Get, respectively.

```
// Sets default values
ASetGetActorLocation::ASetGetActorLocation():SignalReader(nullptr), SignalWriter(nullptr)
{
}
```

Replacement Code: Access Actor Tag Name

The following code allows you to access the tag name of this actor after it is instantiated in the scene with an assigned tag name. The code also initializes the pointers `SignalReader` and `SignalWriter`, to initiate a link between Unreal Editor and Simulink. The variables represent these block Signal Name parameter values:

- `SignalReaderTag` — Simulation 3D Message Set
- `SignalWriterTag` — Simulation 3D Message Get

```
void ASetGetActorLocation::Sim3dSetup()
{
Super::Sim3dSetup();
    if (Tags.Num() != 0) {
            unsigned int numElements = 3;
            FString tagName = Tags.Top().ToString();
```

```
                    FString SignalReaderTag = tagName;
                    SignalReaderTag.Append(TEXT("Set"));
                    SignalReader = StartSimulation3DMessageReader(TCHAR_TO_ANSI(*SignalReaderTag), sizeof(float)*numElements);

                    FString SignalWriterTag = tagName;
                    SignalWriterTag.Append(TEXT("Get"));
                    SignalWriter = StartSimulation3DMessageWriter(TCHAR_TO_ANSI(*SignalWriterTag), sizeof(float)*numElements);
                    }
}
```

Additional Code: Read and Write Data During Run Time

Add this code to allow Unreal Engine to read the data value set by Simulation 3D Message Set and then write back to Simulation 3D Message Get during run time. Unreal Engine uses this data to set the location value of the actor.

```
void ASetGetActorLocation::Sim3dStep(float DeltaSeconds)
{
        unsigned int numElements = 3;
        float array[3];
        int statusR = ReadSimulation3DMessage(SignalReader, sizeof(float)*numElements, array);
        FVector NewLocation;
        NewLocation.X = array[0];
        NewLocation.Y = array[1];
        NewLocation.Z = array[2];
        SetActorLocation(NewLocation);
        float fvector[3] = { NewLocation.X, NewLocation.Y, NewLocation.Z };
        int statusW = WriteSimulation3DMessage(SignalWriter, sizeof(float)*numElements ,fvector);
}
```

Additional Code: Stop Simulation

Add this code so that Unreal Engine stops when you press the Simulink stop button. The code destroys the pointer SignalReader and SignalWriter.

```
void ASetGetActorLocation::Sim3dRelease()
{
        Super::Sim3dRelease();
        if (SignalReader) {
                StopSimulation3DMessageReader(SignalReader);
        }
        SignalReader = nullptr;

        if (SignalWriter) {
                StopSimulation3DMessageWriter(SignalWriter);
        }
        SignalWriter = nullptr;
}
```

Entire Replacement Code: SetGetActorLocation.cpp

This is the entire replacement code for SetGetActorLocation.cpp.

```
// Copyright 2019-2022 The MathWorks, Inc.
#include "SetGetActorLocation.h"

// Sets default values
ASetGetActorLocation::ASetGetActorLocation():SignalReader(nullptr), SignalWriter(nullptr)
{
}

void ASetGetActorLocation::Sim3dSetup()
{
Super::Sim3dSetup();
        if (Tags.Num() != 0) {
                unsigned int numElements = 3;
                FString tagName = Tags.Top().ToString();
```

6-31

```
            FString SignalReaderTag = tagName;
            SignalReaderTag.Append(TEXT("Set"));
            SignalReader = StartSimulation3DMessageReader(TCHAR_TO_ANSI(*SignalReaderTag), sizeof(float)*numElements);

            FString SignalWriterTag = tagName;
            SignalWriterTag.Append(TEXT("Get"));
            SignalWriter = StartSimulation3DMessageWriter(TCHAR_TO_ANSI(*SignalWriterTag), sizeof(float)*numElements);
            }
}

void ASetGetActorLocation::Sim3dStep(float DeltaSeconds)
{
      unsigned int numElements = 3;
      float array[3];
      int statusR = ReadSimulation3DMessage(SignalReader, sizeof(float)*numElements, array);
      FVector NewLocation;
      NewLocation.X = array[0];
      NewLocation.Y = array[1];
      NewLocation.Z = array[2];
      SetActorLocation(NewLocation);
      float fvector[3] = { NewLocation.X, NewLocation.Y, NewLocation.Z };
      int statusW = WriteSimulation3DMessage(SignalWriter, sizeof(float)*numElements ,fvector);
}

void ASetGetActorLocation::Sim3dRelease()
{
      Super::Sim3dRelease();
      if (SignalReader) {
            StopSimulation3DMessageReader(SignalReader);
      }
      SignalReader = nullptr;

      if (SignalWriter) {
            StopSimulation3DMessageWriter(SignalWriter);
      }
      SignalWriter = nullptr;
}
```

Step 8: Build the Visual Studio Project and Open Unreal Editor

In Visual Studio, select **Debug > Start Debugging** or press **F5** to run the `TestSim3dGetSet` solution. The Unreal Editor opens.

Note In the Unreal Editor, save the current level by clicking **Save Current** (located in the top left) and name it `TestMap`. Add this level as default to Project Settings by clicking on **Edit > Project Settings > Maps&Modes**. Then select `TestMap` as the default value for the Editor Startup Map and Game Default Map. Close Project Settings to save the default values.
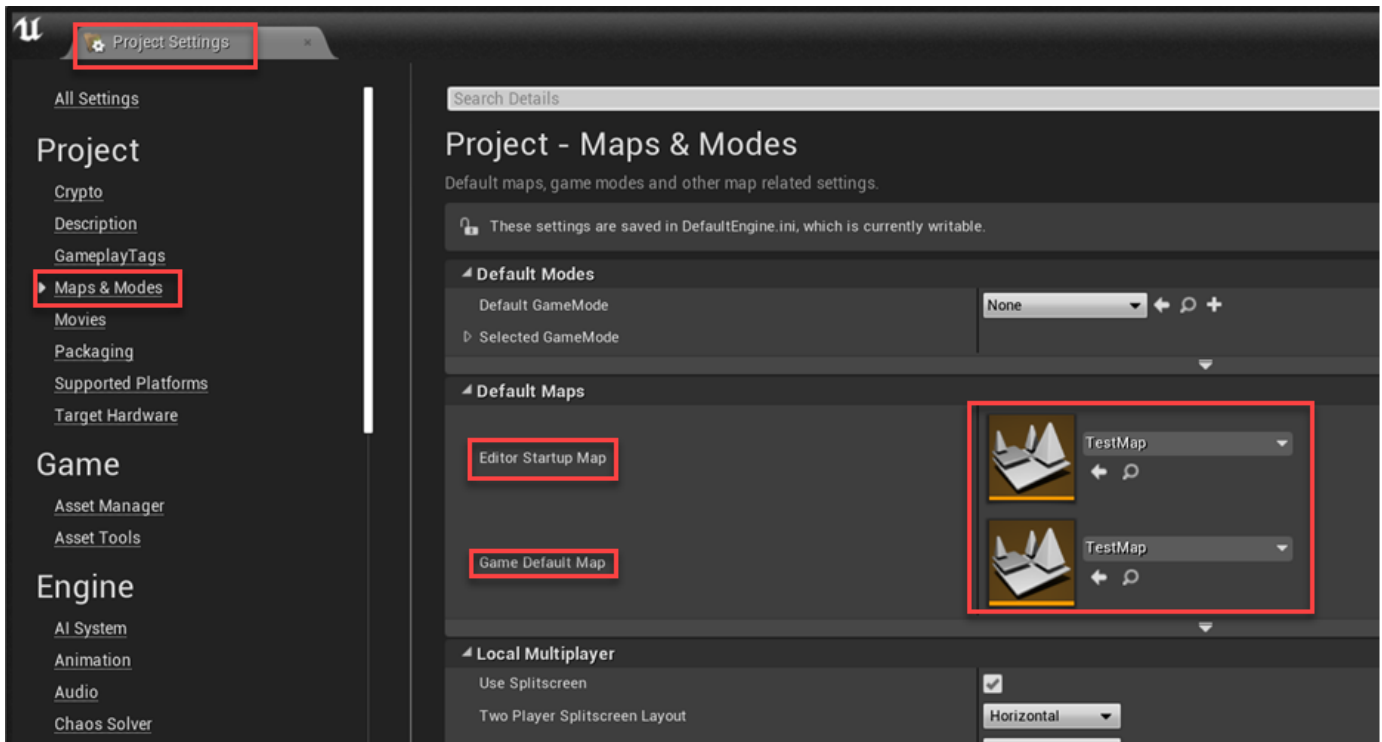
Step 9: Place and Check Actor

1 In the Unreal Editor, find the Set Get Actor Location and place it in the `TestMap`.
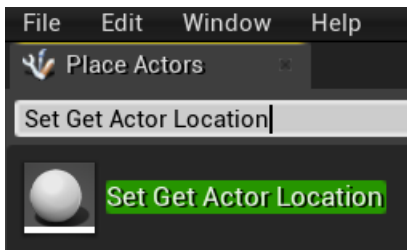
2 On the **World Outliner** tab, check that the new instantiated actor, `SetGetActorLocation1`, is listed.
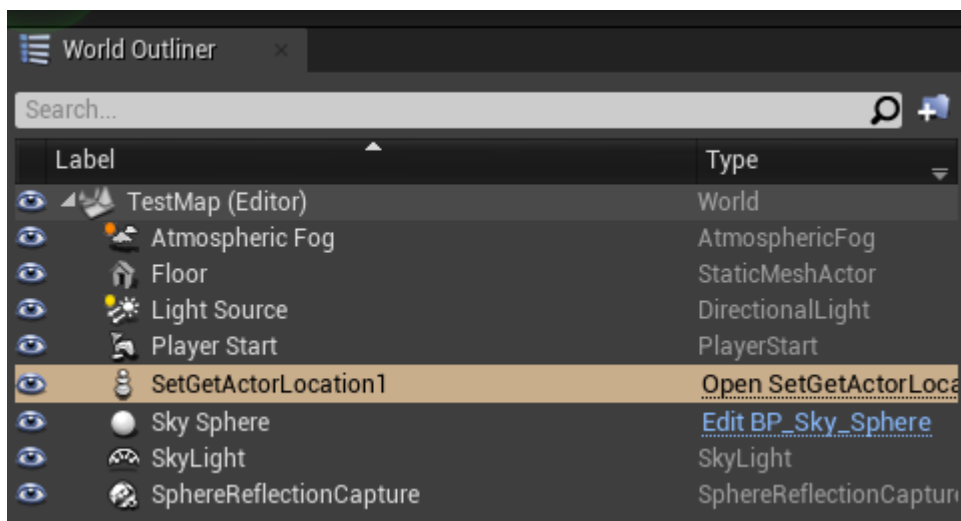
Step 10: Add Mesh

Click on the actor that you created in "Step 9: Place and Check Actor" on page 6-33.

1   In the **Details** panel, click on Add Component to add a mesh to the actor
     `SetGetActorLocation1`. Choose `Cone` as the default mesh.

2   Find the property tags for actor `SetGetActorLocation1`. Add a tag by clicking on the plus sign
     next to 0 Array elements. Name it `ConeLoc`.
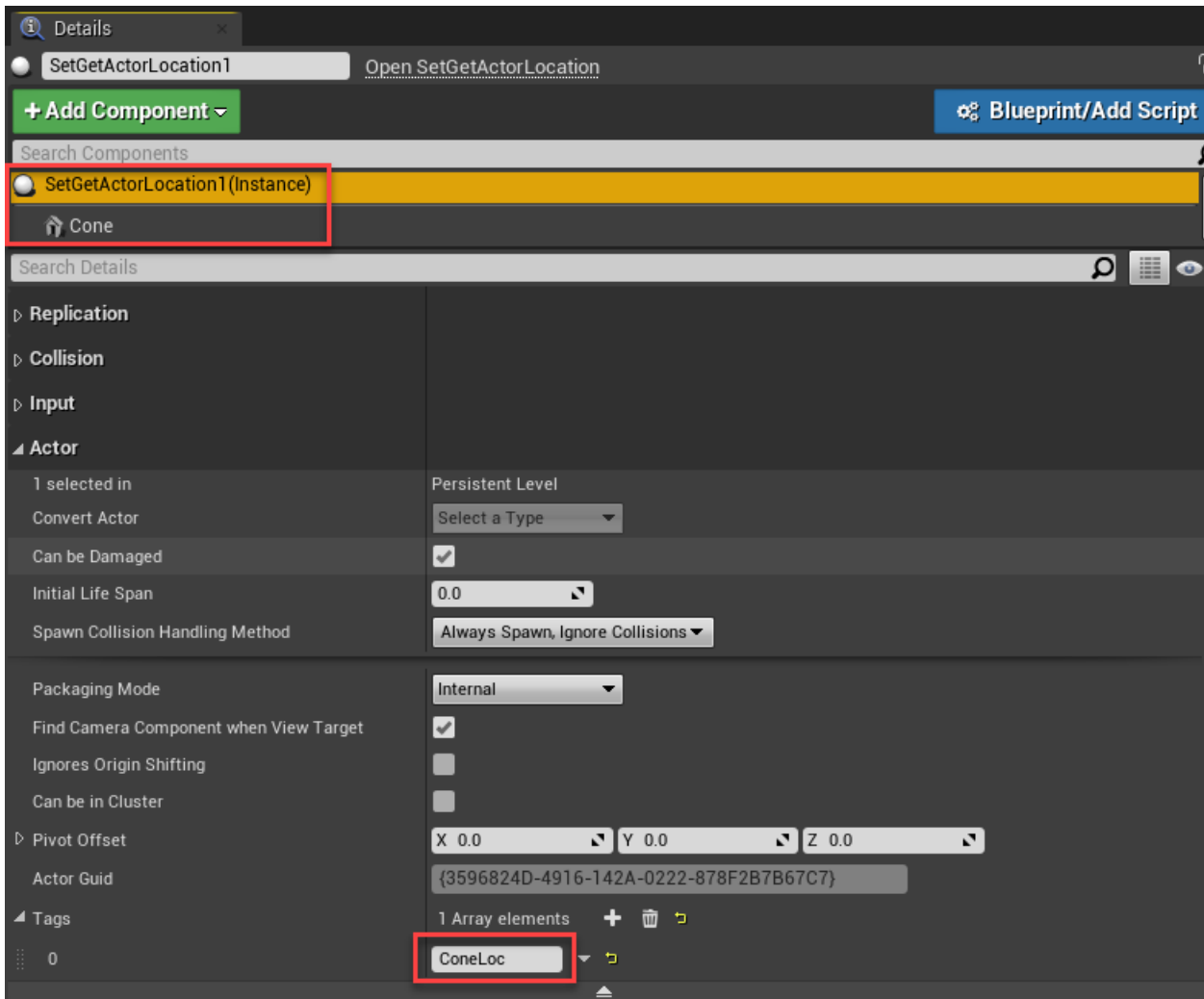
Step 11: Set Cone Location

On the **Details** tab, click **Cone**. Set the cone to X = 0.0, Y = 0.0, and Z = 0.0. Also set the actor **Mobility** property to Movable.
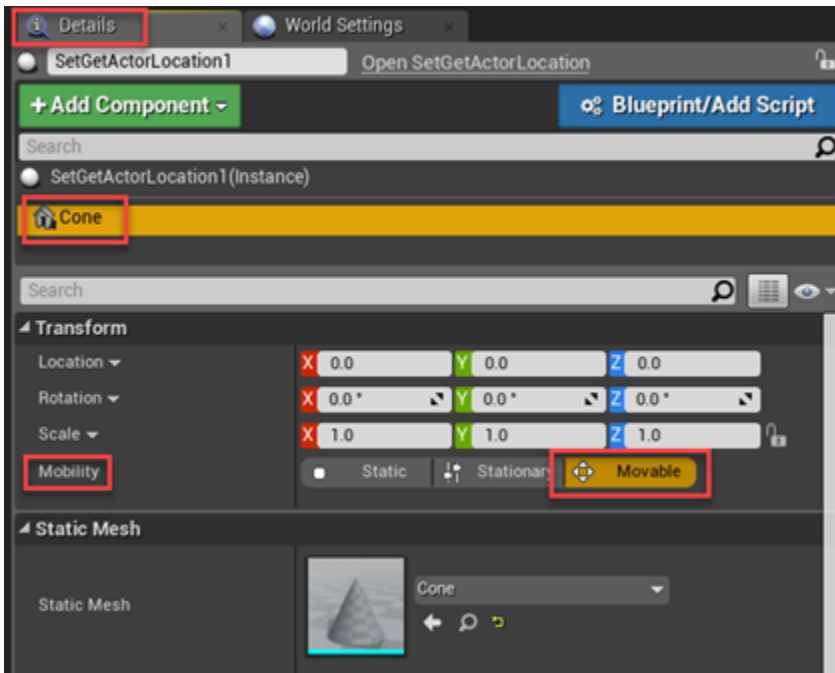
Step 12: Set Parent Class and Save Scene

Set the parent class.

1 Under **Blueprints**, click **Open Level Blueprint**, and select **Class Settings**.
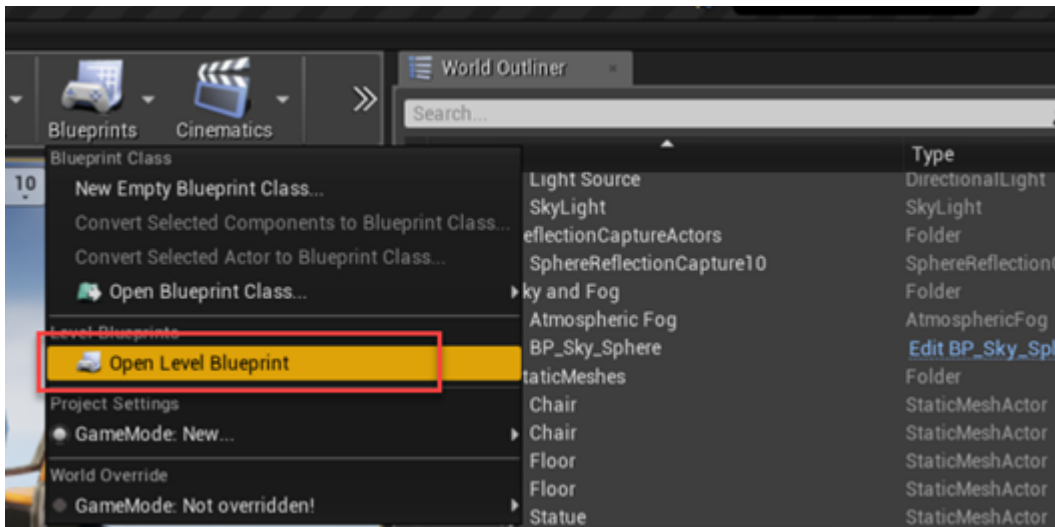
2 In the **Class Options**, set **Parent Class** to `Sim 3d Level Script Actor`.

Save the Unreal Editor scene.

Step 13: Run Simulation

Run the simulation. Go to "Run Simulation" on page 6-42.

Reference: C++ Functions for Sending and Receiving Simulink Data

Call these C++ functions from `Sim3dSetup`, `Sim3dStep`, and `Sim3dRelease` to send and receive Simulink data.

To	C++ Functions
Receive data	`StartSimulation3DMessageReader`
	`ReadSimulation3DMessage`
	`StopSimulation3DMessageReader`
Send data	`StartSimulation3DMessageWriter`
	`WriteSimulation3DMessage`
	`StopSimulation3DMessageWriter`

Blueprint Workflow: Set Up Unreal Engine to Send and Receive Data

Step 4: Configure Scenes to Receive Data

To use the Simulation 3D Message Set block, you must configure scenes in the Unreal Engine environment to receive data from the Simulink model:

1   In the Unreal Editor, instantiate the `Sim3DGet` actor that corresponds to the data type you want to receive from the Simulink model. This example shows the Unreal Editor `Sim3DGet` data types.
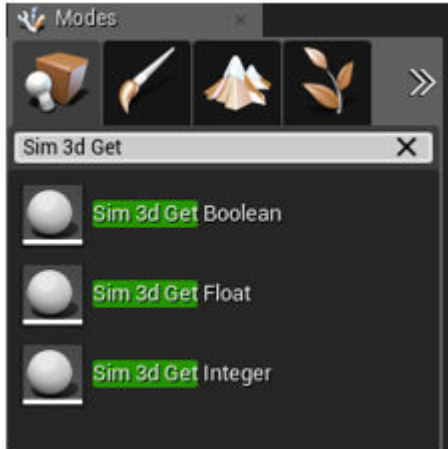
2 Specify an actor tag name that matches the Simulation 3D Message Set block **Signal name** parameter.

3 Navigate to the Level Blueprint.

4 Find the blueprint method for the `Sim3DGet` actor class based on the data type and size that you want to receive from the Simulink model.

For example, in Unreal Editor, this diagram shows that `Read Scalar Integer` is the method for `Sim3DGetInteger` actor class to receive `int32` data type of size scalar.
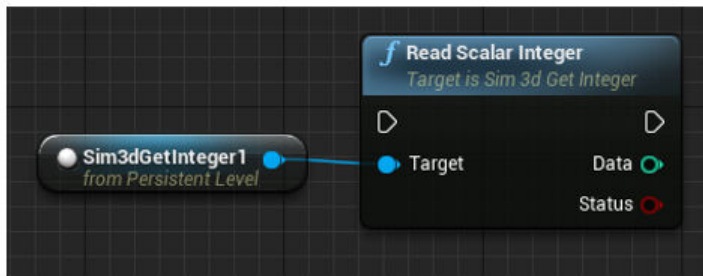
5 Compile and save the scene.

Step 5: Configure Scenes to Send Data

To configure scenes in the Unreal Engine environment to send data to the Simulink model:

1 In the Unreal Editor, instantiate the `Sim3DSet` actor that corresponds to the data type you want to send to the Simulink model. This example shows the Unreal Editor `Sim3DSet` data types.
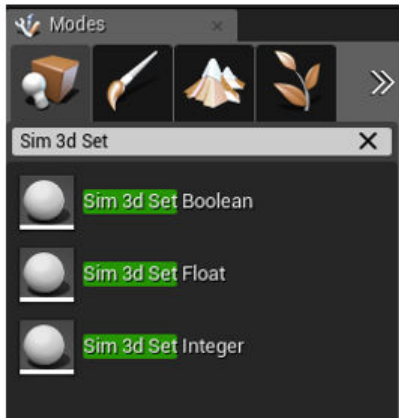
2 Specify an actor tag name that matches the Simulation 3D Message Get block **Signal name** parameter.

3 Navigate to the Level Blueprint.

4 Find the blueprint method for the `Sim3DSet` actor class based on the data type and size specified by the Simulation 3D Message Get block **Data type** and **Message size** parameters.

For this example, the array size is 3. The Unreal Editor diagram shows that `Write Array Float` is the method for the `Sim3DSetFloat3` actor class that sends float data type of array size 3.

5 Compile and save the scene.

Note Optionally, for better performance, set **Read Array Float Max Num Elements** to `Num El` in the Actor Blueprint.
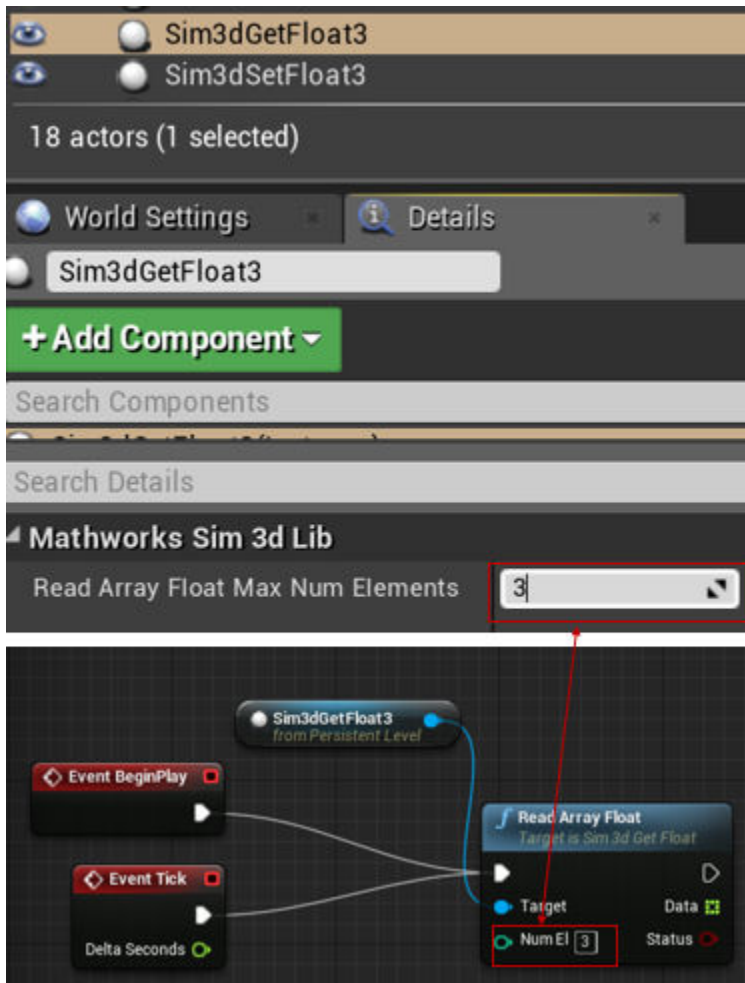
Step 6: Create Blueprint

In the Unreal Editor, create a level blueprint connecting the Get and Set actors.

1. Set the actor tag values.

   - `Sim3dGetFloat1` — Simulation 3D Message Set block **Signal name, SigName** parameter value, for example `ConeLocSet`
   - `Sim3dSetFloat1` — Simulation 3D Message Get block **Signal name, SigName** parameter value, for example `ConeLocGet`

2. Set the parent class.

   a. Under **Blueprints**, click **Open Level Blueprint**, and select **Class Settings**.
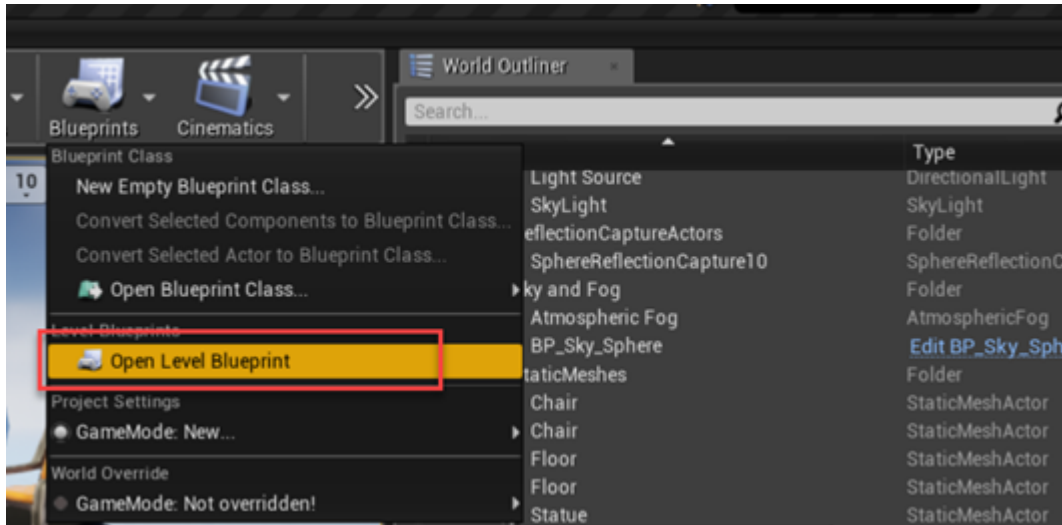
b In the **Class Options**, set **Parent Class** to `Sim 3d Level Script Actor`.
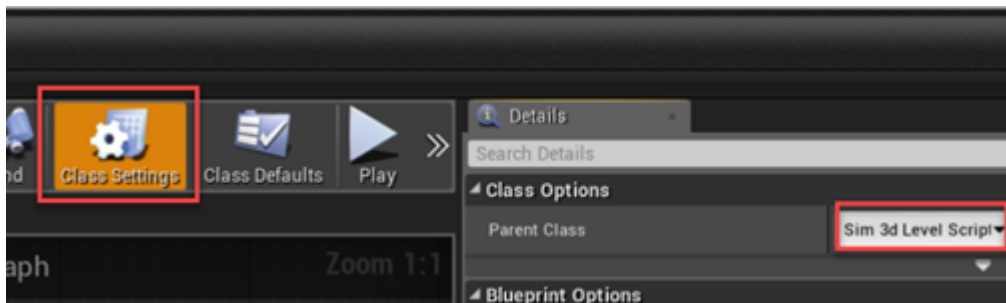
3 In the level blueprint, make the connections, for example:
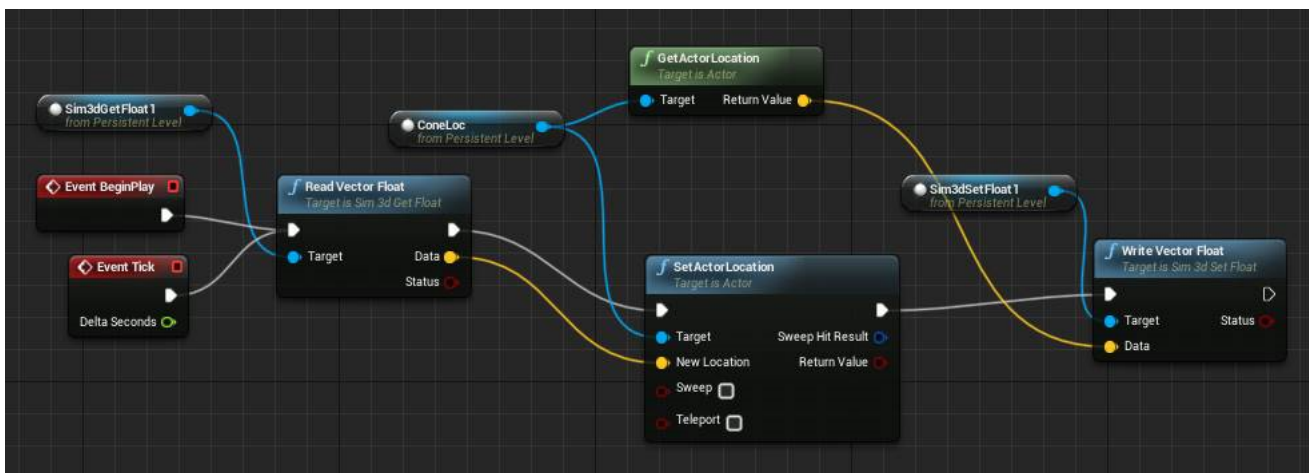
Step 7: Run Simulation

Run the simulation. Go to "Run Simulation" on page 6-42.

Run Simulation

After you configure the Simulink model and Unreal Editor environment, you can run the simulation.

Note At the BeginPlay event, Simulink does not receive data from the Unreal Editor. Simulink receives data at Tick events.

Run the simulation.

1 In the Simulink model, click **Run**.

Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start.

2 Verify that the Diagnostic Viewer window in Simulink displays this message:

```
In the Simulation 3D Scene Configuration block, you set the scene source
to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the scene.
```

This message confirms that Simulink has instantiated the vehicles and other assets in the Unreal Engine 3D environment.

3 In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor.

You can send and receive these data types: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `boolean`. The code in "Step 7: Open SetGetActorLocation.cpp" on page 6-30 reads single data type values (or float values) from Simulink.

See Also

`ASim3dActor` | `Sim3dSetup` | `Sim3dStep` | `Sim3dRelease` | Simulation 3D Scene Configuration | Simulation 3D Message Get | Simulation 3D Message Set

More About

- "Animate Custom Actors in the Unreal Editor" on page 8-21
- "Place Cameras on Actors in the Unreal Editor" on page 8-10
- "Send and Receive Double-Lane Change Scene Data" on page 3-93

External Websites

- Unreal Engine 4 Documentation

Create and Use an Oval Track

You can create a oval track with RoadRunner and use it in a Vehicle Dynamics Blockset simulation that co-simulates with Unreal. This example provides the workflow for creating the oval track that is used in the "Follow Waypoints Around Oval Track" on page 7-37 example.

Before you start, make sure that you have the products required to follow the workflow.

Step		Required Products
1	"Step 1: Create Track in RoadRunner" on page 6-43	RoadRunner
2	"Step 2: Export Track From RoadRunner" on page 6-45	RoadRunner
3	"Step 3: Import Track to Unreal Engine" on page 6-45	Unreal Engine 4.27 RoadRunner plugin Visual Studio 2019
4	"Step 4: Co-Simulate in Vehicle Dynamics Blockset" on page 6-47	Unreal Engine 4.27 Vehicle Dynamics Blockset Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects

Step 1: Create Track in RoadRunner

In this example, you create the oval track specified in the following figure. The locations and reference poses are in the RoadRunner coordinate system, (X, Y, θ). The locations, X and Y, are in m. The reference poses, θ, are in deg.
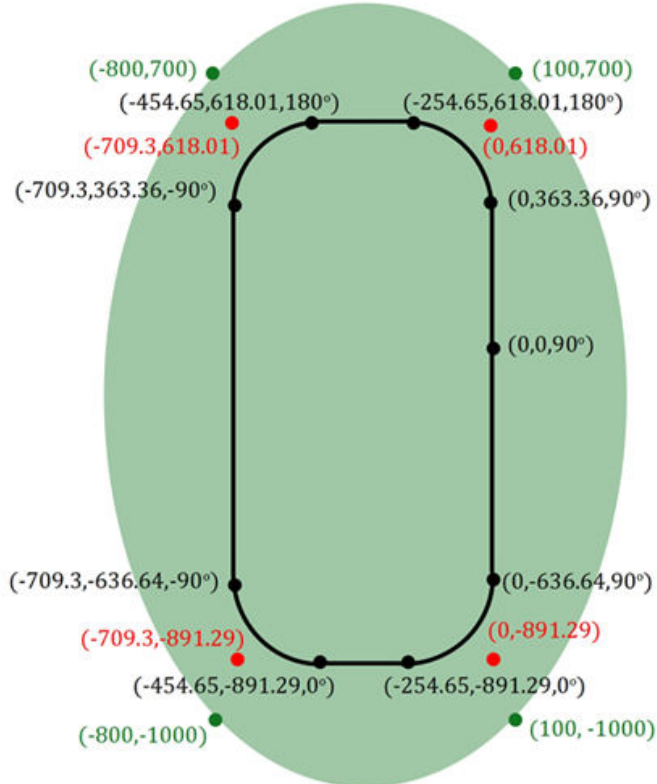
Legend	
Green	Locations of patch boundaries, *X*, *Y*
Red	Intersection points of straight-line segments, *X*, *Y*
Black	Track locations and reference poses, *X*, *Y*, θ

Use RoadRunner to create the oval track. For more information about creating tracks, see

Create Straight Line Segments

1   Open the **Road Plan Tool**.
2   Navigate to **Library Browser** > **RoadStyles**. Select `Residential`.
3   Right-click to place the start and endpoints of a straight-line road.
4   After each straight segment, left-click and repeat the preceding step.
5   Selecting the road control points. Use the preceding figure to the enter coordinate information.

Create Circular Arc

1   Open the **Road Plan Tool**. If you are continuing from creating the straight-line segments, the `Residential` road is the default road.
2   Right-click to place the start and endpoints of the arc. If you use this method to connect straight lines, there might be multiple control points. Delete all but one control point.
3   To create a control point, right-click anywhere in the road. To position it as specified in red in the preceding figure, left-click in each arc control point.

4   To create the circular arc, edit the positions of the control points. If the start of a road marks the end of another, RoadRunner connects them.

Adjust Road Width

1   Open the **Lane Width Tool**.

2   Select the road that needs the width adjustment.

3   To adjust the road width, select any purple or red segment.

4   Modify the lines so that the entire width of the road is 14 m, or 7 m from the road centerline to each edge.

Create Patch

1   Open the **Surface Tool**.

2   Right-click to create four nodes.

3   Right-click the first node to close the loop. By default, the surface is green.

Add Trees

To provide visual cues that indicate how fast the vehicle is traveling, you can add trees.

1   Open the **Prop Curve Tool**.

2   Select an asset to place in the scene. For example, select `Props\Trees\Eucalyptus_Sm01`.

3   Right-click to select the inner boundary of the trajectory. To limit the trees in the green areas, adjust the tangents at the control points.

4   To facilitate faster import into Unreal Engine, choose an appropriate spacing that limits the number of trees in the scene. The import time is proportional to the number of scene assets.

Step 2: Export Track From RoadRunner

1   In RoadRunner, open the scene.

2   Select **File > Export > Unreal (.fbx, .xodr + rrdata.xml)**

3   In the Export Unreal dialog box, select **Split by Segmentation** and export folder. Click **Export**.

Step 3: Import Track to Unreal Engine

After exporting from RoadRunner, you import the data into Unreal Engine.

Create Empty Project
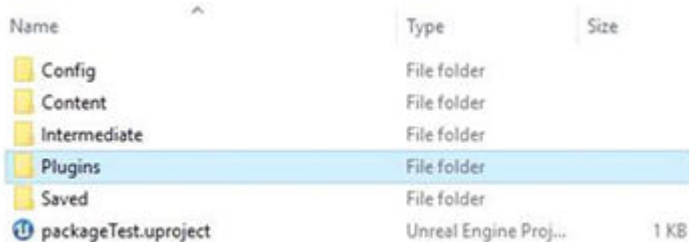
Create an empty project by following the steps listed in "Create Empty Project in Unreal Engine" on page 6-49.

Acquire and Rebuild RoadRunner Plugins

1   Download the RoadRunner plugin. For more information, see "Downloading Plugins" (RoadRunner).

2   Extract the RoadRunner plugin `.zip` file. Locate the `RoadRunnerImporter` and `RoadRunnerMaterials` folders under the Unreal Engine plugins.

Note The Unreal Engine plugin folder also contains a `RoadRunnerCarla` integration plugin. If you are not using CARLA, do not copy this folder.

3 Copy the `RoadRunnerImporter` and `RoadRunnerMaterials` folders into the **Plugins** folder under the project folder. If a Plugins folder does not exist, create one.

Name ^	Type	Size
📁 Config	File folder	
📁 Content	File folder	
📁 Intermediate	File folder	
📁 Plugins	File folder	
📁 Saved	File folder	
ⓘ packageTest.uproject	Unreal Engine Proj...	1 KB

4 Use a or b to rebuild the plugin.

a Generate the project files.

- Windows® – Right-click the `.uproject` file and select **Generate Visual Studio project files**.
- Linux® – Set environment variable `UE4_ROOT` to your Unreal Engine installation folder. At the command line, run this code:

  `$UE4_ROOT/GenerateProjectFiles.sh -project="<Path to .uproject file>" -game -engine`

b Open the project. Select **Yes** to build the plugins.

If both a and b fail, try using Visual Studio to build the binaries.

5 Verify that the RoadRunner and MathWorks Interface plugins are enabled. Select **Edit > Plugins**. Confirm that **Enabled** is selected.

Import to Unreal Engine

1 In the Unreal Editor, click **Import**. Select the `.fbx` file from Step 2.

Note Selecting **File > Import Into Level** does not use the exported RoadRunner `xml`. Instead, it uses the Unreal importer.

2 Use the default options in the RoadRunner Import Options Dialog Box. Click **Import**.

3 Under the **Scene** tab, select **Import as Dynamic**. This enables translation of the whole scene.

4 Under **Static Meshes**, clear **Remove Degenerates**. Set **Normal Input Method** as `Input Normals`. Click **Import**. The import can take up to 1 hour to complete.

5 In the **World Outliner**, select the scene that you imported, for example `FbxScene_OvalTrack1`. To align the RoadRunner and Unreal coordinate systems, enter a 90° rotation about the *Z*-axis.
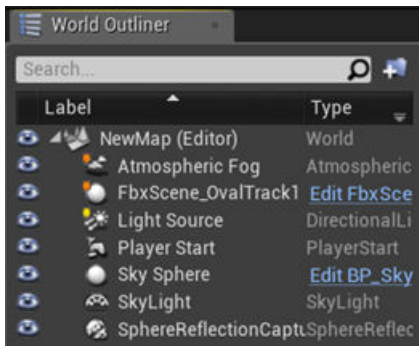
6 Optionally, consider using the editor to add terrain and foliage in the scene.

7 Save the project (`.uproject`) file. Close the Unreal Editor.

Step 4: Co-Simulate in Vehicle Dynamics Blockset

1 Open the Simulink model. Do not open the Unreal Editor.

2 Open the Simulation 3D Scene Configuration block.

    a Set **Scene source** to `Unreal Editor`.

    b Set **Project** to the project (`.uproject`) file that you saved in "Step 3: Import Track to Unreal Engine" on page 6-45.

    c Click **Apply**.

    d Click **Open Unreal Editor**.

The project opens in the Unreal Editor.

3 Select **Blueprints > Open Level Blueprint**.

4 In the level blueprint:

    a Select **File > Reparent Blueprint**.

    b Select **Sim3dLevelScriptActor**.

    c Click **Save**.

    d Close the level blueprint.

    e In the editor, click **Save Current**.

This ensures that the vehicle identifies the ground properly during co-simulation.

5 Run the simulation.

    a In the Simulink model, click **Run**.

Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start.

    b Verify that the Diagnostic Viewer window in Simulink displays this message:

```
In the Simulation 3D Scene Configuration block, you set the scene
source to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the
scene.
```

This message confirms that Simulink has instantiated the vehicles and other assets in the Unreal Engine 3D environment.

c In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor.

See Also
Simulation 3D Scene Configuration

Related Examples
- "Follow Waypoints Around Oval Track" on page 7-37

More About
- "Integrate Scenes with MATLAB and Simulink" (RoadRunner)

Create Empty Project in Unreal Engine

If you do not have an existing Unreal Engine project, you can create an empty project by following these steps.

1 In Unreal Engine, select **File** > **New Project**.

2 Create a project. For example, select the **Games** template category. Click **Next**.
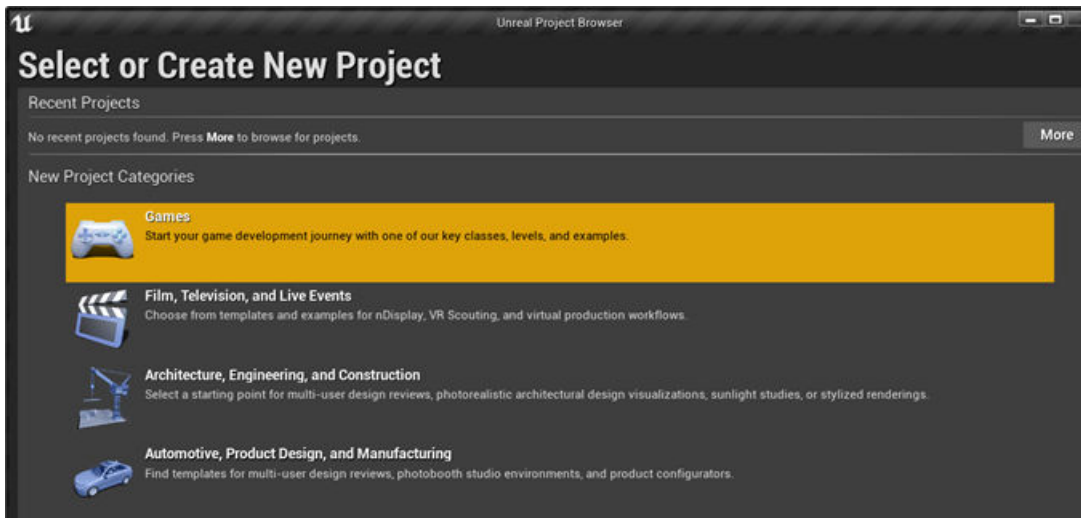
3 Select a **Blank** template. Click **Next**.
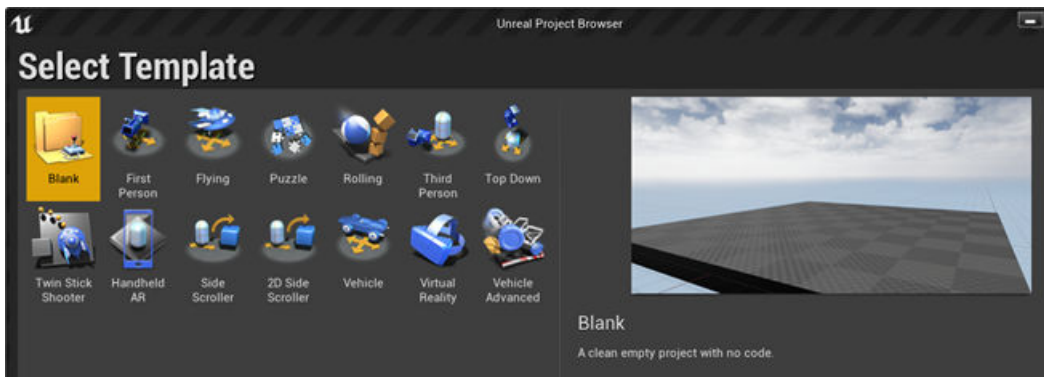
4 In **Project Settings**, create a Blueprint or C++ project, and select a project name and location. Click **Create Project**.
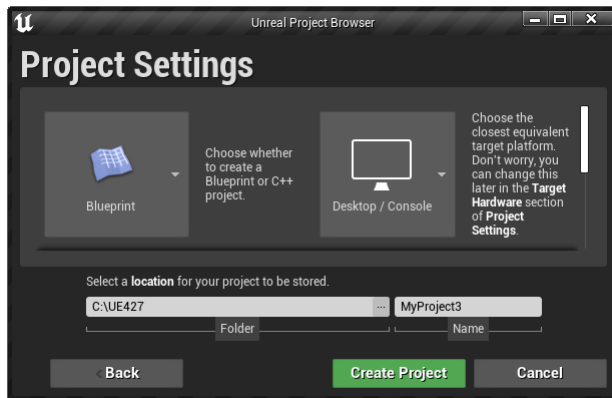
The Epic Games Launcher creates a new project and opens the Unreal Editor.

5   Enable the MathWorks Interface plugin.

    a   Select **Edit** > **Plugins**.

    b   On the **Plugins** tab, navigate to MathWorks Interface. Select **Enabled**.

6   Save the project. Close the Unreal Editor.

7   Launch Simulink. In a Simulation 3D Scene Configuration block, set:

    •   **Scene Source** to `Unreal Editor`

    •   **Project** to the project created in step 6.

Select **Open Unreal Editor** to open the editor.

See Also
Simulation 3D Scene Configuration

More About
•   "Animate Custom Actors in the Unreal Editor" on page 8-21
•   "Build Light in Unreal Editor" on page 6-6
•   "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

External Websites

- Unreal Engine

Vehicle Dynamics Blockset Examples

Scene Interrogation with Camera and Ray Tracing Reference Application

Interrogate a 3D scene with a vehicle dynamics model by using a camera and ray tracing reference application project.

To create or modify other scenes, you need the Vehicle Dynamics Blockset Interface for Unreal Engine® 4 Projects support package. For more information, see "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8.

For the minimum hardware required to run the example, see "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6.

For more information about the reference application, see "Scene Interrogation in 3D Environment" on page 3-34.
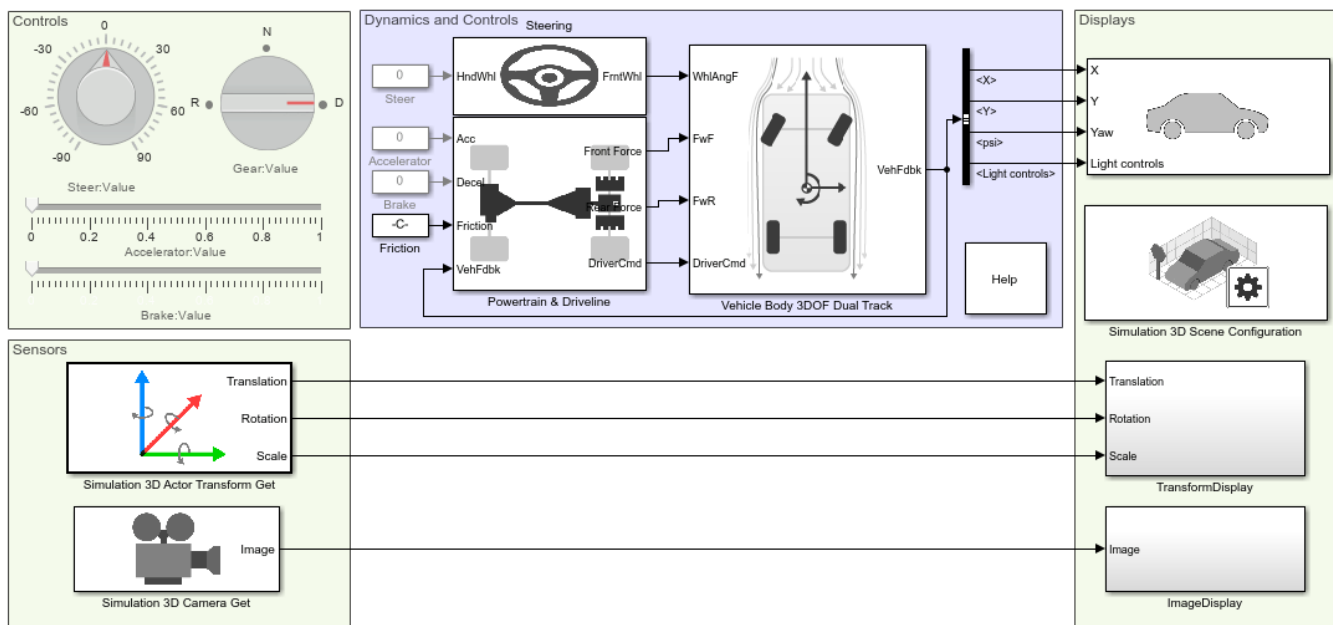
See Also

Simulation 3D Vehicle with Ground Following | Simulation 3D Camera Get | Simulation 3D Actor Transform Get | Simulation 3D Scene Configuration | Vehicle Body 3DOF

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8

External Websites

- Unreal Engine

Braking Test Reference Application

Simulate a full vehicle dynamics model undergoing a braking test, including a split-mu test. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use this reference application in ride and handling studies and chassis controls development to characterize the vehicle dynamics during a braking test. For information about this and similar maneuvers, see standards SAE J299_200901 and ISO 21994:2007.

For more information about the reference application, see "Braking Test" on page 3-11.
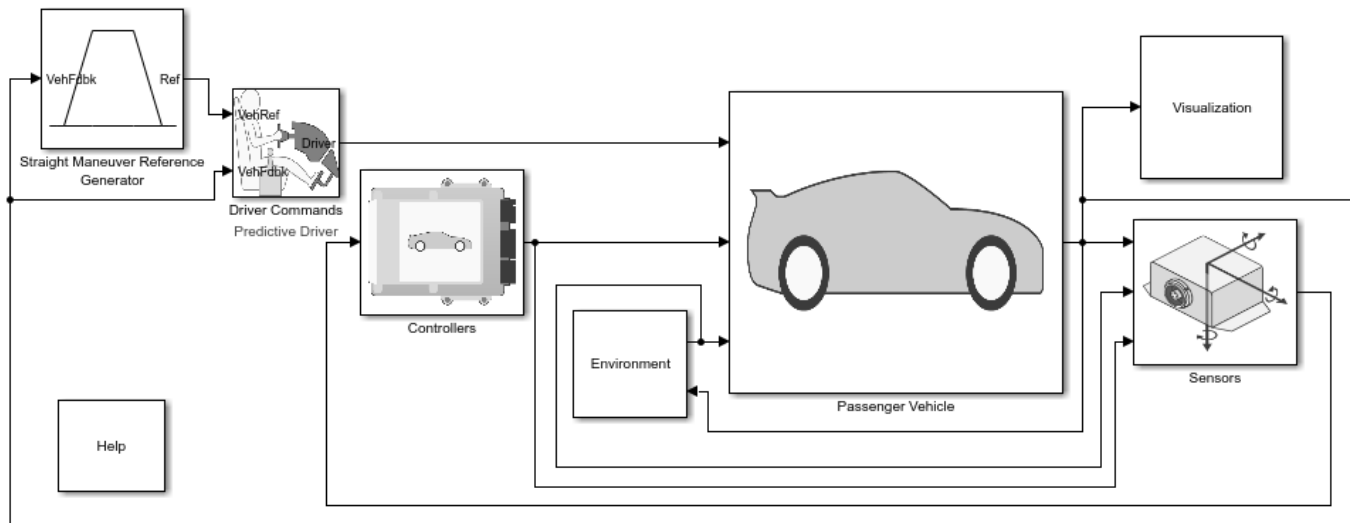
Copyright 2020-2022 The MathWorks, Inc.

References

[1] J299_200901. *Stopping Distance Test Procedure*. Warrendale, PA: SAE International, 2009.

[2] ISO 21994:2007. *Passenger cars — Stopping distance at straight-line braking with ABS — Open-loop test method*. Geneva: ISO, 2007.

[3] ISO 14512:1999. *Passenger cars — Straight-ahead braking on surfaces with split coefficient of friction -- Open-loop test procedure*. Geneva: ISO, 2007.

See Also

3D Engine | Road Track Friction | Straight Maneuver Reference Generator

More About

• "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6

• "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2

• "ISO 15037-1:2006 Standard Measurement Signals" on page 5-2

- Simulation Data Inspector

Longitudinal Motorcycle Braking Test Reference Application

Simulate an in-plane motorcycle undergoing a braking test. You can create your own versions, establishing a framework to test that your motorcycle meets the design requirements under normal and extreme driving conditions. Use this reference application in ride and handling studies and chassis controls development to characterize the vehicle dynamics of a motorcycle during a braking test.

For more information about the reference application, see "Longitudinal Motorcycle Braking Test" on page 3-4.
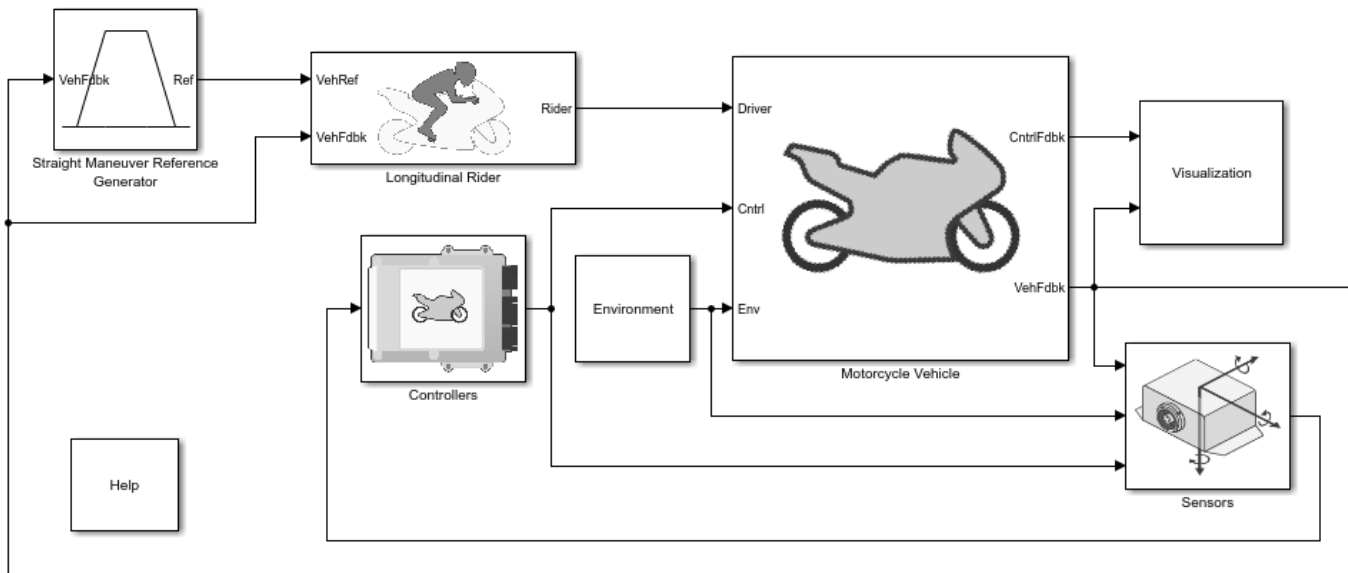
Copyright 2021-2022 The MathWorks, Inc.

See Also

Straight Maneuver Reference Generator | Motorcycle Body Longitudinal In-Plane | Motorcycle Chain | Simulation 3D Terrain Sensor

More About

- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- Simulation Data Inspector

Double Lane Change Reference Application

Simulate a full vehicle dynamics model undergoing a double-lane change maneuver according to standard ISO 3888-2. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application for vehicle dynamics ride and handling analysis and chassis controls development, including yaw stability and lateral acceleration limits.

For more information about the reference application, see "Double-Lane Change Maneuver" on page 3-22.
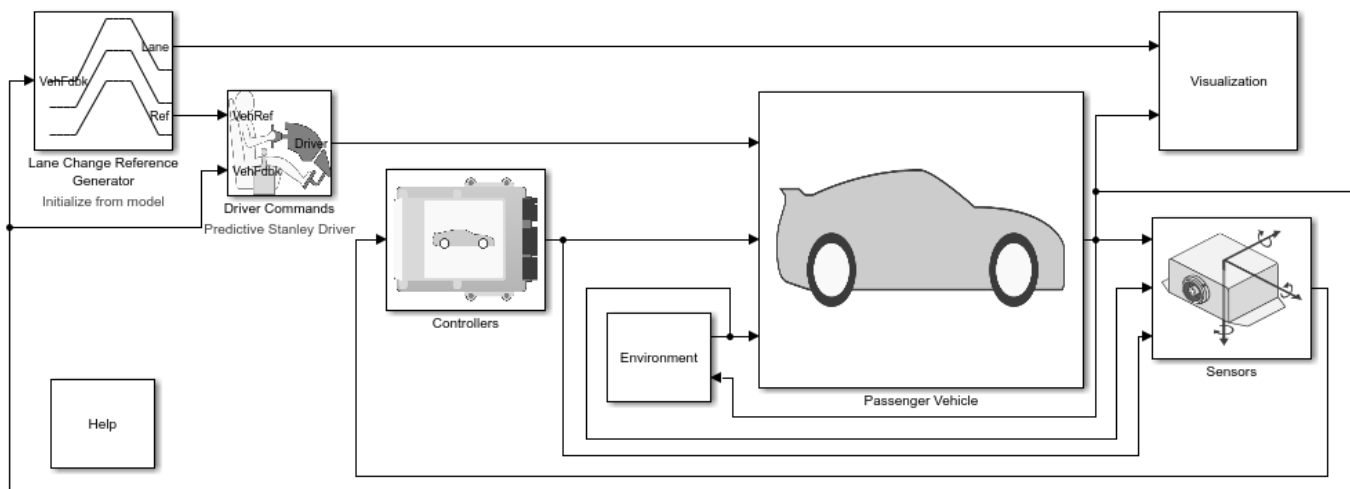
Copyright 2018-2023 The MathWorks, Inc.

References

[1] ISO 3888-2: 2011. *Passenger cars — Test track for a severe lane-change manoeuvre*.

See Also

Predictive Driver | Mapped SI Engine | Simulation 3D Terrain Sensor | 3D Engine

Related Examples

- "Send and Receive Double-Lane Change Scene Data" on page 3-93
- "Yaw Stability on Varying Road Surfaces" on page 1-16

More About

- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- Simulation Data Inspector

Swept-Sine Steering Reference Application

Simulate a full vehicle dynamics model undergoing a swept-sine steering maneuver. You can create your own versions, providing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application for vehicle dynamics ride and handling analysis and chassis controls development, including the dynamic steering response.

For more information about the reference application, see "Swept-Sine Steering Maneuver" on page 3-41.

Copyright 2018-2022 The MathWorks, Inc.

See Also
Longitudinal Driver | Mapped SI Engine | Simulation 3D Terrain Sensor | 3D Engine

Related Examples
- "Frequency Response to Steering Angle Input" on page 1-46

More About
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- Simulation Data Inspector

Increasing Steering Reference Application

Simulate a full vehicle dynamics model undergoing a slowly increasing steering maneuver according to standard SAE J266. You can create your own versions, establishing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application for lateral vehicle dynamics ride and handling analysis and chassis controls development, including the steering response.

For more information about the reference application, see "Slowly Increasing Steering Maneuver" on page 3-53.
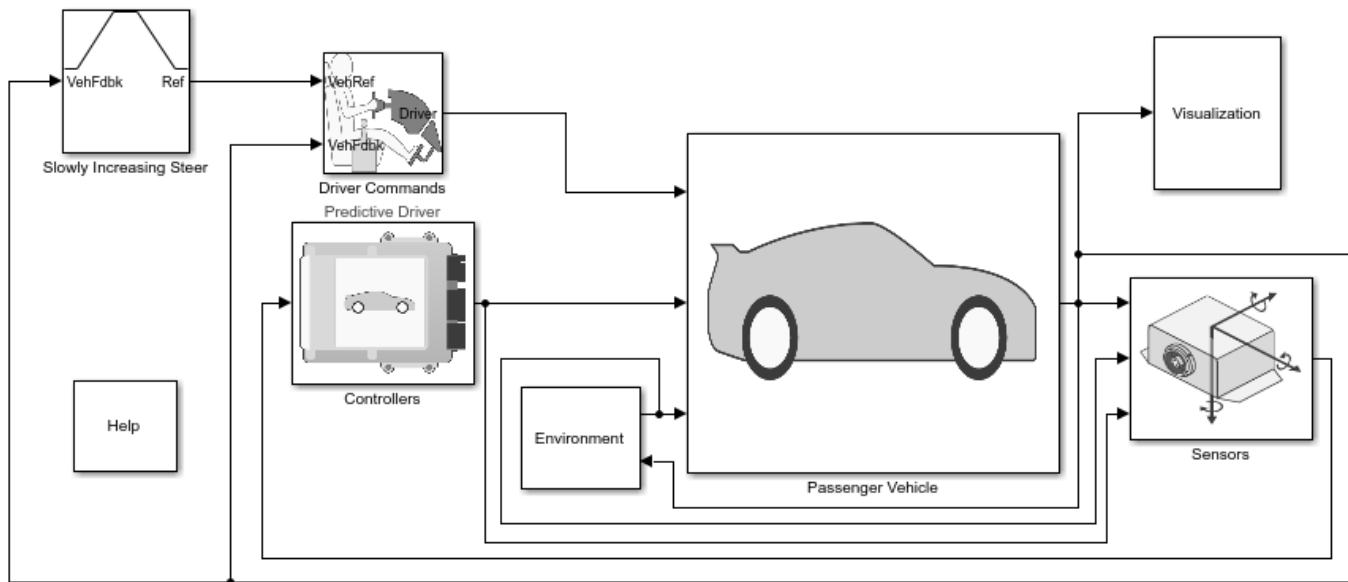
Copyright 2018-2023 The MathWorks, Inc.

References

[1] SAE J266. *Steady-State Directional Control Test Procedures For Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

See Also

Longitudinal Driver | Mapped SI Engine | Simulation 3D Terrain Sensor | 3D Engine

Related Examples

*   "Vehicle Steering Gain at Different Speeds" on page 1-26

More About

*   "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
*   Simulation Data Inspector

Constant Radius Reference Application

Simulate a full vehicle dynamics model undergoing a constant radius maneuver. You can create your own versions, providing a framework to test that your vehicle meets the design requirements under normal and extreme driving conditions. Use the reference application for vehicle dynamics ride and handling analysis and chassis controls development, including the dynamic steering response.

For more information about the reference application, see "Constant Radius Maneuver" on page 3-65.
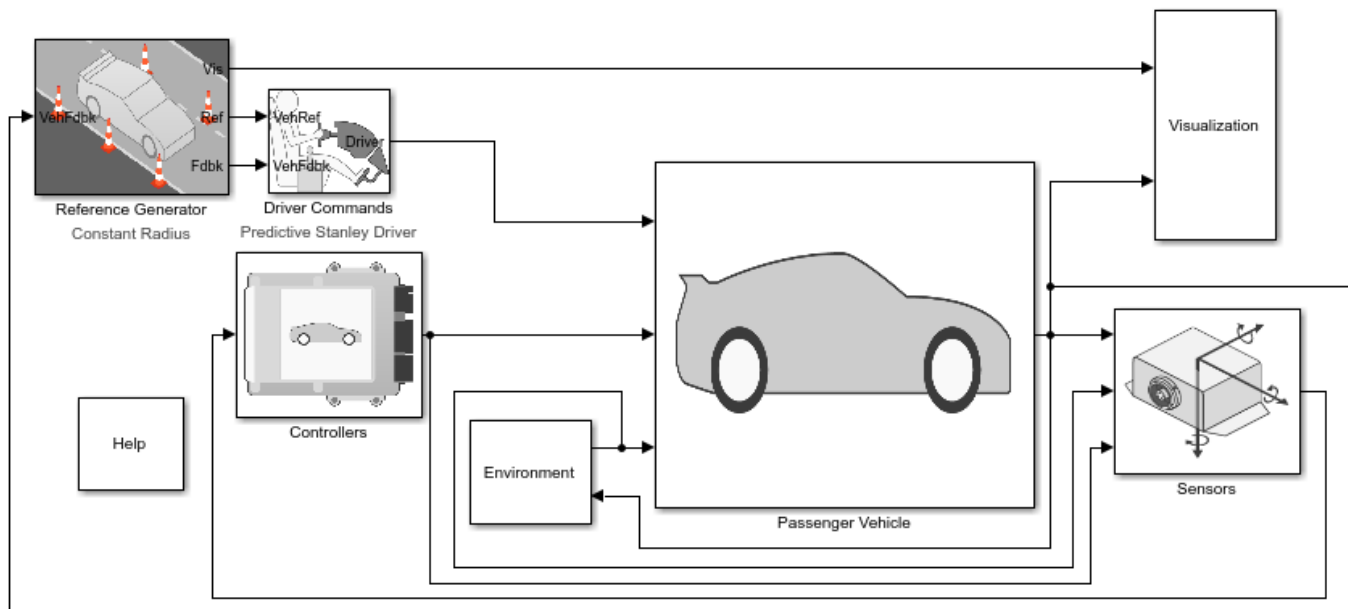
Copyright 2018-2023 The MathWorks, Inc.

References

[1] J266_199601. *Steady-State Directional Control Test Procedures for Passenger Cars and Light Trucks*. Warrendale, PA: SAE International, 1996.

[2] ISO 4138:2012. *Passenger cars — Steady-state circular driving behaviour — Open-loop test methods*. Geneva: ISO, 2012.

See Also

3D Engine | Driver Commands | Reference Generator | Simulation 3D Terrain Sensor

Related Examples

*   "Vehicle Lateral Acceleration at Different Speeds" on page 1-36

More About

*   "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2

- Simulation Data Inspector

Kinematics and Compliance Virtual Test Laboratory Reference Application

Generate optimized suspension parameters for the vehicle dynamics mapped suspension blocks.

Generate Mapped Suspension from Spreadsheet Data uses Model-Based Calibration Toolbox™ to generate calibrated suspension parameters from measured vertical force and suspension geometry data.

Generate Mapped Suspension from Simscape Suspension uses a Simscape™ Multibody™ suspension system to generate calibrated suspension parameters for the mapped suspension blocks.

Compare Mapped and Simscape Suspension Responses compares the mapped suspension with the Simscape Multibody suspension results.

For more information about the reference application, see "Kinematics and Compliance Virtual Test Laboratory" on page 3-76.

Virtual Kinematics and Compliance Test Laboratory
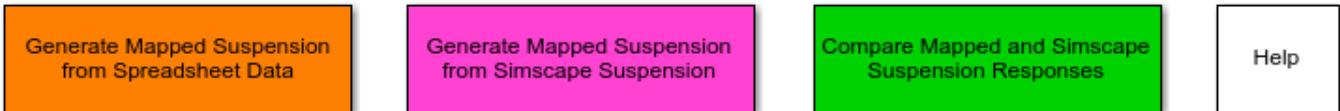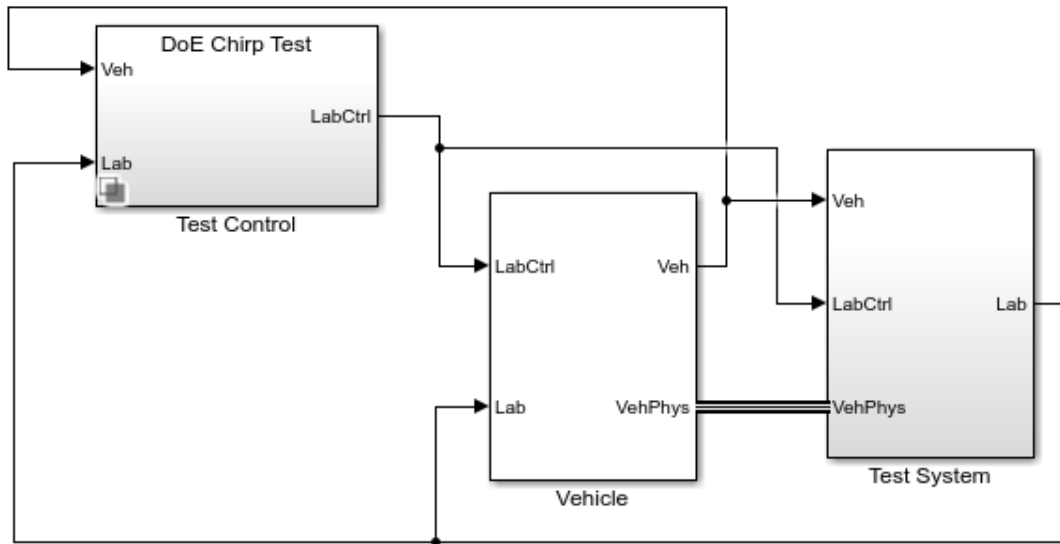
Copyright 2020-2021 The MathWorks, Inc.

See Also

Independent Suspension - Mapped | Solid Axle Suspension - Mapped

More About

• Simulation Data Inspector

Three-Axle Tractor Towing a Three-Axle Trailer

This example shows how to simulate a three-axle tractor towing a three-axle trailer for a commercial trucking application. The model implements a hitch subsystem, a sinusoidal steering or braking test, and an axle torque applied to the rear wheels of the tractor.

By default, the model implements the sinusoidal steering test. To implement the braking test, click the **Toggle Between Sine Steer and Braking** button. Use the braking test to assess the ability of the service braking control system to stop the commercial truck at .5 g deceleration without activating the ABS (anti-lock braking system). See Braking Test.

To view the simulation in the 3D visualization environment, use the Vehicle Monitor 3D variant. Right-click the Vehicle Monitor block and select **Variant > Label Mode Active Choice > Vehicle Monitor 3D**. See Run Simulation in 3D Visualization Environment.

To implement the tractor and trailer, by default, the three DOF model uses the Vehicle Body 3DOF and Trailer Body 3DOF blocks. Click the **Toggle Between 3DOF and 6DOF** button to configure a six DOF model that uses the Vehicle Body 6 DOF block, Trailer Body 6DOF block, and a 6DOF hitch subsystem. See Six Degrees-of-Freedom Model.

Model

Three-Axle Tractor Towing Three-Axle Trailer

Copyright 2020-2022 The MathWorks, Inc.

Run Simulation

On the **Simulation** tab, click **Run**. As the simulation runs, the Vehicle Position window provides the trace of the tractor and trailer.

Three-Axle Tractor Subsystem

To steer and drive the tractor, the three-axle tractor subsystem uses a sinusoidal wave steering input and an axle torque applied to the rear wheels. The subsystem includes models for the wheels, suspension, and vehicle body.

Three-Axle Trailer Subsystem

The three-axle trailer subsystem includes models for the wheels, suspension, and the trailer body.

Hitch Subsystem

When you select the three DOF model variant, the hitch model allows relative longitudinal, lateral, and yaw motion between the tractor and trailer. To limit the longitudinal and lateral motion, the hitch model implements a stiff translational spring-damper in the xy plane of the vehicle-fixed reference frame. The resulting spring-damper forces approximately limits the relative motion between the tractor and trailer to yaw rotation about a vertical axis at the hitch connection point. The hitch model transfers the vertical hitch force from the trailer to the tractor.

When you select the six DOF model variant, the hitch model allows relative longitudinal, lateral, vertical, and yaw motion between the tractor and trailer. The hitch model implements another translational spring-damper along the z -axis of the vehicle-fixed reference frame. The effects of hitch moments due to the relative rotations of the hitches are considered negligible.

- Spring forces are linear functions of the planar distance from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.
- Damper forces are linear functions of the planar velocity from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.

Braking Test

To implement the braking test, click **Toggle Between Sine Steer and Braking**. The model switches the Axle Torque block to indicate a braking test. The ABS block turns red to indicate ABS control.
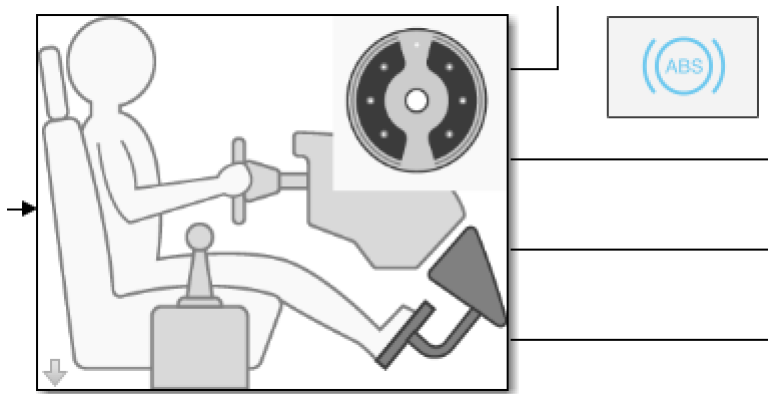
Then, on the **Simulation** tab, click **Run**. The scope block shows plots of the tractor acceleration and velocity, and tractor and trailer brake pressure.
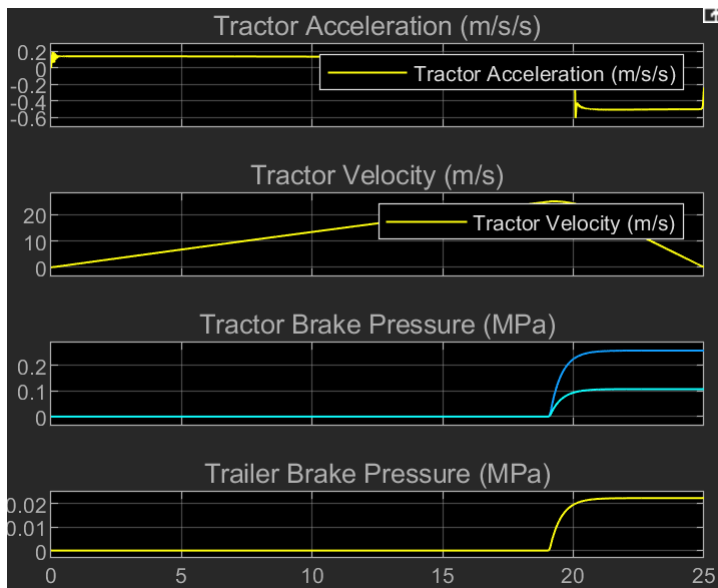
To view the simulation in the 3D visualization environment, use the Vehicle Monitor 3D variant.

Run Simulation in 3D Visualization Environment

In the Vehicle Monitor subsystem, use the Vehicle Monitor 3D variant to visualize the tractor and trailer in the 3D simulation environment.

1 Right-click the Vehicle Monitor block and select **Variant > Label Mode Active Choice > Vehicle Monitor 3D**.

2. Click **Run**. In the `AutoVrtlEnv` window, view the tractor and trailer in the 3D visualization environment. You can use the key numbers to change camera views of the tractor and trailer. For example, press 7 for a front left camera view.

Six Degrees-of-Freedom Model

To implement a 6 DOF tractor, trailer, and hitch model, click **Toggle Between 3DOF and 6DOF**. Then, on the **Simulation** tab, click **Run**.

Three-Axle Tractor Towing Three-Axle Trailer
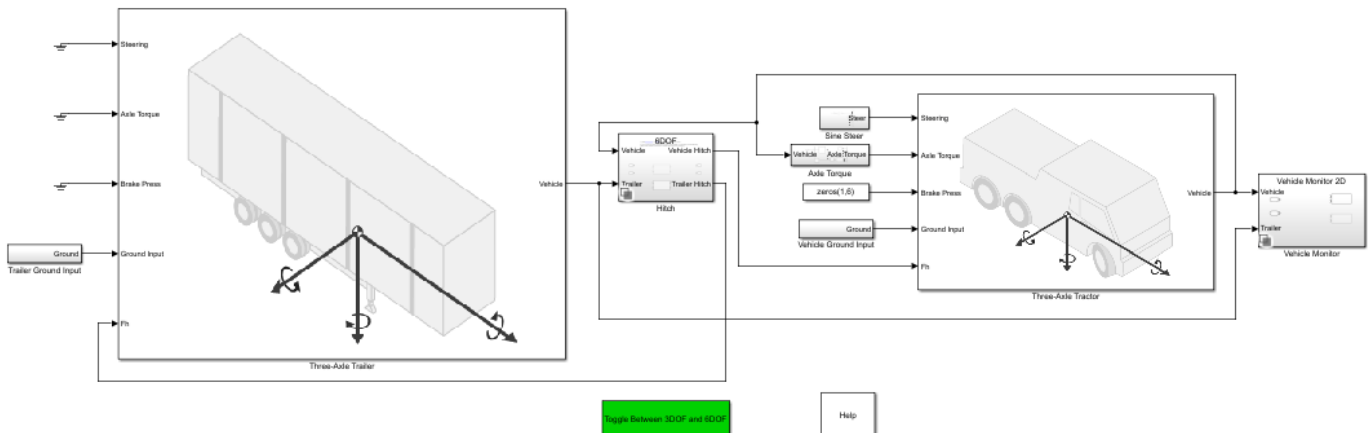
To view the simulation in the 3D visualization environment, use the Vehicle Monitor 3D variant.

See Also

Trailer Body 3DOF | Trailer Body 6DOF | Vehicle Body 3DOF | Vehicle Body 6DOF

More About

- "Two-Axle Tractor Towing a One-Axle Trailer" on page 7-32
- "Two-Axle Tractor Towing a Two-Axle Trailer" on page 7-27
- "Three-Axle Tractor Towing Two Three-Axle Trailers" on page 7-21

Three-Axle Tractor Towing Two Three-Axle Trailers

This example shows how to simulate a three-axle tractor towing two three-axle trailers for a commercial trucking application. The model implements hitch subsystems, sinusoidal wave steering input, and an axle torque applied to the rear wheels of the tractor.

To view the simulation in the 3D visualization environment, use the Vehicle Monitor 3D variant. Right-click the Vehicle Monitor block and select **Variant > Label Mode Active Choice > Vehicle Monitor 3D**. See Run Simulation in 3D Visualization Environment.

To implement the tractor and trailers, by default, the three DOF model uses the Vehicle Body 3DOF and Trailer Body 3DOF blocks. You can use the **Toggle Between 3DOF and 6DOF** button to configure a six DOF model that uses the Vehicle Body 6 DOF block, Trailer Body 6DOF blocks, and a 6DOF hitch subsystem. See Six Degrees-of-Freedom Model.
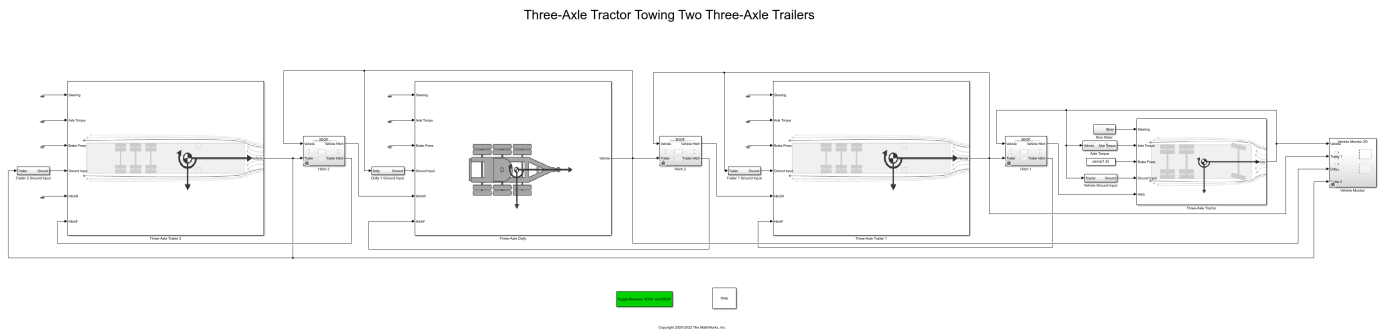
Model

Three-Axle Tractor Towing Two Three-Axle Trailers

Run Simulation

On the **Simulation** tab, click **Run**. As the simulation runs, the Vehicle Position window provides the trace of the tractor and trailer.
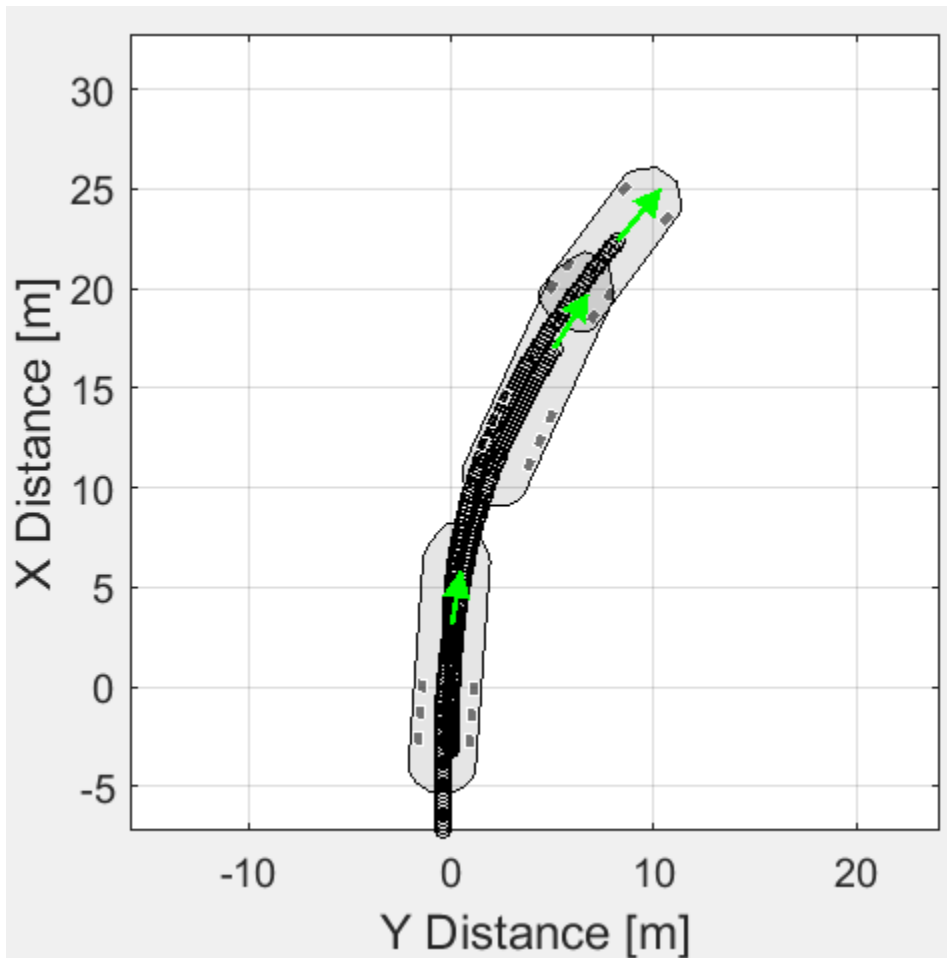
Three-Axle Tractor Subsystem

To steer and drive the tractor, the three-axle tractor subsystem uses a sinusoidal wave steering input and an axle torque applied to the rear wheels. The subsystem includes models for the wheels, suspension, and vehicle body.
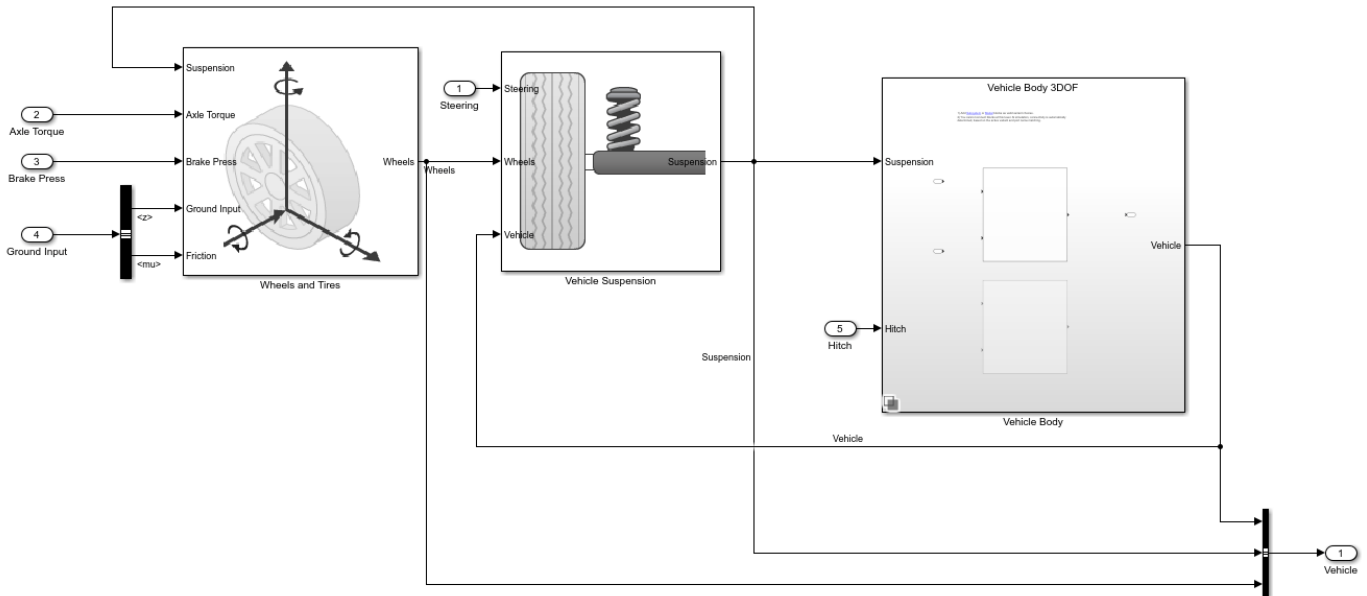
Three-Axle Trailer Subsystems

The three-axle trailer subsystems include models for the wheels, suspension, and the trailer body.
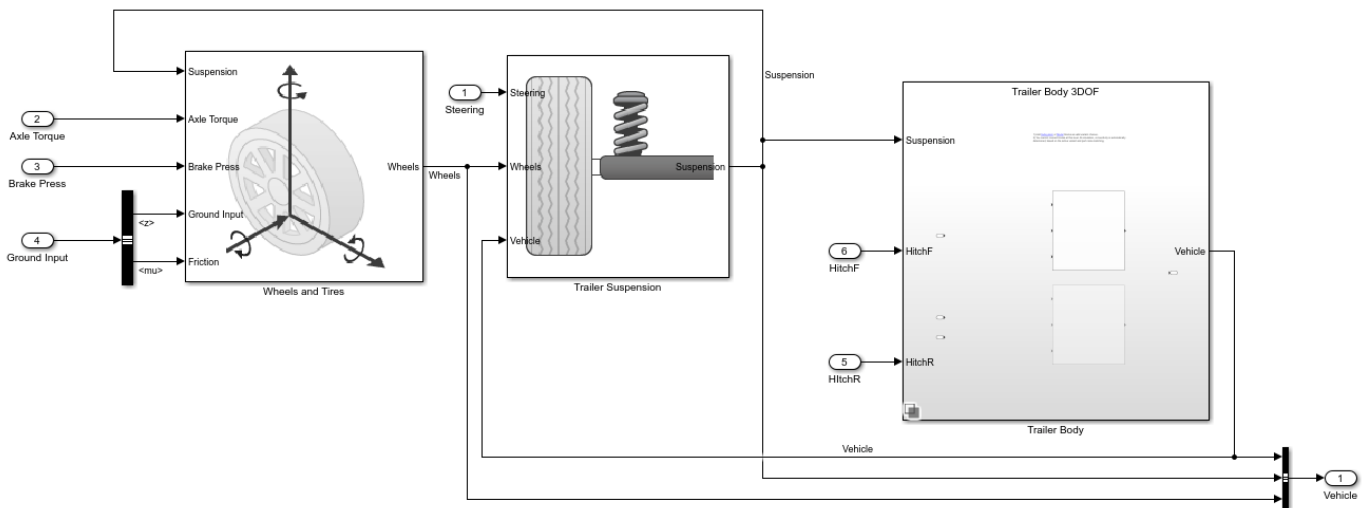
Hitch Subsystems

When you select the three DOF model variant, the hitch models allows relative longitudinal, lateral, and yaw motion between the tractor and trailer. To limit the longitudinal and lateral motion, the hitch model implements a stiff translational spring-damper in the xy plane of the vehicle-fixed reference frame. The resulting spring-damper forces approximately limits the relative motion between the tractor and trailer to yaw rotation about a vertical axis at the hitch connection point. The hitch model transfers the vertical hitch force from the trailer to the tractor.

When you select the six DOF model variant, the hitch model allows relative longitudinal, lateral, vertical, and yaw motion between the tractor and trailer. The hitch model implements another

translational spring-damper along the z-axis of the vehicle-fixed reference frame. The effects of hitch moments due to the relative rotations of the hitches are considered negligible.

- Spring forces are linear functions of the planar distance from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.
- Damper forces are linear functions of the planar velocity from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.

Three-Axle Dolly Subsystem

The three-axle dolly subsystem includes models for the wheels and suspension. To implement the dolly, the subsystem uses a Trailer Body block. If you enable the 3D environment, the model uses the Simulation 3D Dolly block to visualize the dolly.
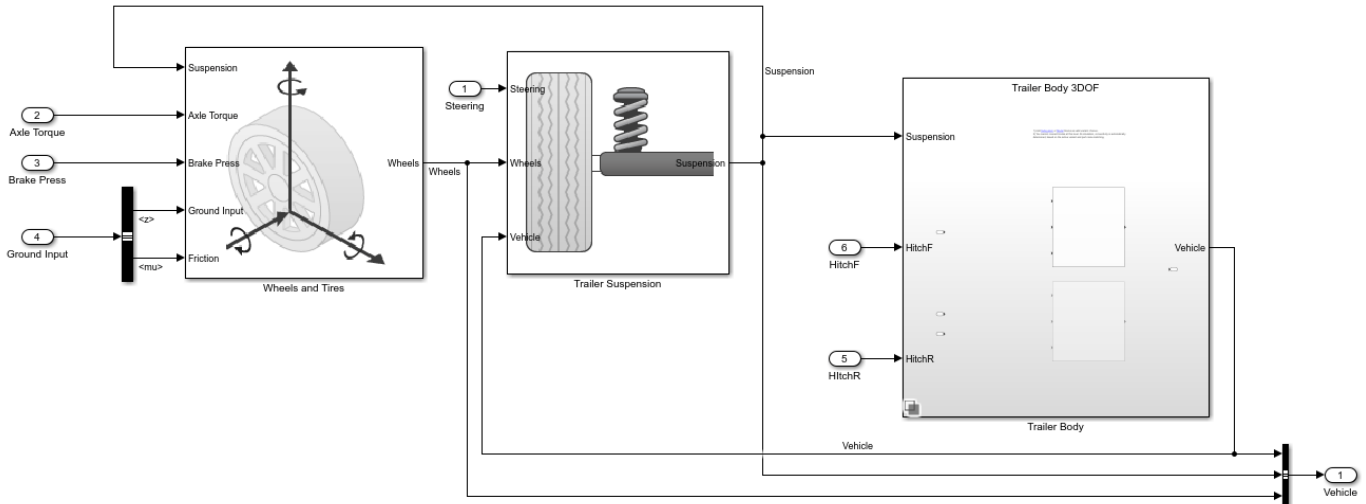
Run Simulation in 3D Visualization Environment

In the Vehicle Monitor subsystem, use the Vehicle Monitor 3D variant to visualize the tractor and trailer in the 3D simulation environment.

1   Right-click the Vehicle Monitor block and select **Variant** > **Label Mode Active Choice** > **Vehicle Monitor 3D**.

2. Click **Run**. In the `AutoVrtlEnv` window, view the tractor and trailer in the 3D visualization environment. You can use the key numbers to change camera views of the tractor and trailer. For example, press 7 for a front left camera view.

Six Degrees-of-Freedom Model
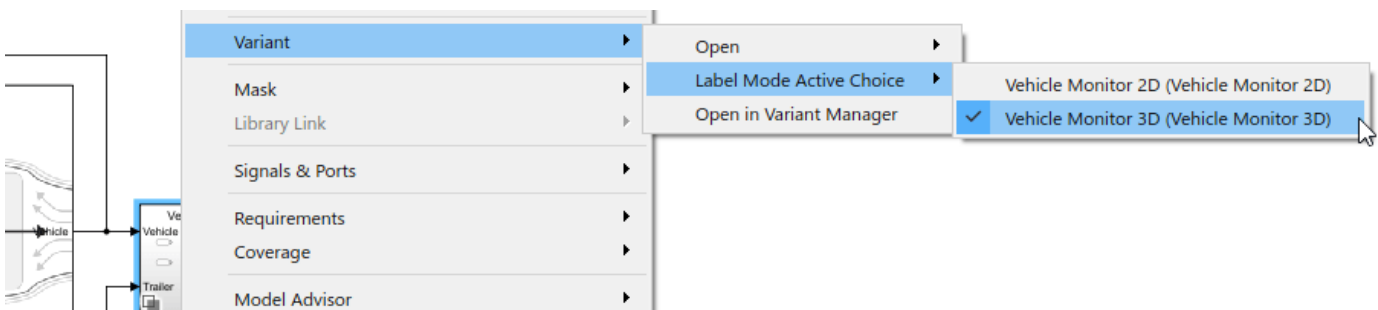
To implement a 6 DOF tractor, trailers, and hitch model, click **Toggle Between 3DOF and 6DOF**. Then, on the **Simulation** tab, click **Run**.
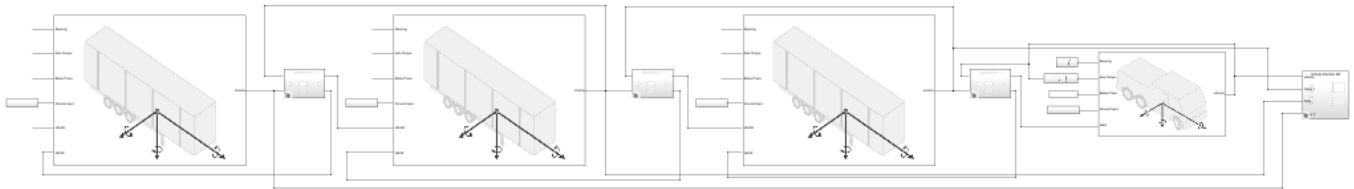
To view the simulation in the 3D visualization environment, use the Vehicle Monitor 3D variant.

See Also

Trailer Body 3DOF | Trailer Body 6DOF | Vehicle Body 3DOF | Vehicle Body 6DOF | Simulation 3D Dolly

More About

- "Two-Axle Tractor Towing a One-Axle Trailer" on page 7-32
- "Two-Axle Tractor Towing a Two-Axle Trailer" on page 7-27
- "Three-Axle Tractor Towing a Three-Axle Trailer" on page 7-14

Two-Axle Tractor Towing a Two-Axle Trailer

This example shows how to simulate a two-axle tractor towing a two-axle trailer for a commercial trucking application. The model implements a hitch subsystem, sinusoidal wave steering input, and an axle torque applied to the rear wheels of the tractor.

To implement the tractor and trailer, by default, the three degrees-of-freedom (DOF) model uses the Vehicle Body 3DOF and Trailer Body 3DOF blocks. You can use the **Toggle Between 3DOF and 6DOF** button to configure a six DOF model that uses the Vehicle Body 6 DOF block, Trailer Body 6DOF block, and a hitch subsystem. See Six Degrees-of-Freedom Model.

Model

Two-Axle Vehicle Towing Two-Axle Trailer

Copyright 2020-2021 The MathWorks, Inc.

Run Simulation

On the **Simulation** tab, click **Run**. As the simulation runs, the Vehicle Position window provides the trace of the tractor and trailer.

Two-Axle Vehicle Subsystem

To steer and drive the tractor, the two-axle tractor subsystem uses a sinusoidal wave steering input and an axle torque applied to the rear wheels of the tractor. The subsystem includes models for the tires, wheels, suspension, and vehicle body.

Two-Axle Trailer Subsystem

The two-axle trailer subsystem includes models for the wheels, suspension, and the trailer body.

Hitch Subsystem

When you select the three DOF model variant, the hitch model allows relative longitudinal, lateral, and yaw motion between the tractor and trailer. To limit the longitudinal and lateral motion, the hitch model implements a stiff translational spring-damper in the xy plane of the vehicle-fixed reference frame. The resulting spring-damper forces approximately limits the relative motion between the tractor and trailer to yaw rotation about a vertical axis at the hitch connection point. The hitch model transfers the vertical hitch force from the trailer to the tractor.

When you select the six DOF model variant, the hitch model allows relative longitudinal, lateral, vertical, and yaw motion between the tractor and trailer. The hitch model implements another translational spring-damper along the z-axis of the vehicle-fixed reference frame. The effects of hitch moments due to the relative rotations of the hitches are considered negligible.
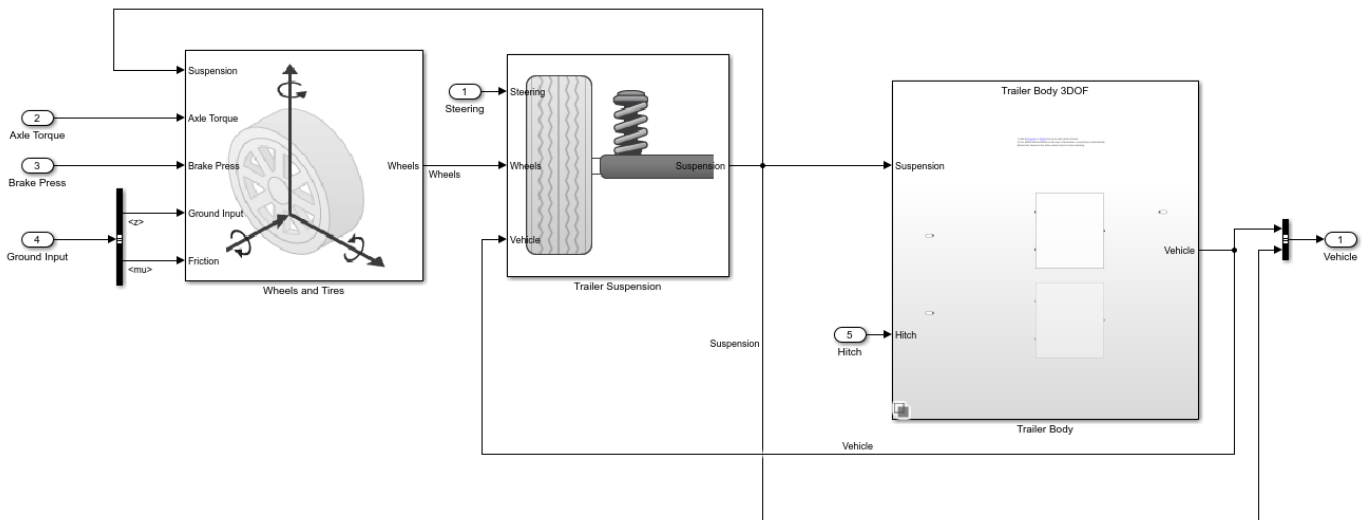
- Spring forces are linear functions of the planar distance from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.
- Damper forces are linear functions of the planar velocity from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.

Six Degrees-of-Freedom Model

To implement a six DOF tractor, trailer, and hitch model, click **Toggle Between 3DOF and 6DOF**. Then, on the **Simulation** tab, click **Run**.

Two-Axle Vehicle Towing Two-Axle Trailer

See Also

Trailer Body 3DOF | Trailer Body 6DOF | Vehicle Body 3DOF | Vehicle Body 6DOF

More About

- "Two-Axle Tractor Towing a One-Axle Trailer" on page 7-32
- "Three-Axle Tractor Towing a Three-Axle Trailer" on page 7-14
- "Three-Axle Tractor Towing Two Three-Axle Trailers" on page 7-21

Two-Axle Tractor Towing a One-Axle Trailer

This example shows how to simulate a two-axle tractor towing a one-axle trailer for a commercial trucking application. The model implements a hitch subsystem, sinusoidal wave steering input, and an axle torque applied to the rear wheels of the tractor.

To implement the tractor and trailer, by default, the three degrees-of-freedom (DOF) model uses the Vehicle Body 3DOF and Trailer Body 3DOF blocks. You can use the **Toggle Between 3DOF and 6DOF** button to configure a six DOF model that uses the Vehicle Body 6 DOF block, Trailer Body 6DOF block, and a hitch subsystem. See Six Degrees-of-Freedom Model.

Model

Two-Axle Vehicle Towing One-Axle Trailer
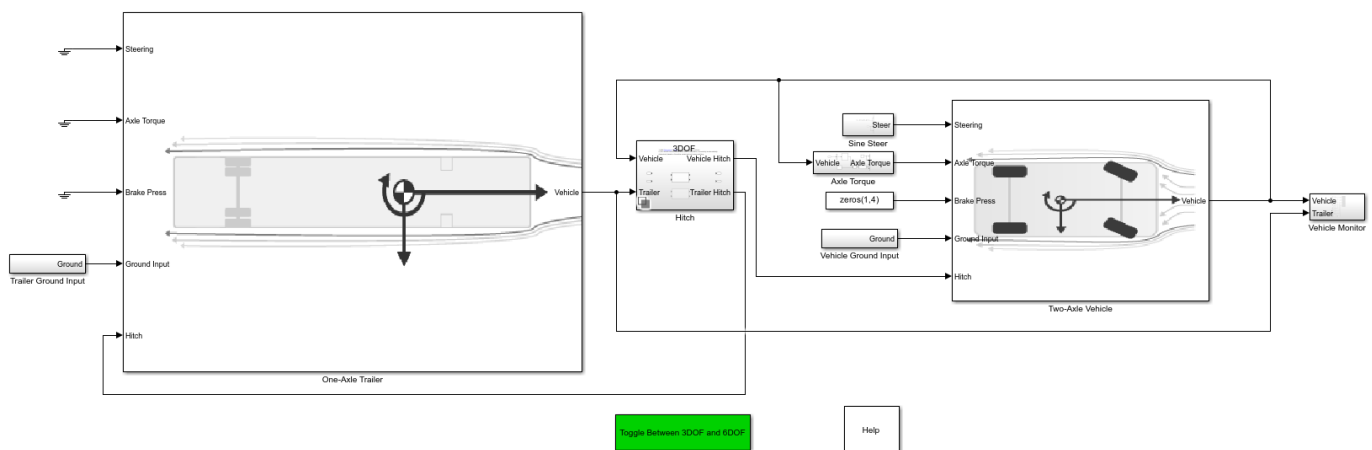
Copyright 2021 The MathWorks, Inc.

Run Simulation

On the **Simulation** tab, click **Run**. As the simulation runs, the Vehicle Position window provides the trace of the tractor and trailer.
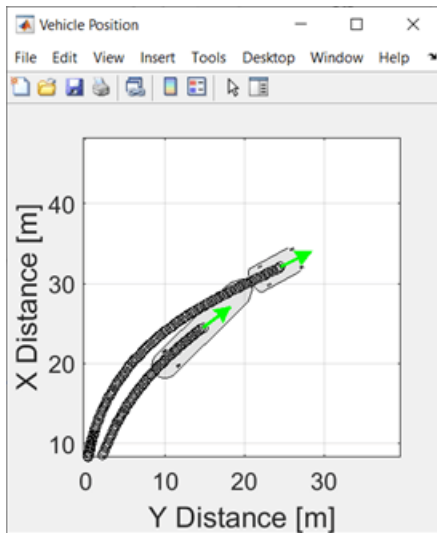
Two-Axle Vehicle Subsystem

To steer and drive the tractor, the two-axle tractor subsystem uses a sinusoidal wave steering input and an axle torque applied to the rear wheels of the tractor. The subsystem includes models for the tires, wheels, suspension, and vehicle body.
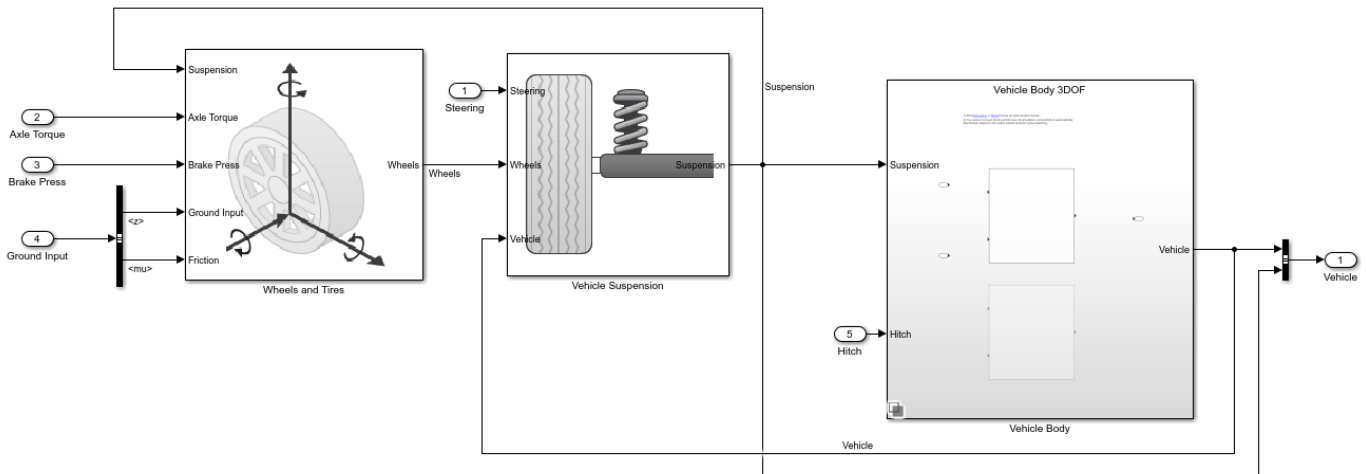
One-Axle Trailer Subsystem

The one-axle trailer subsystem includes models for the wheels, suspension, and the trailer body.

Hitch Subsystem

When you select the three DOF model variant, the hitch model allows relative longitudinal, lateral, and yaw motion between the tractor and trailer. To limit the longitudinal and lateral motion, the hitch model implements a stiff translational spring-damper in the xy plane of the vehicle-fixed reference frame. The resulting spring-damper forces approximately limits the relative motion between the tractor and trailer to yaw rotation about a vertical axis at the hitch connection point. The hitch model transfers the vertical hitch force from the trailer to the tractor.

When you select the six DOF model variant, the hitch model allows relative longitudinal, lateral, vertical, and yaw motion between the tractor and trailer. The hitch model implements another translational spring-damper along the z-axis of the vehicle-fixed reference frame. The effects of hitch moments due to the relative rotations of the hitches are considered negligible.

- Spring forces are linear functions of the planar distance from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.
- Damper forces are linear functions of the planar velocity from the tractor hitch location to the first trailer front hitch location in the inertial reference frame.

Six Degrees-of-Freedom Model

To implement a six DOF tractor, trailer, and hitch model, click **Toggle Between 3DOF and 6DOF**. Then, on the **Simulation** tab, click **Run**.

Two-Axle Vehicle Towing One-Axle Trailer

See Also

Trailer Body 3DOF | Trailer Body 6DOF | Vehicle Body 3DOF | Vehicle Body 6DOF

More About

- "Two-Axle Tractor Towing a Two-Axle Trailer" on page 7-27
- "Three-Axle Tractor Towing a Three-Axle Trailer" on page 7-14
- "Three-Axle Tractor Towing Two Three-Axle Trailers" on page 7-21

Follow Waypoints Around Oval Track

This example simulates a 3 degree-of-freedom (DOF) vehicle driving around an oval track that is specified by waypoints. The model contains waypoints and uses a MATLAB® function to determine the next heading waypoint.

To create your own track and use it in Unreal®, you can use RoadRunner and a RoadRunner plugin. To simulate a vehicle on the track in Unreal, you need the Vehicle Dynamics Blockset™ Interface for Unreal Engine® 4 Projects support package. For more information, see "Create and Use an Oval Track" on page 6-43.
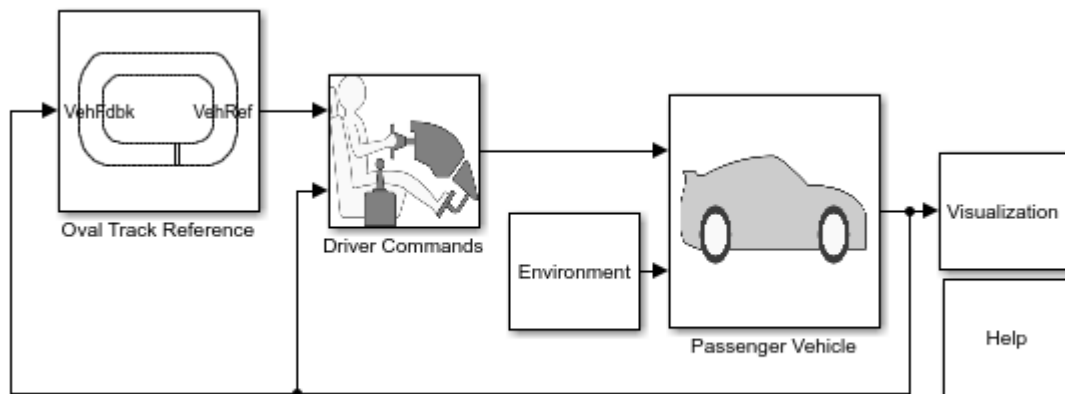
Copyright 2020-2022 The MathWorks, Inc.

Waypoints

The model workspace contains a variable, `TrackPoints`, that specifies X, Y, and reference pose waypoints for an oval track like the Indy 500® racing track. The points are X and Y locations in the Z-down vehicle coordinate system, in m. The reference poses are specified in rad.
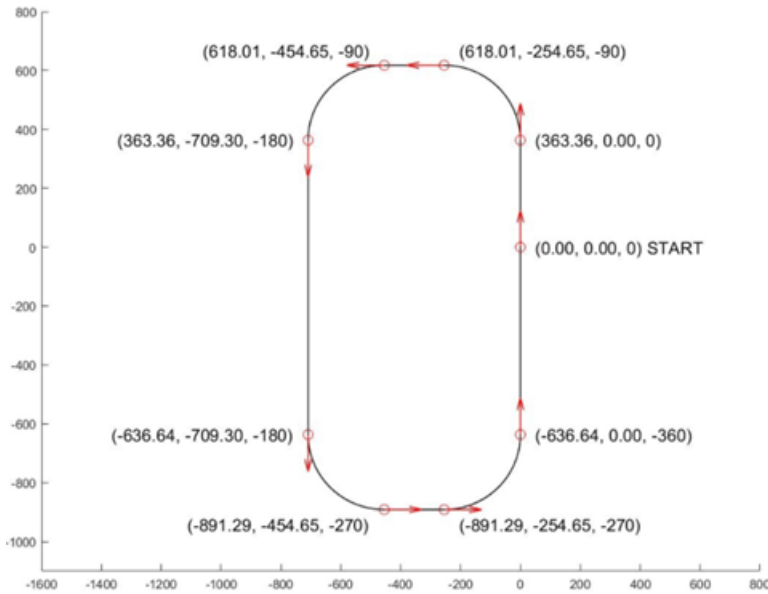
Run Simulation

On the **Simulation** tab, click **Run**. As the simulation runs, the Vehicle Position window provides the trace of the vehicle as it moves around the track.
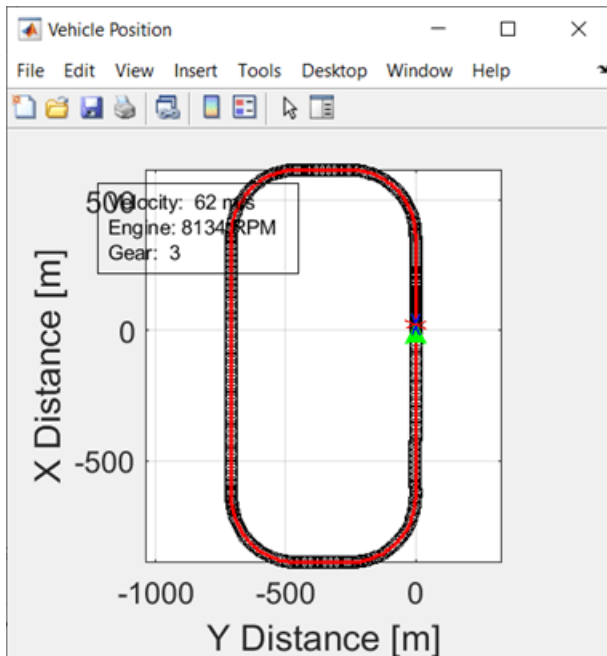
Oval Track Reference

The Oval Track Reference subsystem uses the `TrackPoints` waypoints to generate the reference path. The subsystem also includes a MATLAB® Function block that determines the next heading waypoint based on the current vehicle position and pose. The reference block then provides the vehicle commands to the driver block.
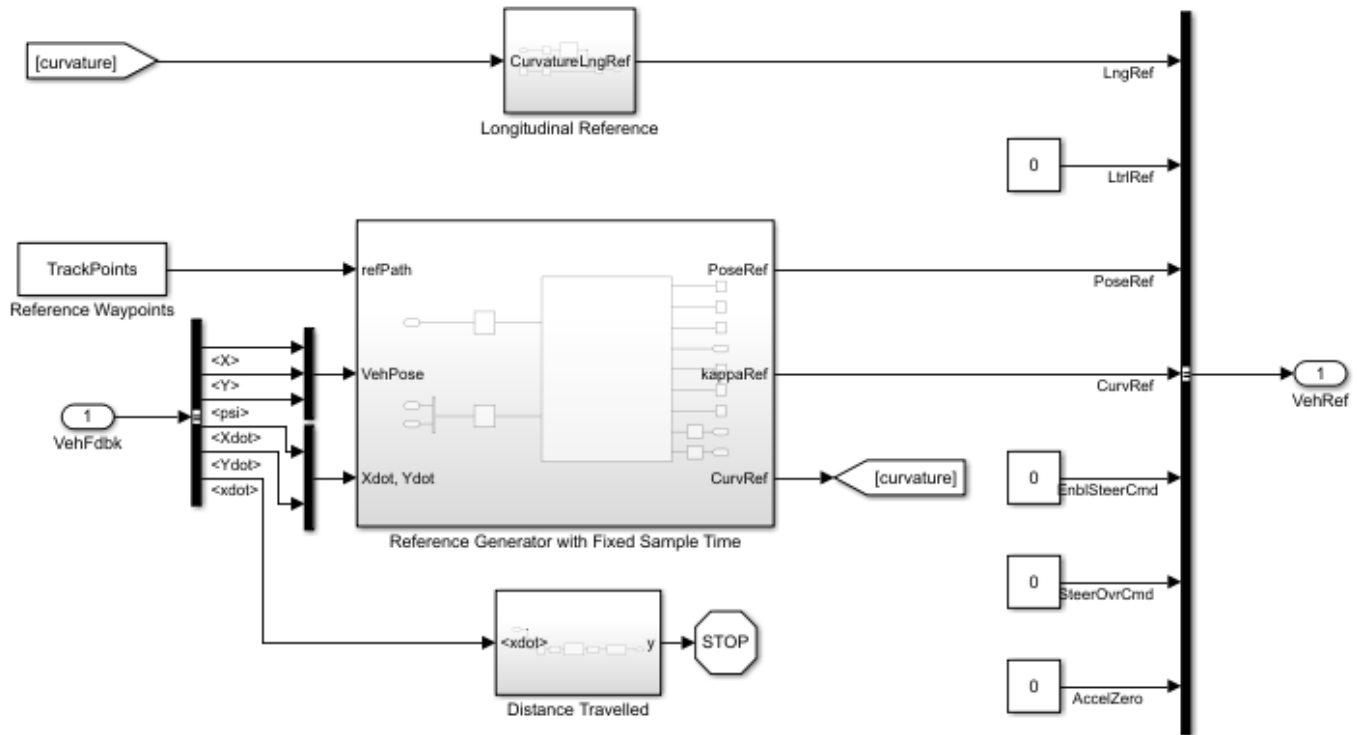
See Also
Predictive Driver | MATLAB Function | Vehicle Body 3DOF

More About
- "Create and Use an Oval Track" on page 6-43
- "Install Support Package and Configure Environment" on page 6-10
- "Export to Unreal Using Filmbox (.fbx) File" (RoadRunner)
- "RoadRunner"

Read and Write Block Parameters to Excel

If you manage model data in external files, you can use scripts to pass the data between the data file and a Simulink® model. This example shows you how to read block parameter data from and write parameter data to an Excel® data file. Specifically, the example provides functions that read and write Mapped SI Engine parameter data. You can adapt the functions to read and write parameters for additional blocks.

Open Mapped SI Engine Block

Open the Mapped SI Engine block in the double-lane change reference application.

Open the Double-Lane Change Reference Application

```
workDir = pwd;
vdynblksDblLaneChangeStart;
cd(workDir);
```

Set a variable equal to the block path.

```
bp = 'SiMappedEngineV/Mapped SI Engine'; % block path
```

Open Mapped SI Engine Block

In the `DLCReferenceApplication` model, navigate to `Passenger Vehicle` > `Ideal Mapped Engine` > `SiMappedEngineV`. Open the Mapped SI Engine block. The **Breakpoints for commanded torque**, **Breakpoints for engine speed input**, **Number of cylinders**, **Crank revolutions per power stroke**, and **Total displaced volume** parameters are set to workspace variables.

Breakpoints for commanded torque input, f_tbrake_t_bpt [Nm]:	f_tbrake_t_bpt
Breakpoints for engine speed input, f_tbrake_n_bpt [rpm]:	f_tbrake_n_bpt

Engine Characteristics

Number of cylinders, NCyl []:	NCyl
Crank revolutions per power stroke, Cps []:	Cps
Total displaced volume, Vd [m^3]:	Vd
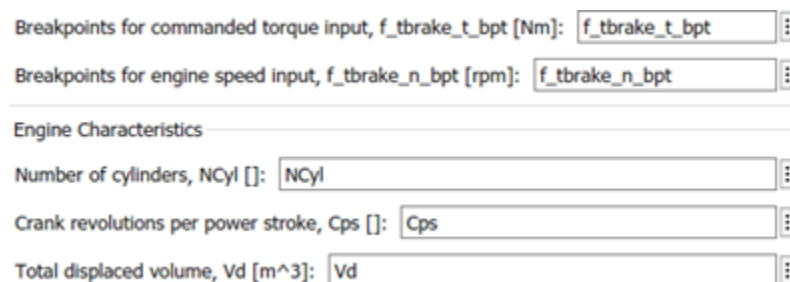
The functions in the example overwrite the workspace variables with the values in the data file.

Specify Data File Configuration

First, specify the file name. This example file `SiEngineData.xlsx` contains three sheets. The first sheet contains scalar values for commanded torque breakpoints, breakpoints for engine speed input breakpoints, number of cylinders, crank revolutions, and total displaced volume. The second sheet contains a table values for the brake torque map. The third sheet contains table values for the fuel torque map.

```
fileName = 'SiEngineData.xlsx';
```

Note that the first sheet in the file specifies the **Number of cylinders, Ncyl** parameter as 6.

MappedSIEngineBlock		VarName2	VarName3	VarName4
Text	▼	Text ▼	Number ▼	Number ▼
1 Mapped SI Engine Block				
2 Input Configuration Parameters		Unit	Value	
3 Breakpoints for commanded torqu...		Nm	0	35
4 Breakpoints for engine speed inpu...		rpm	0	641.2000
5				
6 Engine Characteristics		Unit	Value	
7 Number of cylinders, Ncyl		-	6	
8 Crank revolutions per power strok...		-	2	
9 Total displaced volume, Vd		m^3	0.0036	

Next, define the configuration data for the engine subsystem. This example sets a configuration for double variables of size scalar, vector, or a 2D array.

- Scalar data structure specifies the data on the first sheet.
- Vector data structure specifies the data on the second sheet.
- Array data structure specifies the data on the third sheet.

```
engData = struct(); % engine parameter data

% Scalar data
engData.Ncyl = struct('xlSheet','Main', 'xlRange','C7:C7', 'slBlockPath',bp, 'slBlockParam','Ncy
engData.Cps = struct('xlSheet','Main', 'xlRange','C8:C8', 'slBlockPath',bp, 'slBlockParam','Cps')
engData.Vd = struct('xlSheet','Main', 'xlRange','C9:C9', 'slBlockPath',bp, 'slBlockParam','Vd');

% Vector data
engData.t_bpt = struct('xlSheet','Main', 'xlRange','C3:R3', 'slBlockPath',bp, 'slBlockParam','f_
engData.n_bpt = struct('xlSheet','Main', 'xlRange','C4:R4', 'slBlockPath',bp, 'slBlockParam','f_

% 2D array data
engData.torque = struct('xlSheet','Brake Torque', 'xlRange','B2:Q17', 'slBlockPath',bp, 'slBlockI
engData.fuel = struct('xlSheet','Fuel Map', 'xlRange','B2:Q17', 'slBlockPath',bp, 'slBlockParam'
```

Read Mapped SI Engine Block Parameters

Update the Mapped SI Engine block to the values specified in the data file.

Read Data File and Update Parameters

Use this code to read the data file and update the Mapped SI Engine block parameters.

```
f = fields(engData);
for idx = 1:length(f)
    try
        var = getfield(engData, f{idx});
        % read value from Excel
        val = readmatrix(fileName, 'Sheet',var.xlSheet, 'Range',var.xlRange);
        % open Simulink model
        mdl = fileparts(var.slBlockPath);
        open_system(mdl);
        % set parameter value and save model
        set_param(var.slBlockPath, var.slBlockParam, mat2str(val));
        save_system(mdl);
    catch ME
        % return any error info
```

```
        disp(getReport(ME, 'extended', 'hyperlinks', 'on'))
        fprintf('\nContinuing to next variable...\n\n');
    end
end
fprintf('Done writing values to Simulink\n')
```

Done writing values to Simulink

Open Mapped SI Engine Block

In the `DLCReferenceApplication` model, navigate to `Passenger Vehicle > Ideal Mapped Engine > SiMappedEngineV`. Open the Mapped SI Engine block. The **Breakpoints for commanded torque**, **Breakpoints for engine speed input**, **Number of cylinders**, **Crank revolutions per power stroke**, and **Total displaced volume** parameters are set to the values specified in the data file. Confirm that the **Brake torque map** and **Fuel flow map parameters** are the same as the values specified in the data file.

Breakpoints for commanded torque input, f_tbrake_t_bpt [Nm]: 21 262.14 281.07 300]

Breakpoints for engine speed input, f_tbrake_n_bpt [rpm]: 6.3 3755.8 4015.4 4274.9]

Engine Characteristics

Number of cylinders, NCyl []: 6

Crank revolutions per power stroke, Cps []: 2

Total displaced volume, Vd [m^3]: 0.00359999893917068

Write Modified Parameters to Data File

In the Mapped SI Engine block, change the **Number of cylinders, NCyl** parameter from 6 to 8. Click **Apply**. Save the model.

Breakpoints for commanded torque input, f_tbrake_t_bpt [Nm]: 21 262.14 281.07 300]

Breakpoints for engine speed input, f_tbrake_n_bpt [rpm]: 6.3 3755.8 4015.4 4274.9]

Engine Characteristics

Number of cylinders, NCyl []: 8

Crank revolutions per power stroke, Cps []: 2

Total displaced volume, Vd [m^3]: 0.00359999893917068

Alternatively, use this code to update the parameter and save the model.

```
set_param(bp,'Ncyl','8');
save_system('SiMappedEngineV');
```

Write Parameter Data to File

Create a copy of the data file. Write the modified parameter data to the copy of the data file.

```
copyfile('SiEngineData.xlsx','SiEngineDataCopy.xlsx','f');
fileName = 'SiEngineDataCopy.xlsx';
```
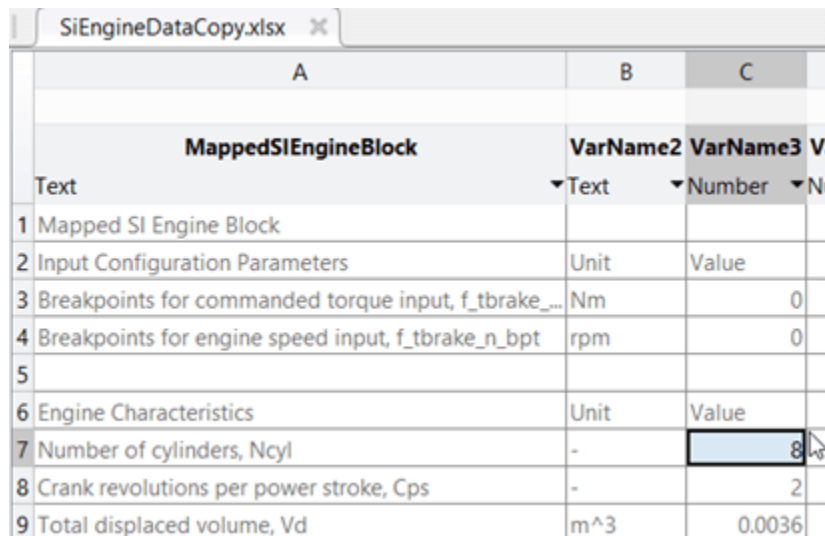
Next, use this code to write the Mapped SI Engine block **Breakpoints for commanded torque**, **Breakpoints for engine speed input**, **Number of cylinders**, **Crank revolutions per power stroke**, **Total displaced volume**, **Brake torque map**, and **Fuel flow map** parameters to the data file.

```
% Read data from Simulink model then write to Excel
f = fields(engData);
for idx = 1:length(f)
    try
        var = getfield(engData, f{idx});
        % open Simulink model
        mdl = fileparts(var.slBlockPath);
        open_system(mdl);
        % read value from Simulink
        val = str2num(get_param(var.slBlockPath, var.slBlockParam));
        % write value to Excel
        writematrix(val, fileName, 'Sheet',var.xlSheet, 'Range',var.xlRange);
    catch ME
        % return any error info
        disp(getReport(ME, 'extended', 'hyperlinks', 'on'))
        fprintf('\nContinuing to next variable...\n\n');
    end
end
fprintf('Done writing values to Excel\n')
```

```
Done writing values to Excel
```

Open the file with the modified data. Confirm that the number of cylinders in the data file is 8.

See Also

Mapped SI Engine

Related Examples

- "Double Lane Change Reference Application" on page 7-7

Generate Skidpad Test

This example shows how to simulate a skidpad test similar to events held in student competitions by Formula Student, Formula SAE, and other organizations internationally. In these, student teams build, operate, and present vehicles of their own design.

In this example, the vehicle starts from rest, then:

1 Accelerates to enter the course

2 Completes two clockwise laps

3 Reverses its steering direction

4 Completes two counterclockwise laps

5 Exits and brakes to a stop
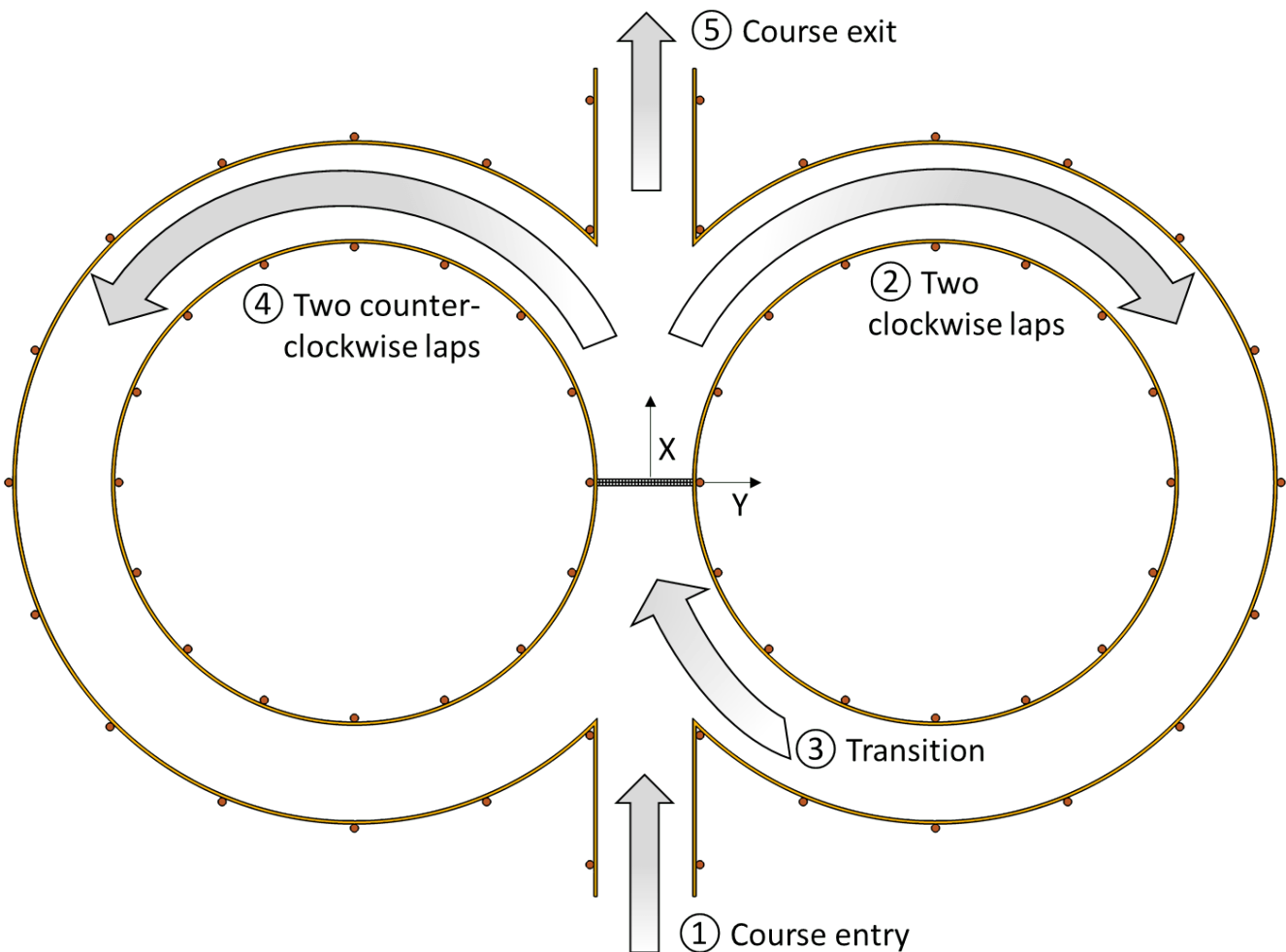
Open Model

The skidpad model includes a reference path, driver, vehicle, and visualization aides. To open the model, use this command.

```
open_system('vdynblksskidpad');
```


Copyright 2022 The MathWorks, Inc.
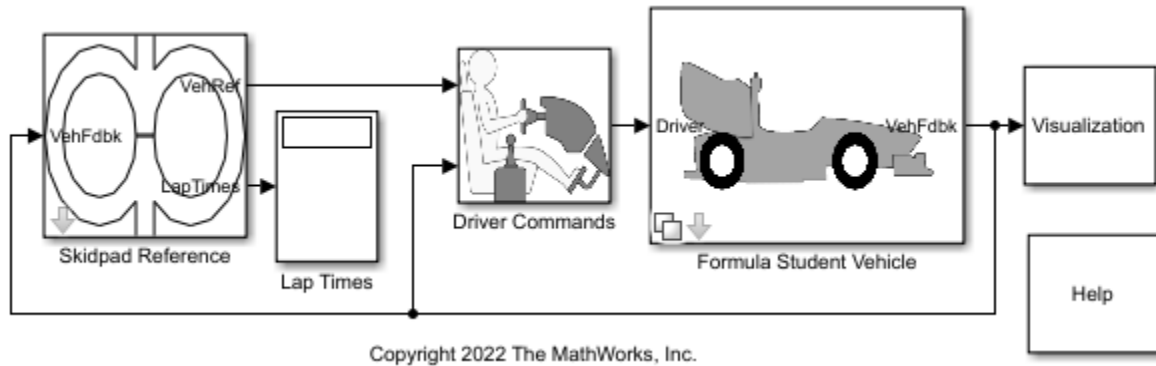
The model workspace variable TrackPoints contains the reference waypoints used to define the reference pose. You can edit or replace them with waypoints of your own. To plot the waypoints, use this code.

```
mdlWks = get_param('vdynblksskidpad','ModelWorkspace');
TrackPoints = evalin(mdlWks,'TrackPoints');
figure;
plot(TrackPoints(:,2),TrackPoints(:,1),'ro-')
hold on
arrowLen = 1.5;
quiver(TrackPoints(:,2),TrackPoints(:,1),arrowLen*sin(TrackPoints(:,3)),arrowLen*cos(TrackPoints
box on
grid on
xlabel('Y [m]')
ylabel('X [m]')
axis equal
legend('Waypoints','Path Direction')
```

7-45
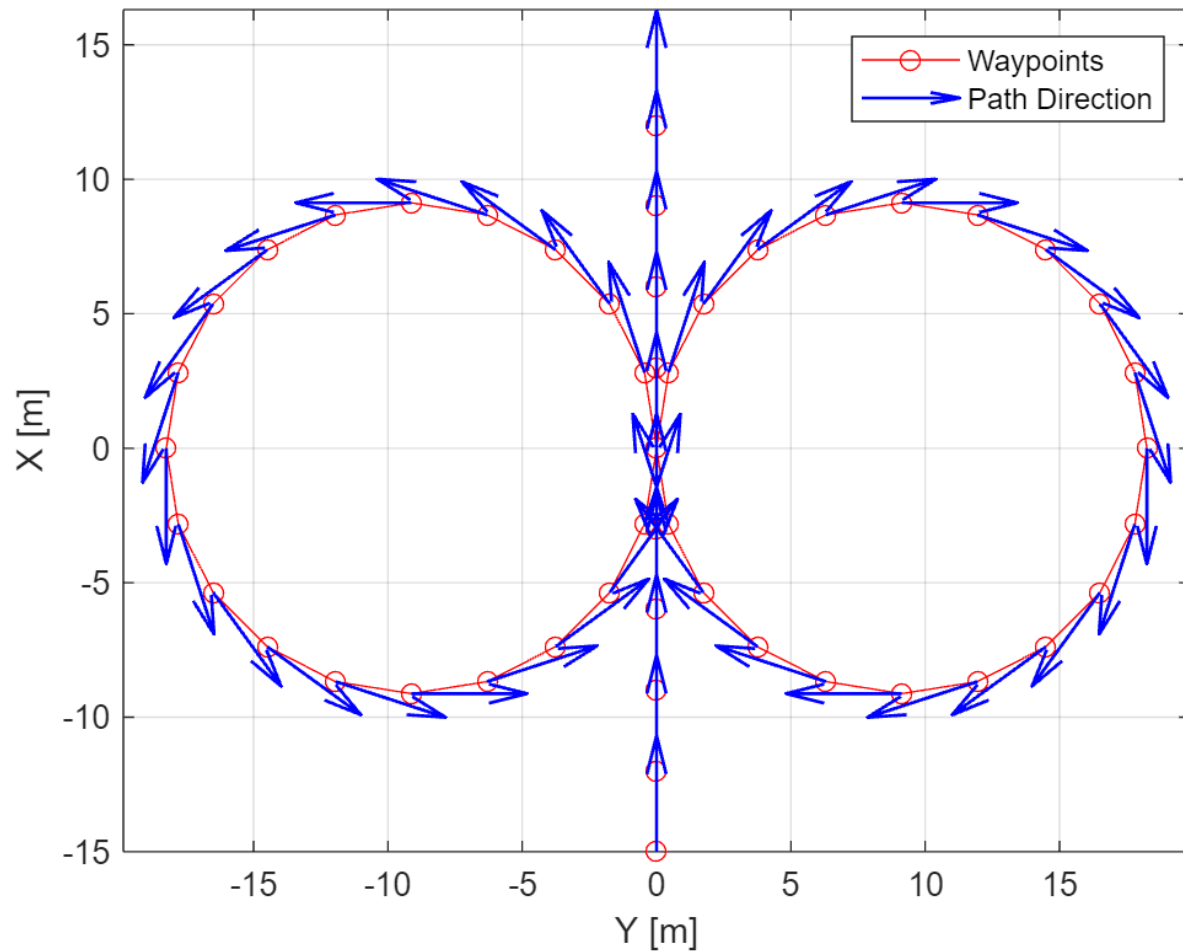
The default vehicle path follows the centerline of the course, with some X,Y points duplicated because the vehicle completes multiple laps. You can change factors such as the number of laps or scaling via the blocks found in the Skidpad Reference subsystem. You can also experiment with path points to improve lap times.

Simulate Model

To run the model and observe the vehicle completing the course, use this command.

```
sim('vdynblksskidpad');
```
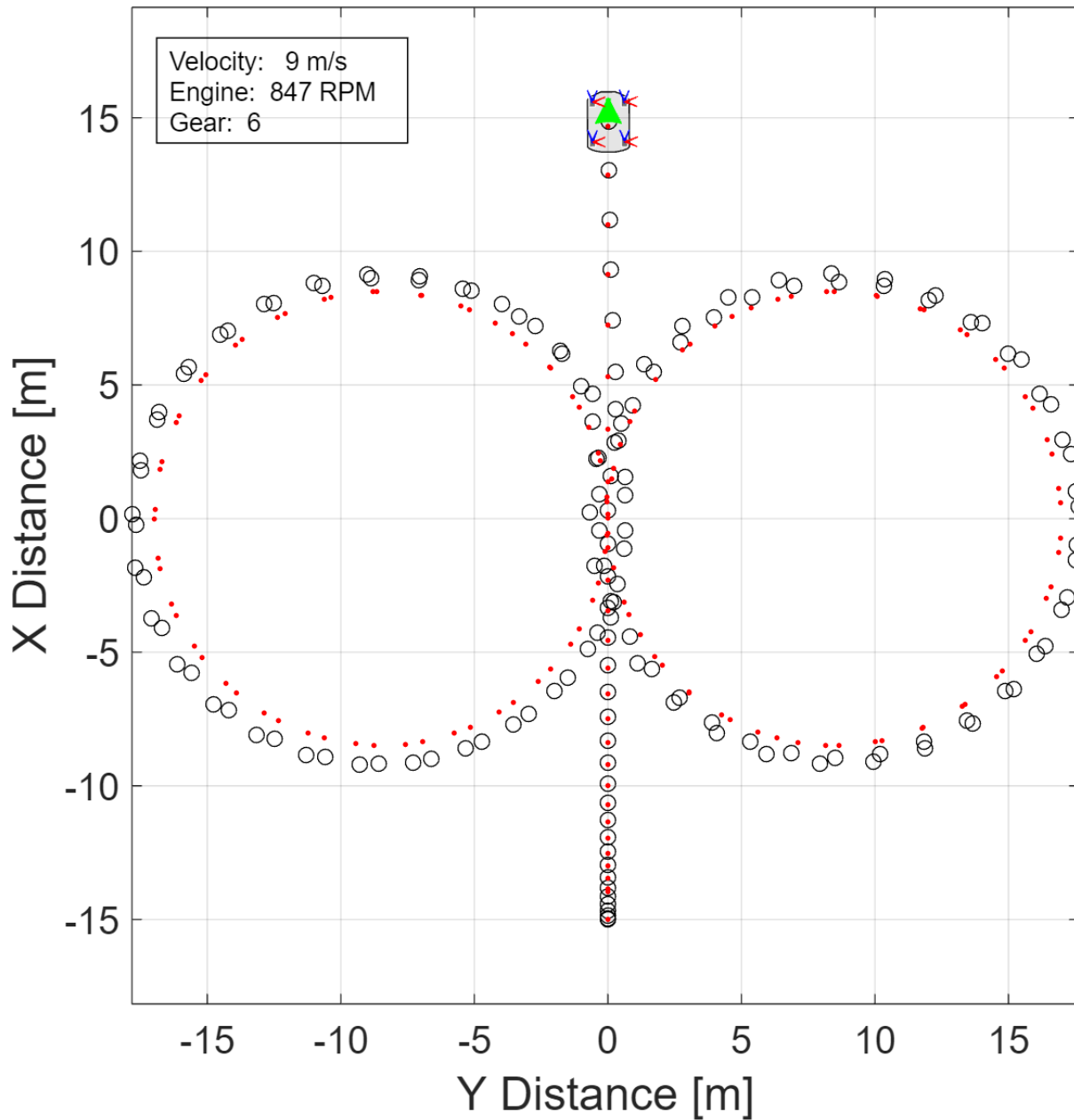
By default, an overhead figure depicting the reference and simulated vehicle path, and acceleration diagram are opened and updated during simulation. Lap times are estimated and displayed in order at the top level of the model via a Display block.

You can model the vehicle as either of two variant subsystems, using blocks from the **Vehicle Dynamics Blockset** library: Simulink Physics or Game Engine Physics. To set the vehicle type, double-click the Formula Student Vehicle block and select from the Vehicle Type list. To set the vehicle type programmatically, use the `set_param` command.

By default, the Simulink Physics setting uses a 7DOF vehicle containing multiple subsystems and components, including an ideal powertrain, driveline, tires, and chassis. The Game Engine Physics setting has similar parameters but uses a simulation 3D interface to configure and solve for the dynamic response of the predefined vehicle template. Given the differences in tire models and solver engines between the two settings, differences in results are to be expected. Choose your setting based on your modeling and visualization preferences.

Use this command to set the model to use the game engine vehicle.

vehType = Game Engine Phys... ▼

```
vehType =
"Game Engine Physics"
```

```
% set_param( 'vdynblksskidpad/Formula Student Vehicle','vehType',vehType);
```

Enable 3D Visualization

Before running the model in the 3D visualization environment, review the "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6. To enable the 3D visualization, use these commands.

```
set_param('vdynblksskidpad/Visualization/3D Visualization','engine3D','Enabled - Simulink 3D Veh
set_param('vdynblksskidpad/Visualization/3D Visualization','conesOn','on');
sim('vdynblksskidpad');
```
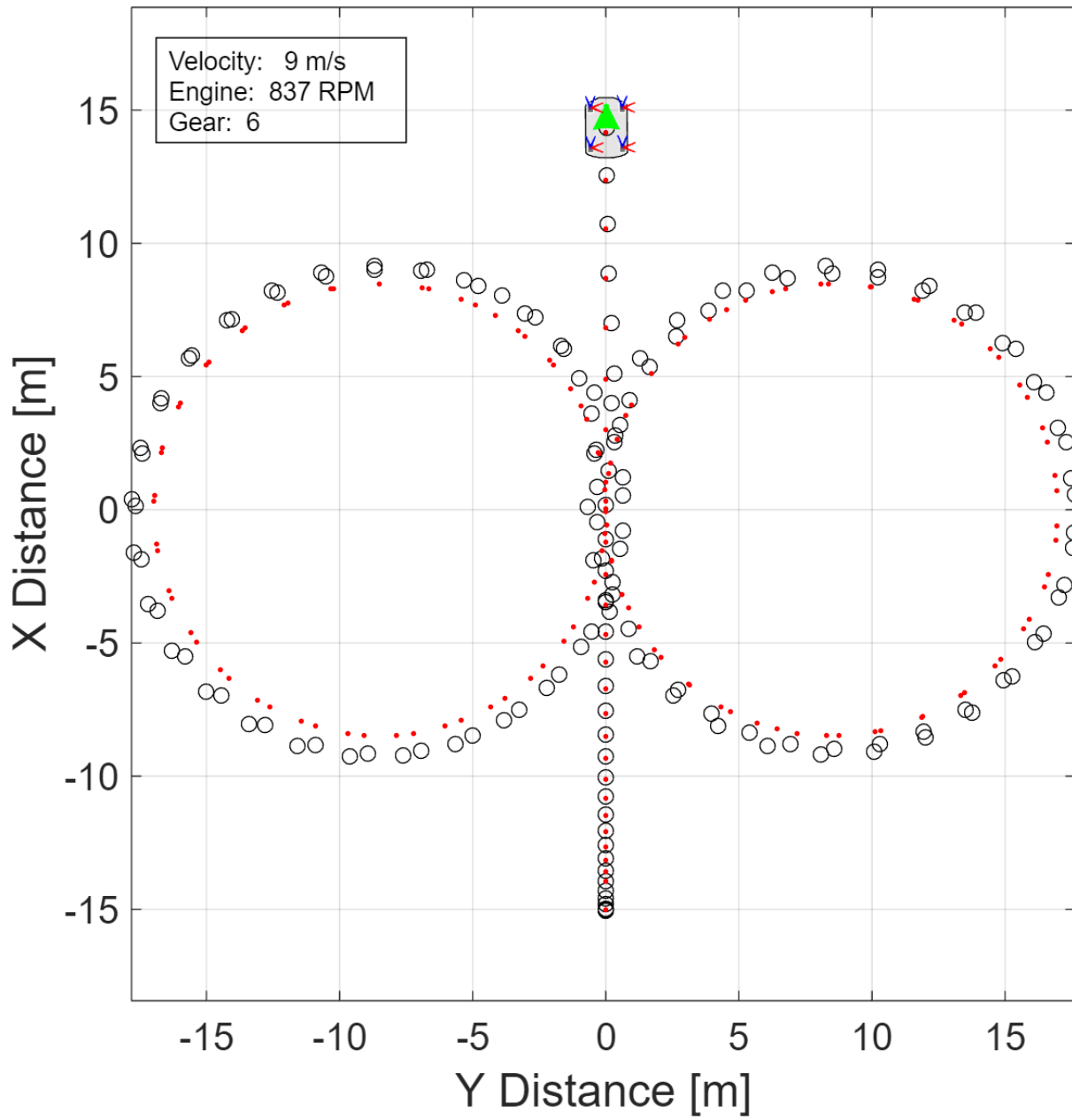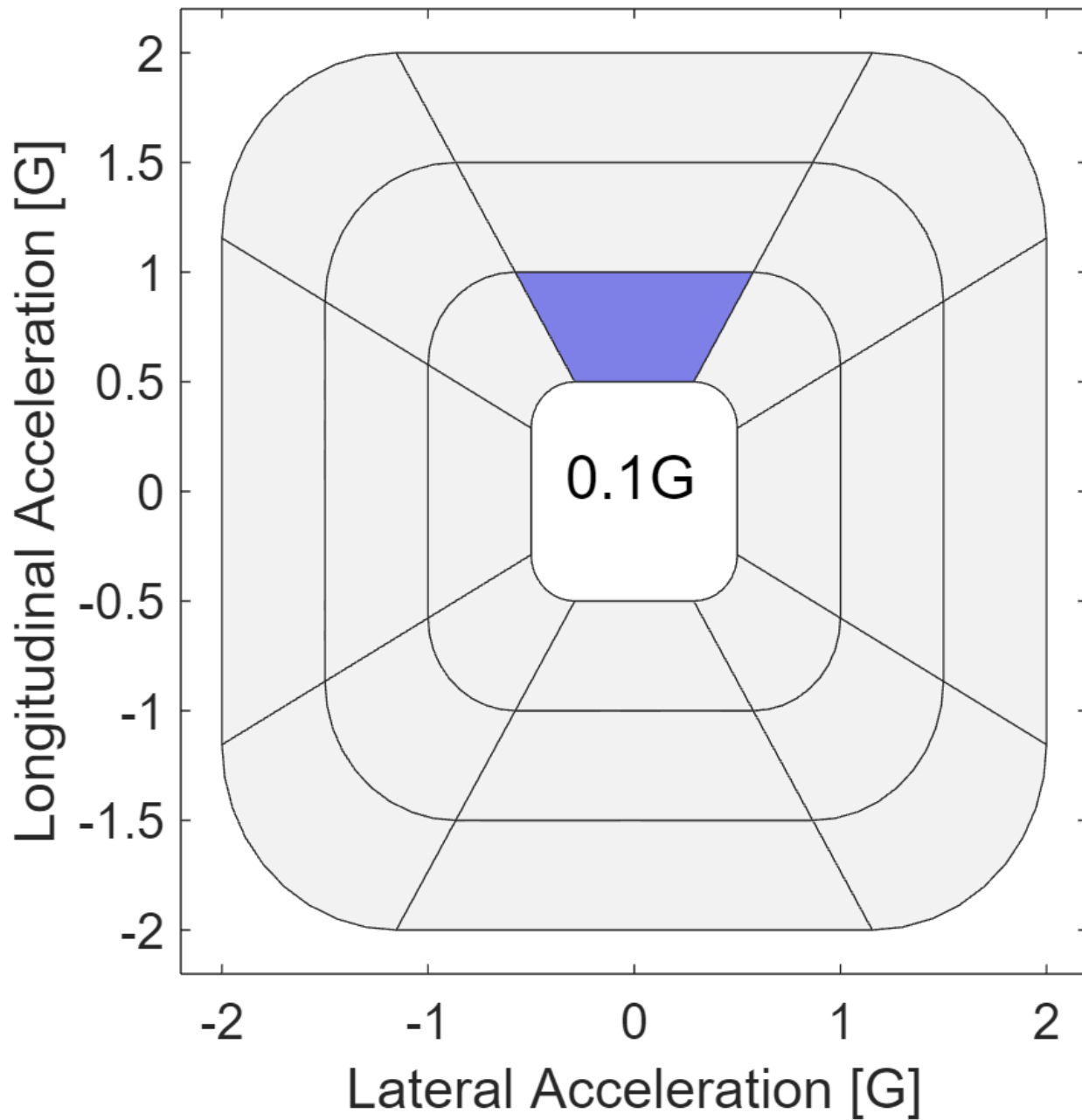
You can also place cones with Simulink 3D Animation to better define the skid pad course when using the 3D visualization engine. You can adjust cone placement through the initialization script in the Simulation 3D Actor block.
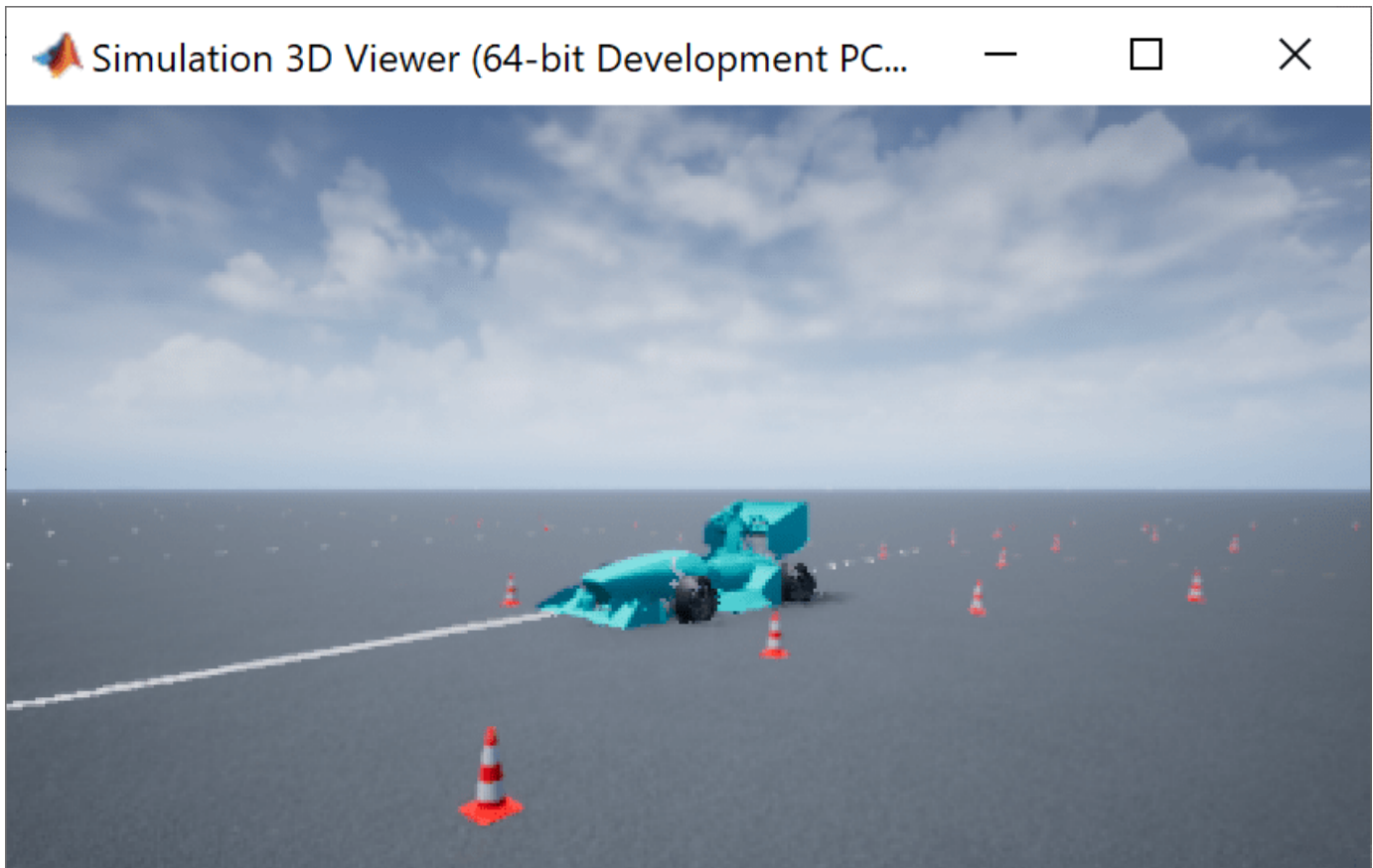
See Also
Simulation 3D Physics Vehicle | Simulation 3D Actor

Related Examples
- "Follow Waypoints Around Oval Track" on page 7-37

3D Simulation

3D Simulation for Vehicle Dynamics Blockset

Vehicle Dynamics Blockset provides a co-simulation framework that models driving algorithms in Simulink and visualizes their performance in a 3D environment. This 3D simulation environment uses the Unreal Engine from Epic Games.
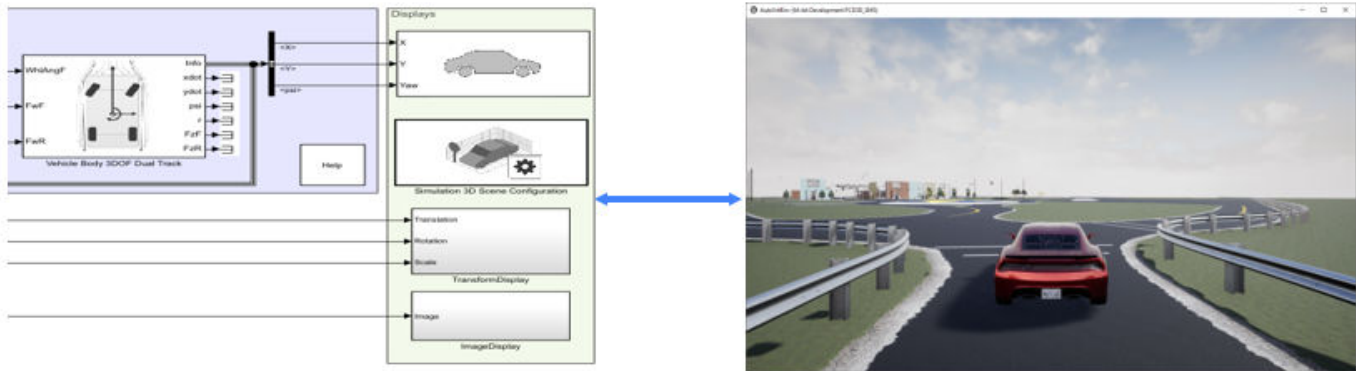
Simulink blocks related to the 3D simulation environment can be found in the **Vehicle Dynamics Blockset > Vehicle Scenarios > Sim3D** block library. These blocks provide the ability to:

- Configure prebuilt scenes in the 3D simulation environment.
- Place and move vehicles within these scenes.
- Set up cameras the vehicles.
- Simulate camera outputs based on the environment around the vehicle.

This simulation tool is commonly used to supplement real data when developing, testing, and verifying the vehicle performance of automated driving algorithms. In conjunction with a vehicle model, you can use these blocks to perform realistic closed-loop simulations that encompass the entire automated driving stack, from perception to control.

For more details on the simulation environment, see "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8.

3D Simulation Blocks

Scenes

To configure a model to co-simulate with the 3D simulation environment, add a Simulation 3D Scene Configuration block to the model. Using this block, you can choose from a set of prebuilt 3D scenes where you can test and visualize your vehicle performance. The following image is from the Virtual Mcity scene.

The toolbox includes these scenes.

Scene	Description
Straight Road	Straight road segment
Curved Road	Curved, looped road
Parking Lot	Empty parking lot
Double Lane Change	Straight road with barrels and traffic signs that are set up for executing a double lane change maneuver
Open Surface	Flat, black pavement surface with no road objects
US City Block	City block with intersections, barriers, and traffic lights
US Highway	Highway with cones, barriers, traffic lights, and traffic signs
Large Parking Lot	Parking lot with parked cars, cones, curbs, and traffic signs
Virtual Mcity	City environment that represents the University of Michigan proving grounds (see Mcity Test Facility); includes cones, barriers, an animal, traffic lights, and traffic signs

If you have the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package, then you can modify these scenes or create new ones. For more details, see "Customize 3D Scenes for Vehicle Dynamics Simulations" on page 6-8.

Vehicles, Tractors, and Trailers

To define a virtual vehicle in a scene, add a Simulation 3D Vehicle with Ground Following, Simulation 3D Vehicle, Simulation 3D Tractor, or Simulation 3D Trailer block to your model. Using the blocks, you can control the movement of the vehicle by supplying the X, Y, and yaw values that define its position and orientation at each time step.

You can also specify the color and type of vehicle. The toolbox includes these vehicle types:

- **Box Truck**
- **Hatchback**
- **Muscle Car**
- **Sedan**
- **Small Pickup Truck**
- **Sport Utility Vehicle**
- **Conventional Tractor**
- **Two-Axle Trailer**
- **Three-Axle Trailer**

Communication

You can define virtual sensors and attach them at various positions on the vehicles. The toolbox includes these sensor modeling and configuration blocks.

Block	Description
Simulation 3D Camera Get	Provides an interface to an ideal camera in the 3D visualization environment. The image output is a red, green, and blue (RGB) array.
Simulation 3D Actor Transform Get	Gets the actor translation, rotation, and scale for the Simulink simulation environment.
Simulation 3D Actor Transform Set	Sets the actor translation, rotation, and scale in the Unreal Engine 3D visualization environment
Simulation 3D Message Get	Retrieves data from the Unreal Engine 3D visualization environment.
Simulation 3D Message Set	Sends data to the Unreal Engine 3D visualization environment.

Algorithm Testing and Visualization

Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects 3D simulation blocks provide the tools for testing and visualizing path planning, vehicle control, and perception algorithms.

Closed-Loop Systems

After you design and test a perception system within the 3D simulation environment, you can then use it to drive a control system that actually steers a vehicle. In this case, rather than manually set up a trajectory, the vehicle uses the perception system to drive itself. By combining perception and control into a closed-loop system in the 3D simulation environment, you can develop and test more complex algorithms, such as lane keeping assist and adaptive cruise control.

See Also

More About

- "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25
- "How 3D Simulation for Vehicle Dynamics Blockset Works" on page 8-8

Unreal Engine Simulation Environment Requirements and Limitations

Vehicle Dynamics Blockset provides an interface to a simulation environment that is visualized using the Unreal Engine from Epic Games. This visualization engine comes installed with the toolbox. When simulating in this environment, keep these requirements and limitations in mind.

Software Requirements

The table summarizes the software requirements for Windows.

Software	Windows
Operating system	Windows 64-bit
Integrated development environment (IDE)	Visual Studio
Graphics driver	Microsoft® DirectX® — If this software is not already installed on your machine and you try to simulate in the environment, the toolbox prompts you to install it. Once you install the software, you must restart the simulation.

In you are customizing scenes, verify that Visual Studio and your Unreal Engine project is compatible with the Unreal Engine version supported by your MATLAB release.

MATLAB Release	Unreal Engine Version	Visual Studio Version
R2018a–R2019b	4.19	2017
R2020a–R2021a	4.23	2019
R2021b	4.25	2019
R2022a–R2022b	4.26	2019
R2023a	4.27	2019

Recommended Hardware Requirements

The table summarizes the recommended hardware requirements for Windows.

Hardware	Windows
Graphics card (GPU)	Virtual reality-ready
Video memory (RAM)	8 GB
Processor (CPU)	2.60 GHz
Processor memory (RAM)	32 GB

Limitations

The Simulation 3D blocks do not support:

- Code generation

- Model reference
- Multiple instances of the Simulation 3D Scene Configuration block
- Multiple Unreal Engine instances in the same MATLAB session
- Rapid accelerator mode
- Multiple instances of the same actor tag. To refer to the same scene actor when you use the 3D block pairs, such as Simulation 3D Actor Transform Get and Simulation 3D Actor Transform Set, specify the same **Tag for actor in 3D scene, Actortag** parameter.
- Simulink Online™ simulation.

You cannot create or use `sim3d` objects or functions in MATLAB Online.

In addition, when using these blocks in a closed-loop simulation, all Simulation 3D blocks must be in the same subsystem.

See Also

More About
- "Vehicle Scenarios"

External Websites
- Unreal Engine 4 Documentation

How 3D Simulation for Vehicle Dynamics Blockset Works

The vehicle dynamics models run programmable maneuvers in a photorealistic 3D visualization environment. Vehicle Dynamics Blockset integrates the 3D simulation environment with Simulink so that you can query the world around the vehicle for virtually testing perception, control, and planning algorithms. The Vehicle Dynamics Blockset visualization environment uses the Unreal Engine by Epic Games.

Understanding how this simulation environment works can help you troubleshoot issues and customize your models.

Communication with 3D Simulation Environment

When you use Vehicle Dynamics Blockset to run your algorithms, Simulink co-simulates the algorithms in the visualization engine.

In the Simulink environment, Vehicle Dynamics Blockset:

- Determines the next position of objects by using 3D visualization environment feedback and vehicle dynamics models.
- Configures the 3D visualization environment, specifically:

  - Ray tracing
  - Scene capture cameras
  - Initial object positions

In the visualization engine environment, Vehicle Dynamics Blockset positions the objects and uses ray tracing to query the environment.

The diagram summarizes the communication between Simulink and the visualization engine.
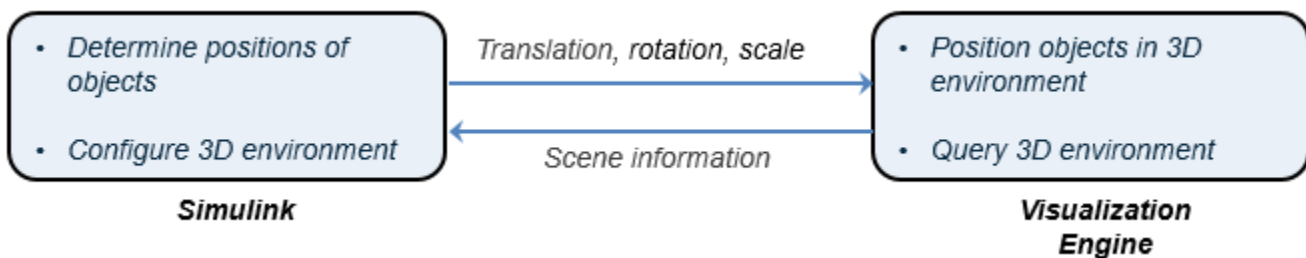
Block Execution Order

During simulation, the 3D simulation blocks follow a specific execution order:

1 The vehicle blocks initialize the vehicles and send their **X**, **Y**, and **Yaw** signal data to the Simulation 3D Scene Configuration block.
2 The Simulation 3D Scene Configuration block receives the vehicle data and sends it to the sensor blocks.
3 The sensor blocks receive the vehicle data and use it to accurately locate and visualize the vehicles.

The **Priority** property of the blocks controls this execution order. To access this property for any block, right-click the block, select **Properties**, and click the **General** tab. By default, Simulation 3D Vehicle with Ground Following blocks have a priority of `-1`, Simulation 3D Scene Configuration blocks have a priority of `0`, and sensor blocks have a priority of `1`.

If your sensors are not detecting vehicles in the scene, it is possible that the 3D simulation blocks are executing out of order. Try updating the execution order and simulating again. For more details on execution order, see "Control and Display Execution Order".

Also be sure that all 3D simulation blocks are located in the same subsystem. Even if the blocks have the correct **Priority** settings, if they are located in different subsystems, they still might execute out of order.

See Also

Related Examples
- "Send and Receive Double-Lane Change Scene Data" on page 3-93

More About
- "Unreal Engine Simulation Environment Requirements and Limitations" on page 8-6
- "Scene Interrogation in 3D Environment" on page 3-34

External Websites
- Unreal Engine

Place Cameras on Actors in the Unreal Editor

To visualize objects in an Unreal Editor scene, you can place cameras on static or custom actors in the scene. To start, you need the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. See "Install Support Package and Configure Environment" on page 6-10.

To follow this workflow, you should be comfortable using Unreal Engine. Make sure that you have Visual Studio 2019 installed on your computer.

Place Camera on Static Actor

Follow these steps to place a Simulation 3D Camera Get block that is offset from a cone in the Unreal Editor. Although this example uses the To Video Display block from Computer Vision Toolbox™, you can use a different visualization block to display the image.

1   In a Simulink model, add the Simulation 3D Scene Configuration, Simulation 3D Camera Get, and To Video Display blocks.

Simulation 3D Scene Configuration

Simulation 3D Camera Get
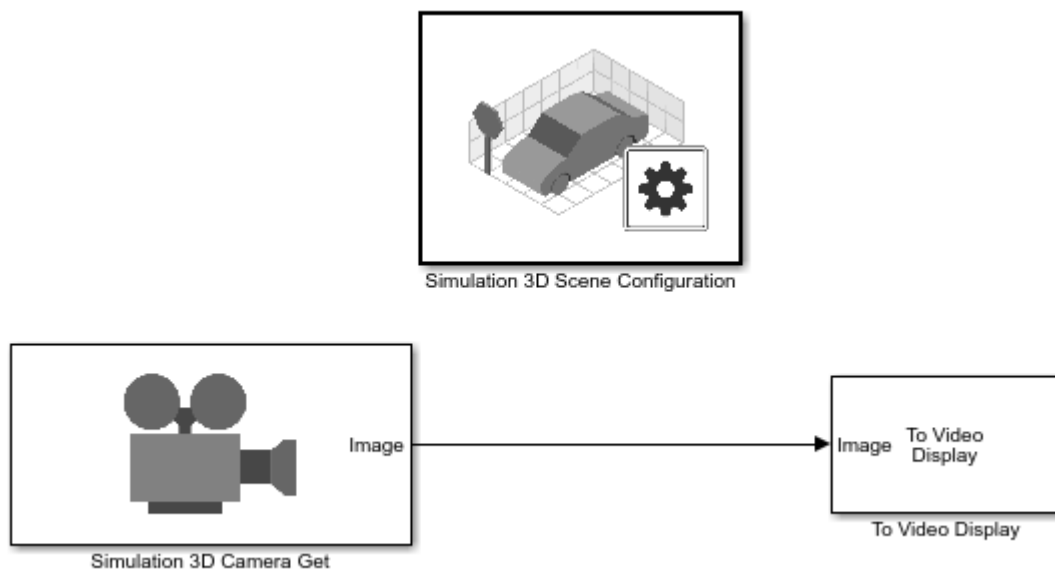
Set these block parameters. In the Simulation 3D Scene Configuration block, select **Open Unreal Editor**.

Block	Parameter Settings
Simulation 3D Scene Configuration	• **Scene Source** — `Unreal Editor` • **Project** — Specify the path and name of the support package project file. For example, `C:\Local\AutoVrtlEnv\AutoVrtlEnv.uproject`

Block	Parameter Settings
Simulation 3D Camera Get	• **Sensor identifier** — 1 • **Vehicle name** — Scene Origin • **Vehicle mounting location** — Origin • **Specify offset** — on • **Relative translation [X, Y, Z]** — [-6, 0, 2] This offsets the camera location from the cone mounting location, 6 m behind, and 2 m up. • **Relative rotation [Roll, Pitch, Yaw]** — [0, 15, 0]

2 In the Unreal Editor, from the **Place Actors** tab, add a **Sim 3d Scene Cap** to the world, scene, or map.

3 In the Unreal Editor, from the **Place Actors** tab, add a **Cone** to the world, scene, or map.

4 On the **World Outliner tab**, right-click the **Sim3DSceneCap1** and attach it to the **Cone**.

5 On the **Details** tab, under **Transform**, add a location offset of -500,0,100 in the X, Y, and Z world coordinate system, respectively. This attaches the camera 500 cm behind the cone and 100 cm above it. The values match the Simulation 3D Camera Get block parameter **Relative translation [X, Y, Z]** value.
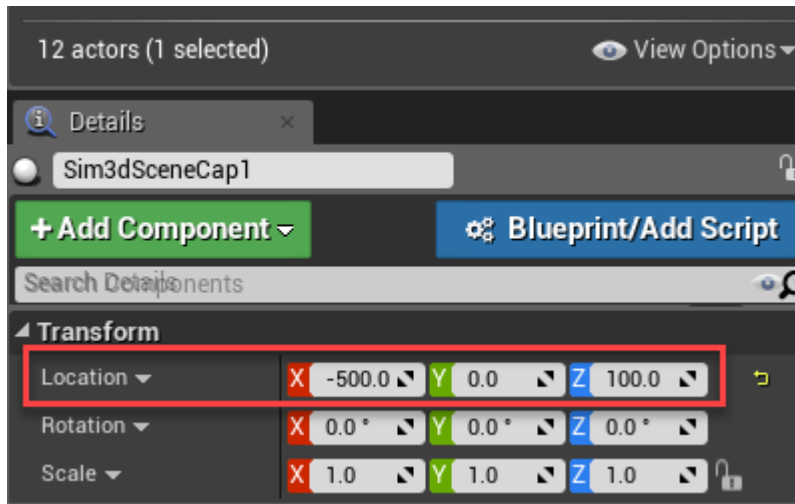
6   On the **Details** tab, under **Actor**, tag the **Sim3DSceneCap1** with the name Camera1.
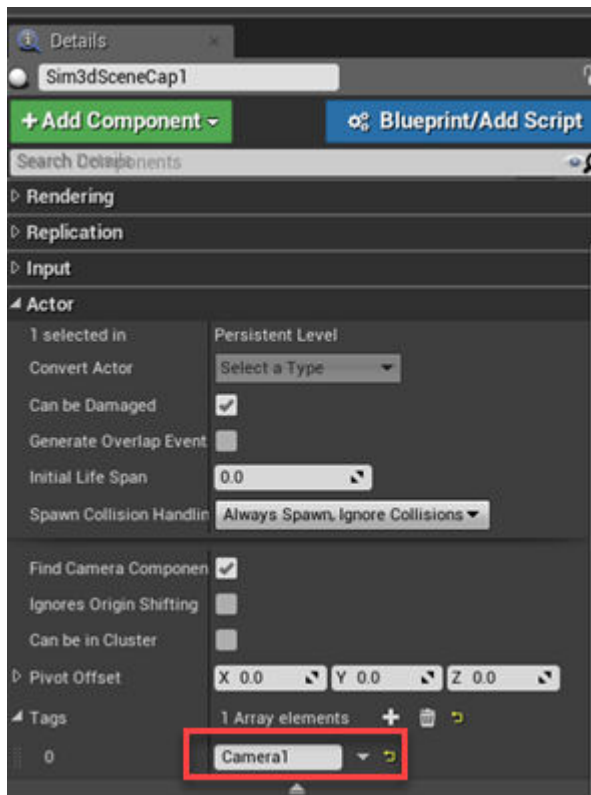
7   Run the simulation.

   a   In the Simulink model, click **Run**.

         Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start.

   b   Verify that the Diagnostic Viewer window in Simulink displays this message:

> In the Simulation 3D Scene Configuration block, you set the scene source to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the scene.

This message confirms that Simulink has instantiated the vehicles and other assets in the Unreal Engine 3D environment.

c In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor.

Observe the results in the To Video display window. The window displays the image from the camera.

Place Camera on Vehicle in Custom Project

Follow these steps to create a custom Unreal Engine project and place a camera on a vehicle in the project. Although the example uses the To Video Display block from Computer Vision Toolbox, you can use a different visualization block to display the image.

To start, you need the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package. See "Install Support Package and Configure Environment" on page 6-10.

1 In a Simulink model, add the Simulation 3D Scene Configuration, Simulation 3D Camera Get, To Video Display, Simulation 3D Actor Transform Set, and three Constant blocks.

Simulation 3D Scene Configuration

Simulation 3D Camera Get

To Video Display

ActTranslation

ActRotation

ActScale

Simulation 3D Actor Transform Set
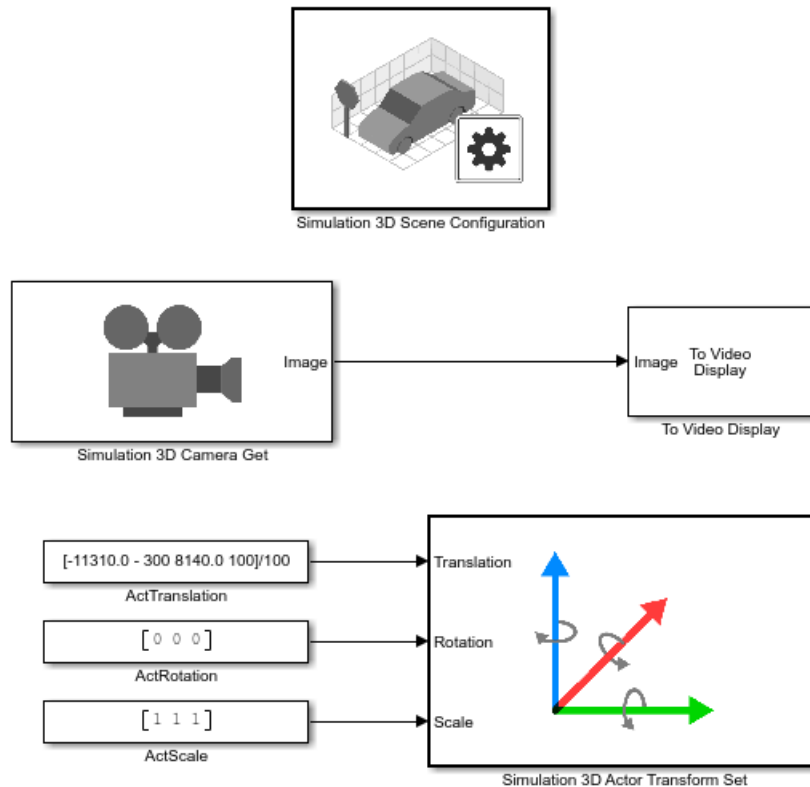
Save the model.

2   Create a new project using the **Vehicle Advanced** template from the Epic Games Launcher by Epic Games.

   a   In the Epic Games Launcher, launch Unreal Engine 4.27.
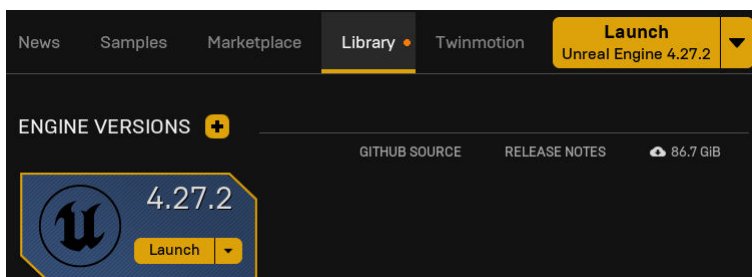
   For more information about the Epic Games Launcher, see Unreal Engine.

   b   In the Unreal Project Browser, select **Games** and **Next**.
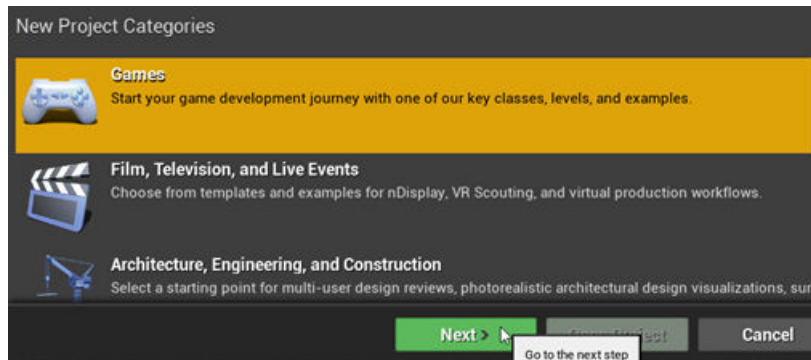
c In **Select Template**, select the **Vehicle Advanced** template and click **Next**.

d In **Project Settings**, create a Blueprint or C++ project, and select a project name and location. Click **Create Project**.
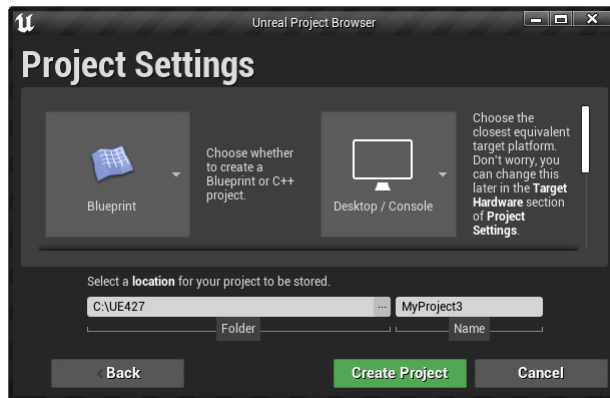
The Epic Games Launcher creates a new project and opens the Unreal Editor.

   e   Enable the MathWorks Interface plugin.

      i   Select **Edit** > **Plugins**.

      ii On the **Plugins** tab, navigate to MathWorks Interface. Select **Enabled**.

   f   Save the project. Close the Unreal Editor.

3   Open the Simulink model that you saved in step 1. Set these block parameters.

Block	Parameter Settings
Simulation 3D Scene Configuration	• **Scene Source** — `Unreal Editor` • **Project** — Specify the path an project that you saved in step 2. For example, *myProjectPath*`\`*myProject*`.uproject`
Simulation 3D Camera Get	• **Sensor identifier** — `1` • **Vehicle name** — `Scene Origin` • **Vehicle mounting location** — `Origin`
Simulation 3D Actor Transform Set	• **Tag for actor in 3D scene** — `MainCamera1`
ActTranslation	• **Constant value** — `[-11310.0 - 300, 8140.0, 100]/100` • **Interpret vector parameters as 1-D** — `off`

Block	Parameter Settings
ActRotation	• **Constant value** — `[0 0 0]` • **Interpret vector parameters as 1-D** — `off`
ActScale	• **Constant value** — `[1 1 1]` • **Interpret vector parameters as 1-D** — `off`

4 In the Simulation 3D Scene Configuration block, select **Open Unreal Editor**.

5 In the Unreal Editor, in the **Content Browser** navigate to `Sim3DCamera`. Add it to the world, scene, or map.
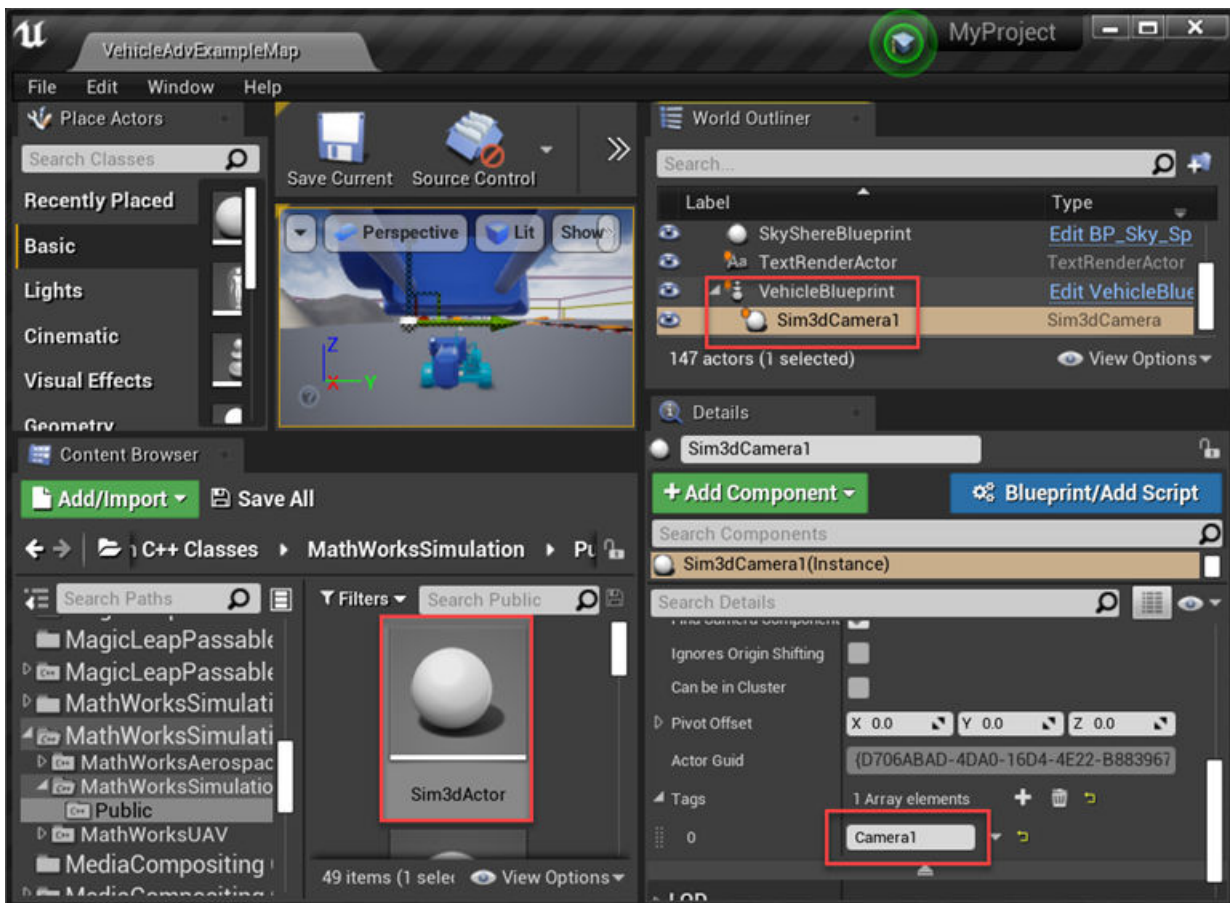
6 On the vehicle `VehicleBlueprint`, drag and drop the camera. Choose a vehicle socket or bone to attach the camera to.
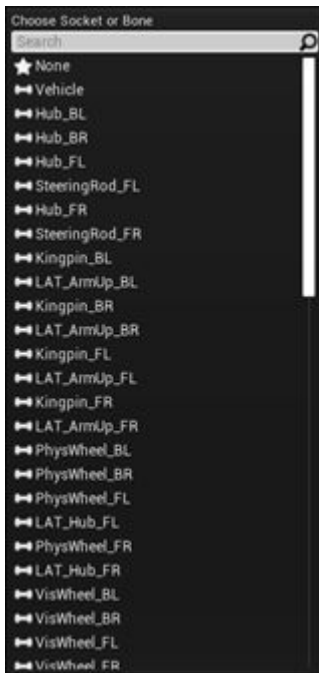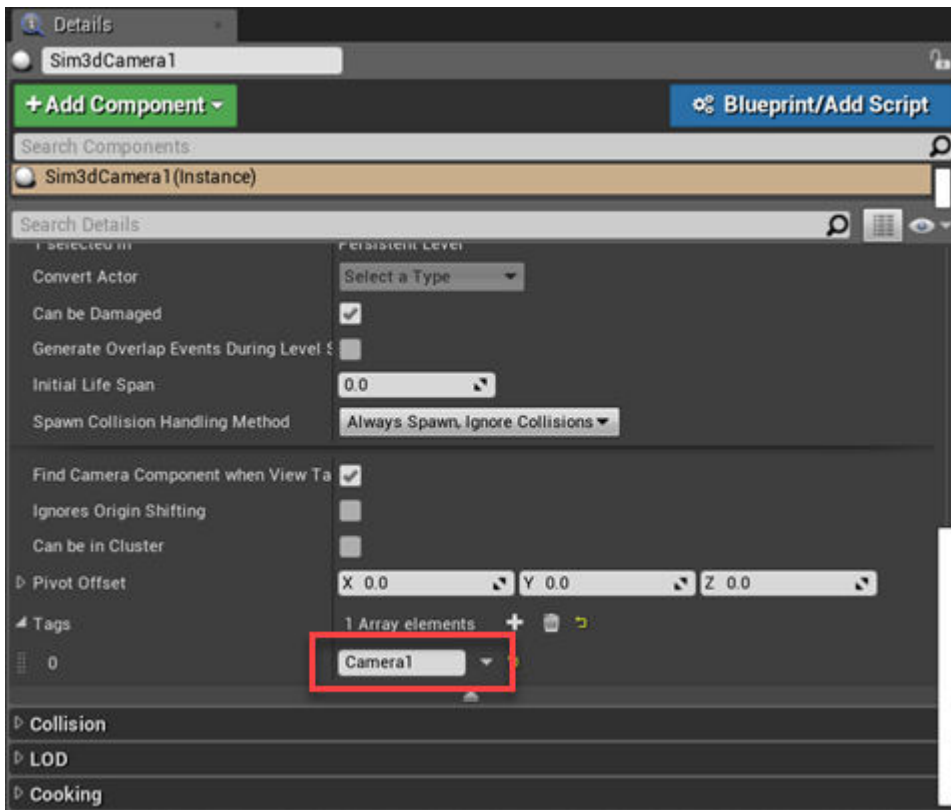
---
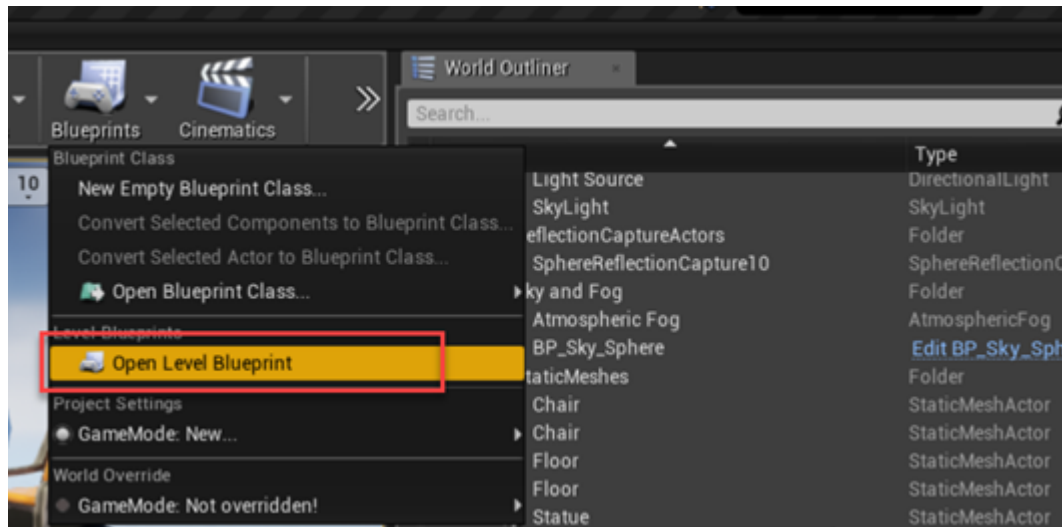
b In the **Class Options**, set **Parent Class** to `Sim 3d Level Script Actor`.
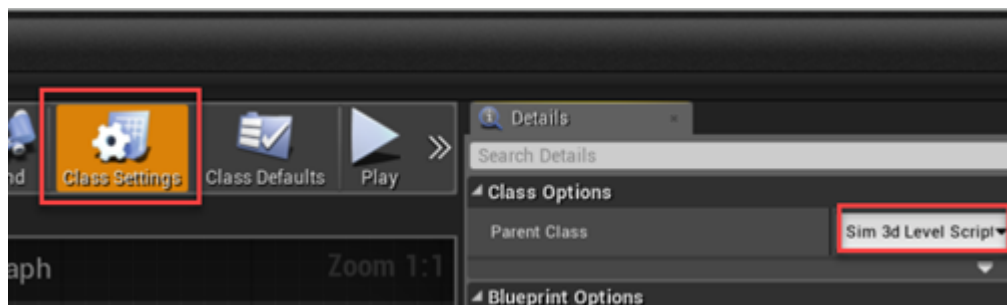
9 Save the project.

10 Run the simulation.

    a In the Simulink model, click **Run**.

    Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start.

    b Verify that the Diagnostic Viewer window in Simulink displays this message:

```
In the Simulation 3D Scene Configuration block, you set the scene
source to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the
scene.
```

    This message confirms that Simulink has instantiated the vehicles and other assets in the Unreal Engine 3D environment.

    c In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor.

Observe the results in the To Video Display window.

See Also

Simulation 3D Camera Get | Simulation 3D Scene Configuration

More About

- "Create Empty Project in Unreal Engine" on page 6-49
- "Animate Custom Actors in the Unreal Editor" on page 8-21
- "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25

External Websites

- Unreal Engine

Animate Custom Actors in the Unreal Editor

Follow these steps to animate a custom actor in the Unreal Editor. Before you start, make sure you that you have Visual Studio 2019 and the Vehicle Dynamics Blockset Interface for Unreal Engine 4 Projects support package installed on your machine. For more information, see "Install Support Package and Configure Environment" on page 6-10.

Additionally, make sure that:

- You are comfortable coding with C++ in Unreal Engine.
- Your Unreal Editor C++ project contains a skeletal actor mesh. This example uses a bicycle mesh.
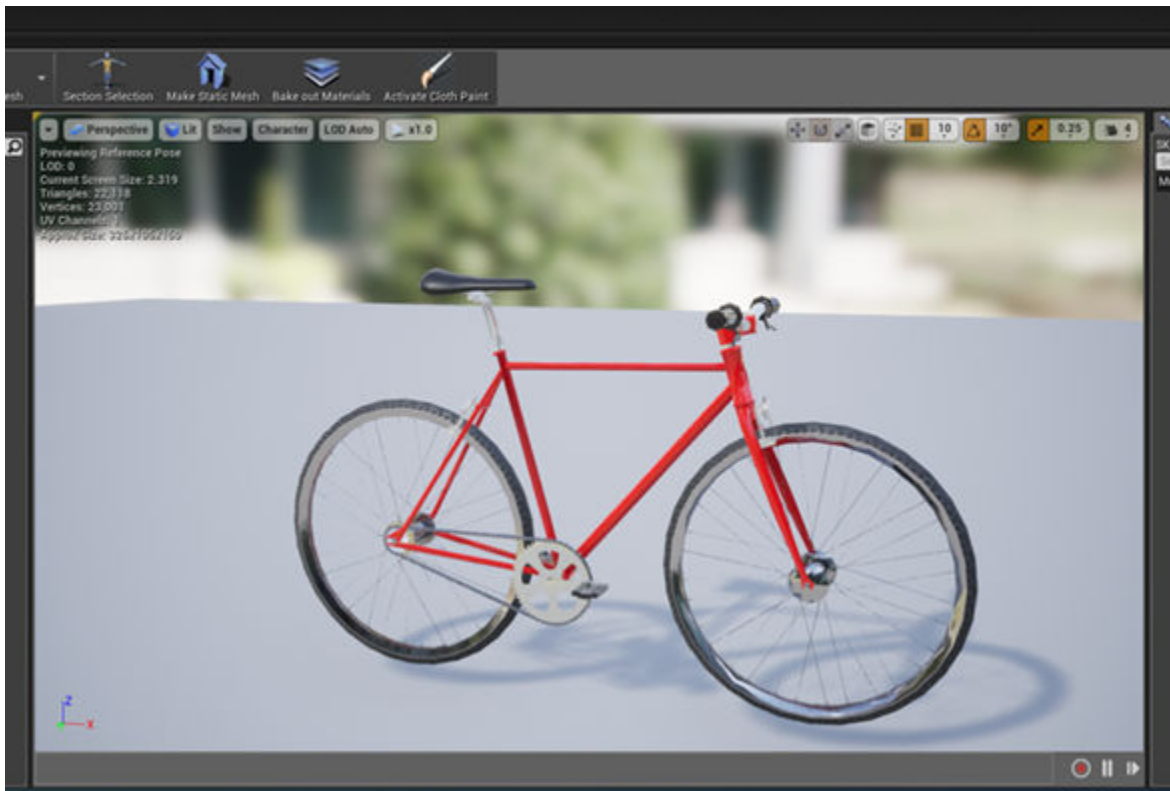
This examples provides the workflow for animating a bicycle actor. The general workflow is adapted from the Unreal Engine *Vehicle User Guide*.

Set up Simulink Model

Step 1: Set up Simulink Model

Open a new Simulink model and add these blocks:

- Two Ramp blocks
- Constant block
- Simulation 3D Actor Transform Set block
- Simulation 3D Scene Configuration block

Connect and name the blocks as shown.
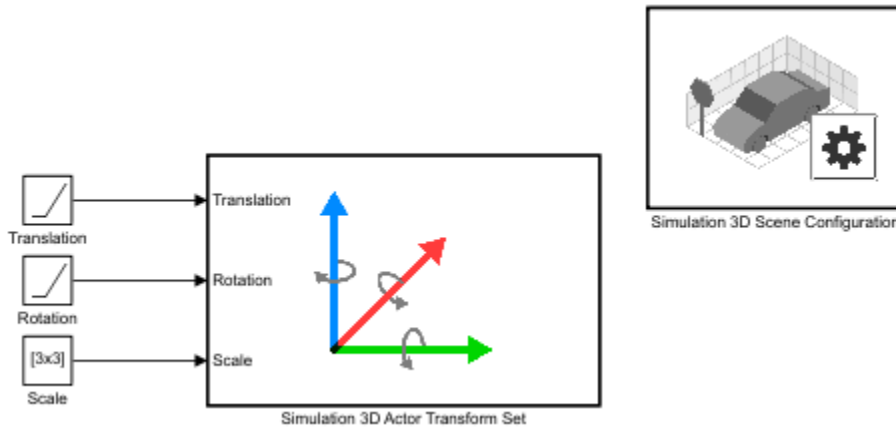
Step 2: Configure Blocks

Configure blocks with these parameter settings.

Block	Parameter Settings
Simulation 3D Scene Configuration	• **Scene source** — `Unreal Editor` • **Project** — Name and location of the installed support package project file, for example, `C:\Local\AutoVrtlEnv \AutoVrtlEnv.uproject`. • **Scene view** — `Scene Origin`

Simulation 3D Actor Transform Set	• **Actor Setup** tab:
	  • **Tag for actor in 3D scene, ActorTag** — `Bike1`
	
	  Note This tag should match the Unreal Editor tag name in "Step 6: Instantiate the Bicycle Actor" on page 8-35.
	  • **Number of parts per actor to set, NumberOfParts** — 3
	• **Initial Values** tab:
	  • **Initial array values to translate actor per part, Translation** — `[0 0 0;0 0 0;0 0 0]`
	  • **Initial array values to rotate actor per part, Rotation** — `[0 0 0;0 0 0;0 0 0]`
	  • **Initial array values to scale actor per part, Scale** — `[1 1 1;1 1 1;1 1 1]`

Translation Ramp	• **Slope** — `[0.35 0 0;0 0 0;0 0 0]`
Rotation Ramp	• **Slope** — `[0 0 0;0 -pi/5 0;0 -pi/5 0]`
Scale Constant	• **Constant value** — `[1 1 1;1 1 1;1 1 1]`

Set up Unreal Editor to Animate Bicycle

Step 3: Set up Animation Instance

1 In your Simulink model, use the Simulation 3D Scene Configuration block **Open Unreal Editor** parameter to open the Unreal Editor.

2 Select **File > New C++ Class**. In the **Choose Parent Class** dialog box, select **Show All Classes**. Search for `AnimInst`. Add the `AnimInstance` parent class.
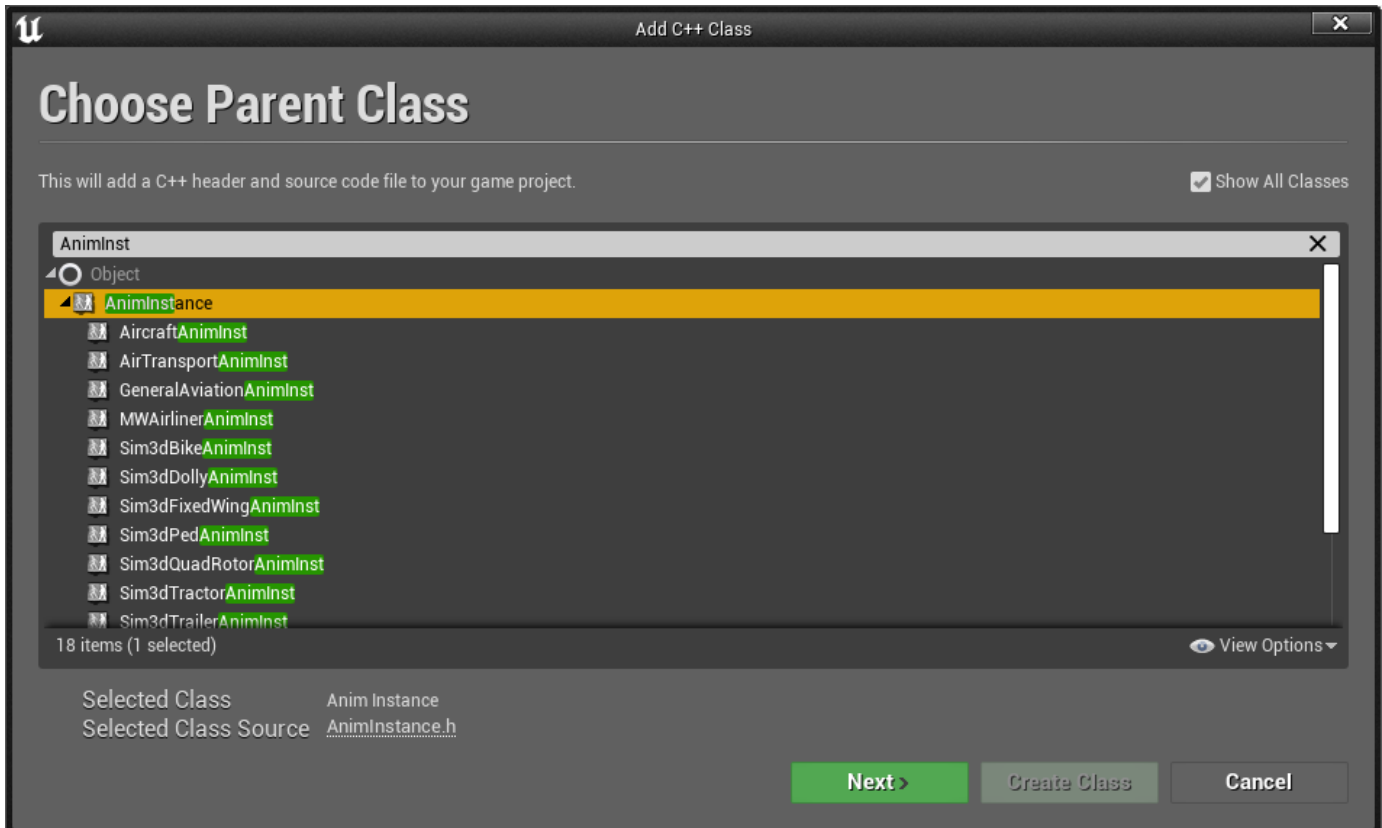
3 Name the new C++ class `SimulinkBikeAnimInst`. Select **Public**. Click **Create Class**.

4 In Visual Studio 2019, open the `C:\Local\AutoVrtlEnv\AutoVrtlEnv.sln` file. Navigate to the `SimulinkBikeAnimInst.cpp` and `SimulinkBikeAnimInst.h` source files.
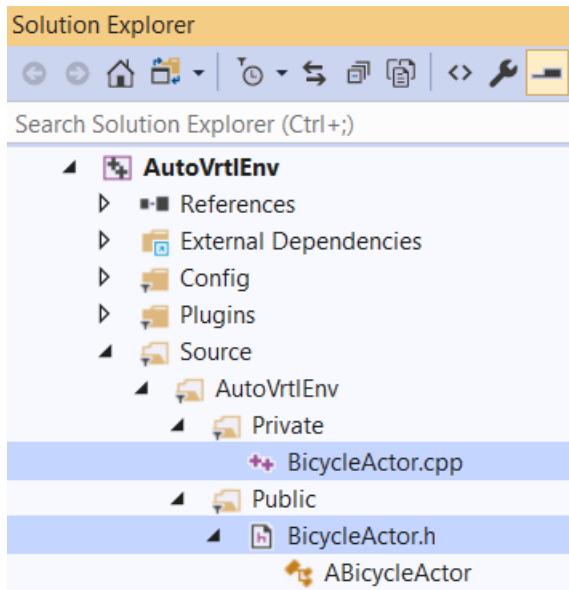
Edit the files as shown.

Tip For this example, the code includes `FWheelRotation` and `RWheelRotation` properties to animate the bicycle wheel rotation. You can add additional properties to animate other parts of the bicycle.

Code: `SimulinkBikeAnimInst.h`

```cpp
// Copyright 2019 The MathWorks, Inc.
#pragma once

#include "CoreMinimal.h"
#include "Animation/AnimInstance.h"
#include "SimulinkBikeAnimInst.generated.h"

/**
 *
 */
UCLASS(transient, Blueprintable, hideCategories = AnimInstance, BlueprintType)
class AUTOVRTLENV_API USimulinkBikeAnimInst : public UAnimInstance
{
    GENERATED_UCLASS_BODY()
public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = WheelRotation)
        float FWheelRotation;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = WheelRotation)
        float RWheelRotation;
};
```

Code: `SimulinkBikeAnimInst.cpp`

```cpp
// Copyright 2019 The MathWorks, Inc.
#include "SimulinkBikeAnimInst.h"

USimulinkBikeAnimInst::USimulinkBikeAnimInst(const FObjectInitializer& ObjectInitializer)
    : Super(ObjectInitializer) {
    FWheelRotation = 0.0f;
    RWheelRotation = 0.0f;
}
```

5   In the Unreal Editor click **Compile**.
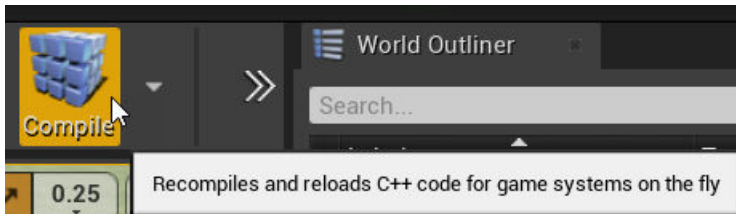
Step 4: Create Animation Blueprint

1   In the Unreal Editor, on the **Content Browser** tab, under **View Options**, select **Show Engine Content** and **Show Plugin Content**.
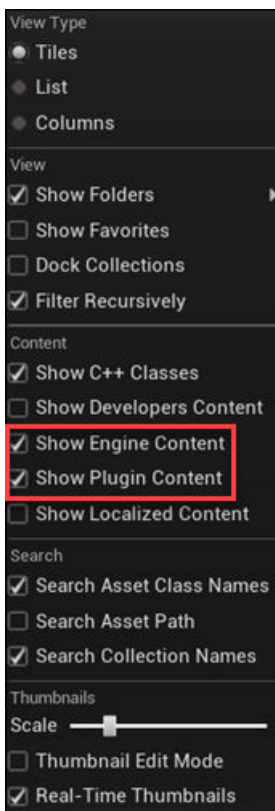
2   Add the animation mesh. On the **Content Browser** tab, navigate to **MathWorksAutomotiveContent Content > Vehicles > Bicyclist > Meshes**.
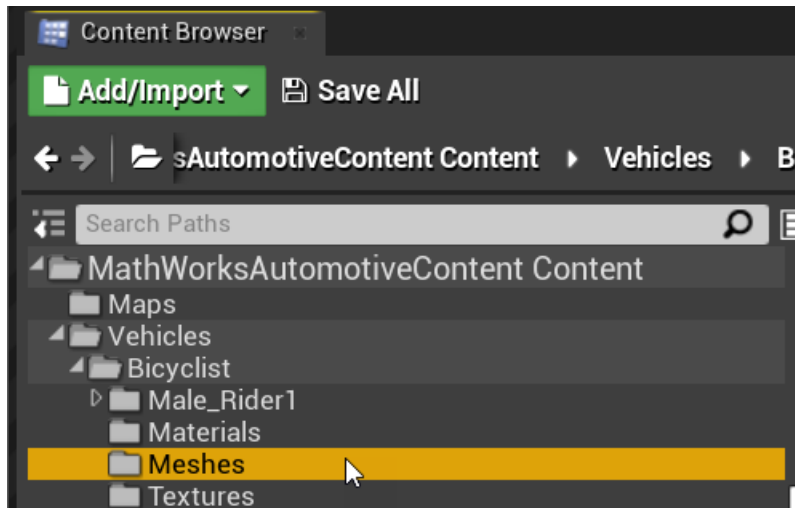
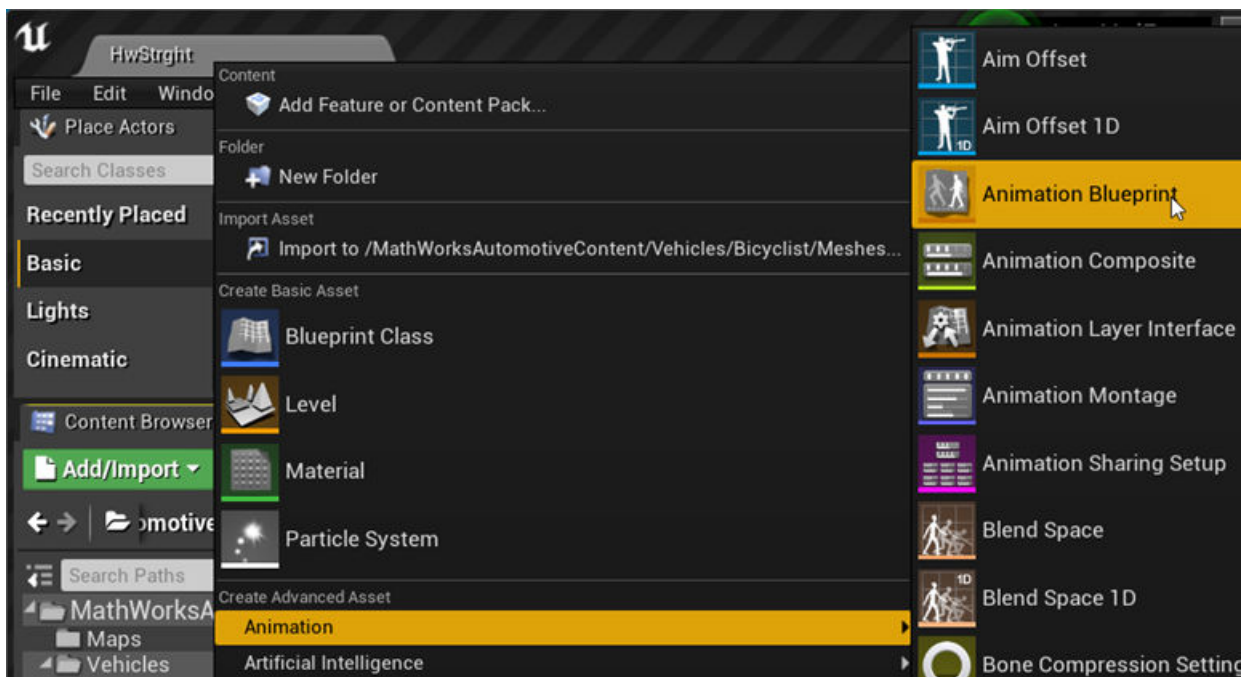**3** Select **Add/Inport > Animation > Animation Blueprint**.
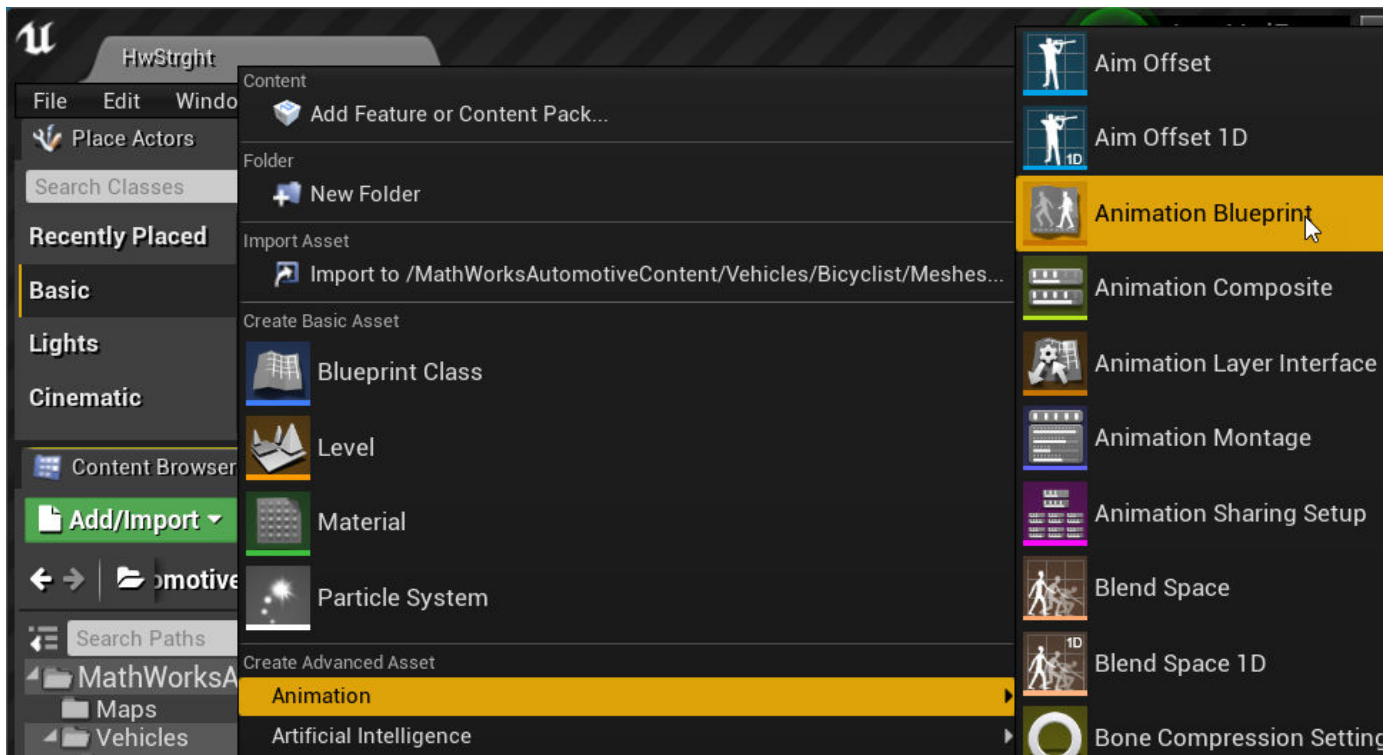
4 In the Create Animation Blueprint dialog box, select:

- **Parent Class**: `SimulinkBikeAnimInst`
- **Target Skeleton**: `SK_Bicycle_Skeleton`
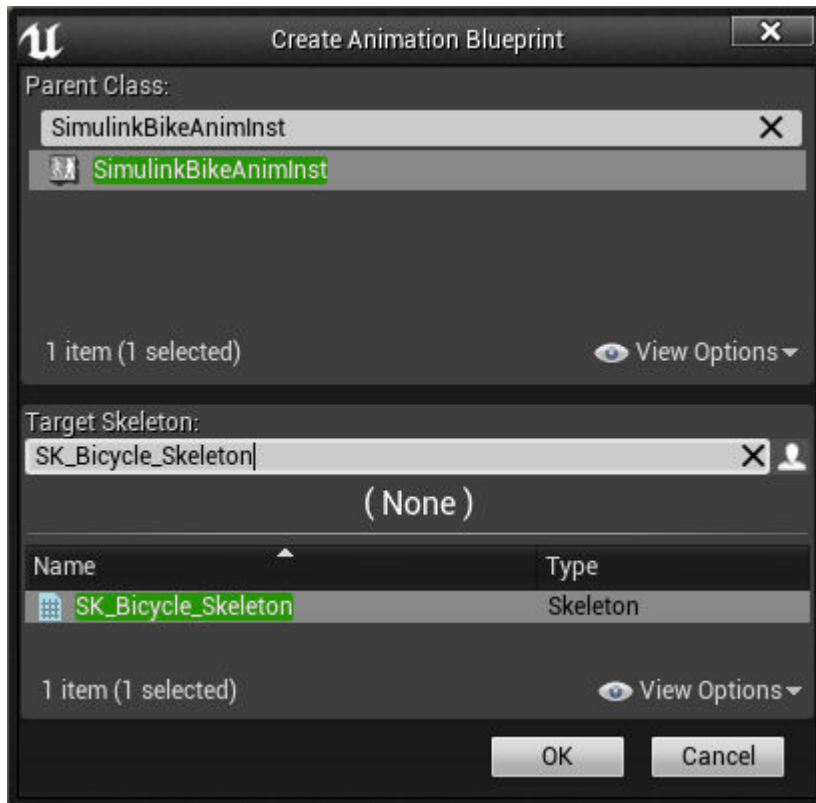
Click **OK**.

5 Name the blueprint `BikeAnimation`. Right-click and select **Save**.

6 Open the `BikeAnimation` blueprint. Make the connections as shown.
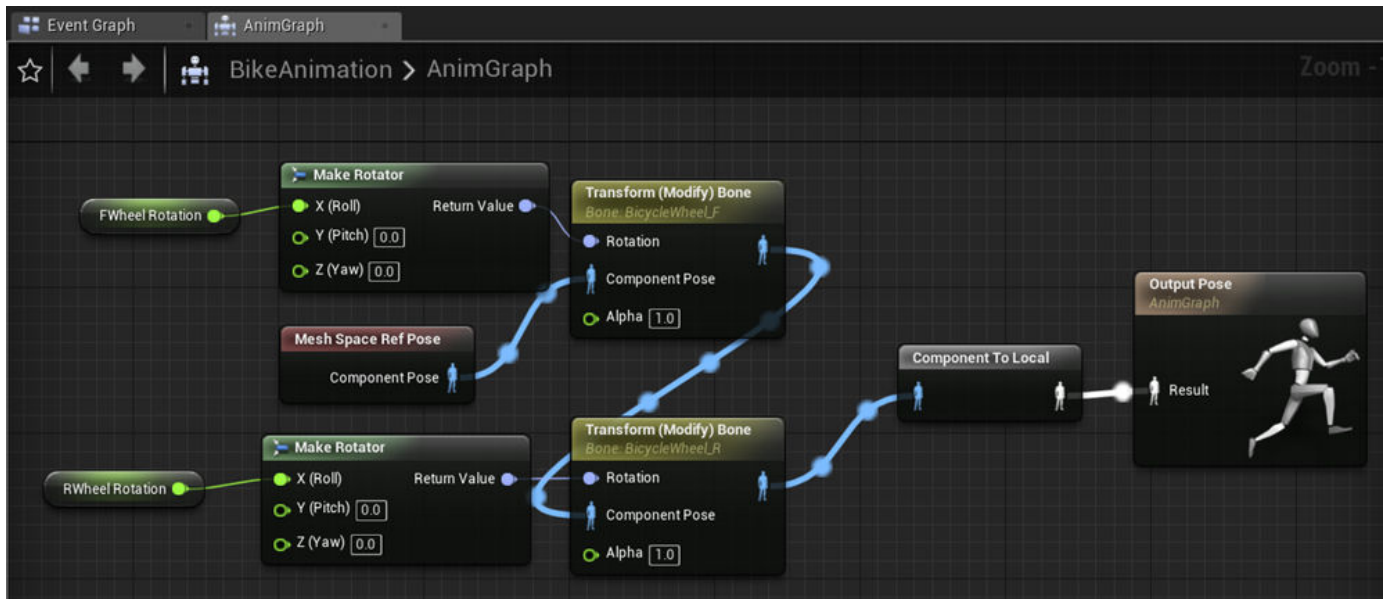
For both front and rear wheels, make sure that you set:

- **Bone to Modify** to the correct bone
- **Rotation Mode** to `Replace Existing`
- **Rotation Space** to `Bone Space`
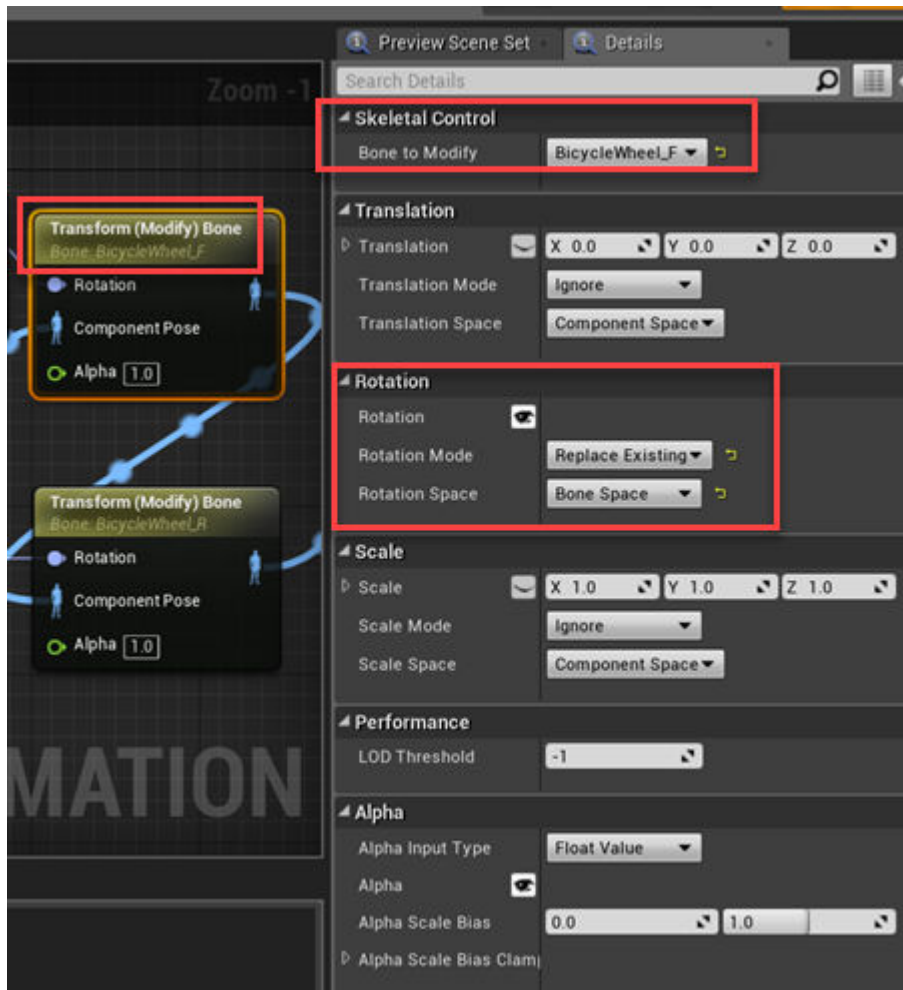
7 Compile and save the blueprint.

Step 5: Create Bicycle Actor C++ Class

1 In the Unreal Editor, on the **Content Browser** tab, under **View Options**, select **Show Engine Content** and **Show Plugin Content**.
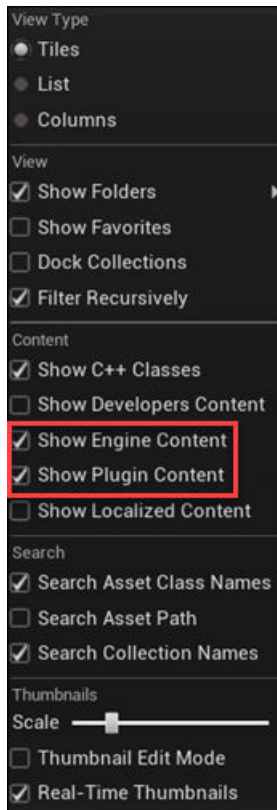
2 From the MathWorksSimulation C++ Classes folder, select **Sim3dActor**.
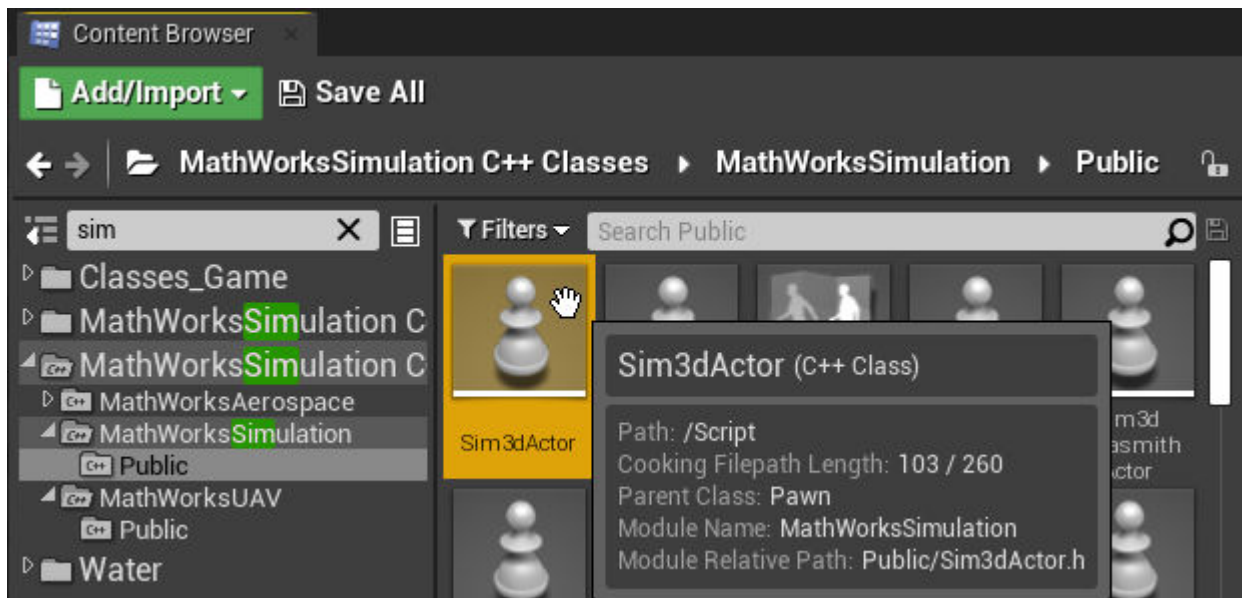
Right-click and select **Create C++ class derived from Sim3dActor**.
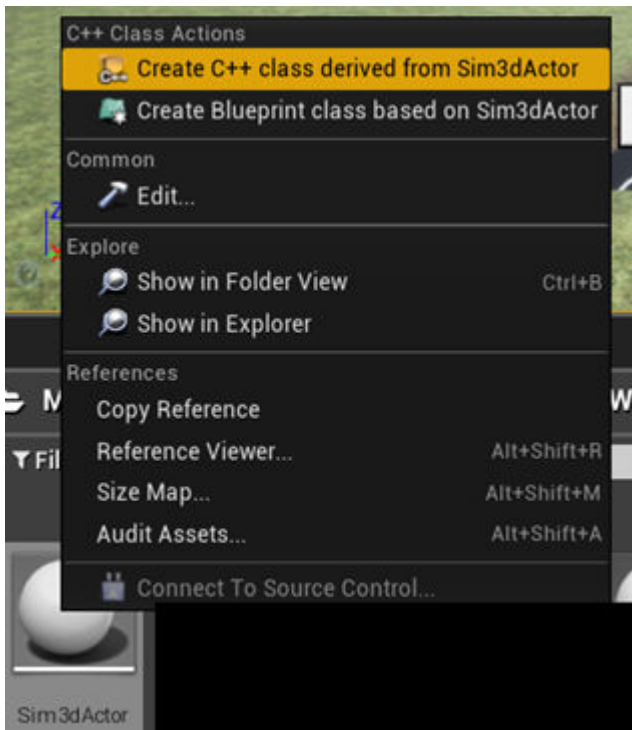
Tip If you do not see the MathWorksSimulation C++ Classes folder, use these steps to check that you have the `MathWorksSimulation` plugin installed and enabled:

a   In the Unreal Editor toolbar, select **Edit > Plugins**.

b   In the Plugins window, verify that the **MathWorks Interface** plugin is listed in the installed window. If the plugin is not already enabled, select the **Enabled** check box.

c   Close the editor and reopen it from Simulink.

3   Name the new Sim3dActor `BicycleActor`. Select **Public**. Click **Create Class**.
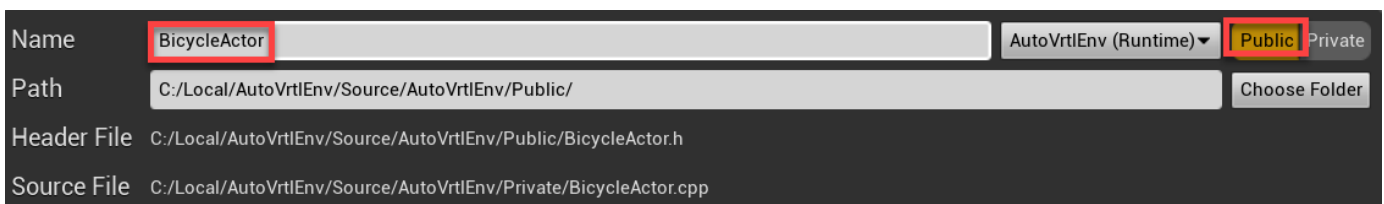
4   In Visual Studio, navigate to `BicycleActor.h` and `BicycleActor.cpp`.
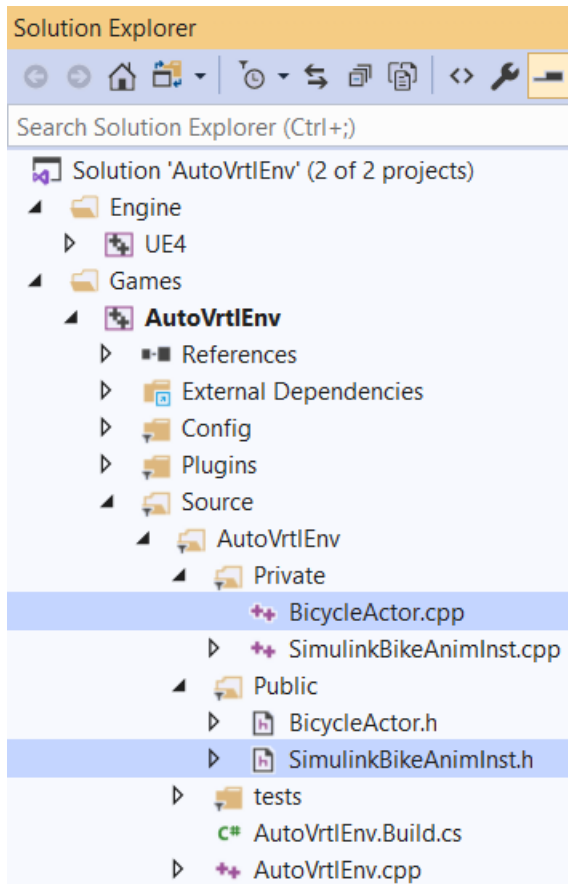
Edit the files as shown.

Tip For this example, the code includes logic to animate the bike body (BIKE_BODY), front wheel (FRONT_WHEEL), and rear wheel (REAR_WHEEL). You can add additional logic to animate other parts of the bicycle.

Code: `BicycleActor.h`

```cpp
// Copyright 2019 The MathWorks, Inc.
#pragma once

#include "CoreMinimal.h"
#include "Sim3dActor.h"
#include "BicycleActor.generated.h"


UCLASS()
class AUTOVRTLENV_API ABicycleActor : public ASim3dActor
{
    GENERATED_BODY()

    // Reference to animation blueprint and skeletal mesh
    UClass* BicycleAnimation;
    USkeletalMesh* BicycleMesh;

    //Enum for parts that we want to control from simulink
    enum {
        BIKE_BODY = 0,
```

```
            FRONT_WHEEL = 1,
            REAR_WHEEL =2,
            NumberOfParts = 3
    };
    enum {
            X = 0,
            Y = 1,
            Z = 2
    };
    enum {
            PITCH = 0,
            ROLL = 1,
            YAW = 2
    };

public:
    //Containers to receive data from Simulink
    float Translation[NumberOfParts][3];
    float Rotation[NumberOfParts][3];
    float Scale[NumberOfParts][3];

    ABicycleActor();

    //Override functions for enabling Simulink to control this actor
    virtual void Sim3dInit() override;
    virtual void Sim3dSetup() override;
    virtual void Sim3dStep(float DeltaSeconds) override;
    virtual void Sim3dRelease() override;

    //Some helper functions
    void SetMesh(FString MeshRef);
    void SetAnim(FString AnimRef);

    //Function to update position/orientation of actor at each step
    virtual void Transform();

    // Returns Mesh subobject
    class USkeletalMeshComponent* GetMesh() const;

    //Reference to skeletal mesh component
    UPROPERTY(Category = Bicyclist,
        VisibleDefaultsOnly,
        BlueprintReadOnly,
        meta = (AllowPrivateAccess = "true"))
        class USkeletalMeshComponent* Mesh;

protected:
    virtual int GetNumberOfParts() { return (NumberOfParts); }

};

// Returns Mesh subobject
FORCEINLINE USkeletalMeshComponent* ABicycleActor::GetMesh() const {
    return Mesh;
}
```

Code: BicycleActor.cpp

```
// Copyright 2019-2021 The MathWorks, Inc.
#include "BicycleActor.h"
#include "SimulinkBikeAnimInst.h"
#include "Math/UnrealMathUtility.h"

ABicycleActor::ABicycleActor() {

    //Create mesh component
    Mesh = CreateOptionalDefaultSubobject<USkeletalMeshComponent>(TEXT("ABicycleMesh"));
    RootComponent = Mesh;
}

void ABicycleActor::Sim3dInit() {
    Super::Sim3dInit();
```

```
}

void ABicycleActor::Sim3dSetup() {
    SetMesh(TEXT("/MathWorksAutomotiveContent/Vehicles/Bicyclist/Meshes/SK_Bicycle"));
    SetAnim(TEXT("/MathWorksAutomotiveContent/Vehicles/Bicyclist/Meshes/BikeAnimation.BikeAnimation_C"));

    GetMesh()->SetSkeletalMesh(BicycleMesh);
    GetMesh()->SetAnimationMode(EAnimationMode::AnimationBlueprint);
    GetMesh()->SetAnimInstanceClass(BicycleAnimation);
    Transform();
}

void ABicycleActor::Sim3dStep(float DeltaTime) {
    Transform();
}

void ABicycleActor::Sim3dRelease() {
    Super::Sim3dRelease();
}

void ABicycleActor::Transform() {
    //Initialize
    int status = 0;
    FVector ActorLocation;
    FRotator ActorRotation;
    FVector ActorScale;
    USimulinkBikeAnimInst* Animation = NULL;
    Animation = Cast<USimulinkBikeAnimInst>(GetMesh()->GetAnimInstance());

    //Read data from simulink
    status = ReadSimulation3DActorTransform(readerTransform, Translation, Rotation, Scale);

    //Set bicycle position and orientation
    ActorLocation.Set(Translation[BIKE_BODY][X], Translation[BIKE_BODY][Y], Translation[BIKE_BODY][Z]);
    ActorRotation.Pitch = Rotation[BIKE_BODY][PITCH];
    ActorRotation.Roll = Rotation[BIKE_BODY][ROLL];
    ActorRotation.Yaw = Rotation[BIKE_BODY][YAW];

    //Unit conversion from simulink to UE, meteres to cm and radians to degrees
    ActorLocation = ActorLocation * 100.0f;
    ActorRotation = FMath::RadiansToDegrees(ActorRotation);
    ActorScale.Set(Scale[BIKE_BODY][X], Scale[BIKE_BODY][Y], Scale[BIKE_BODY][Z]);

    SetActorLocation(ActorLocation);
    SetActorRotation(ActorRotation);
    SetActorScale3D(ActorScale);

    //Set properies in animation blueprint
    Animation->FWheelRotation = FMath::RadiansToDegrees(Rotation[FRONT_WHEEL][ROLL]);
    Animation->RWheelRotation = FMath::RadiansToDegrees(Rotation[REAR_WHEEL][ROLL]);

    //Unit conversion from UE to simulink
    ActorLocation = GetActorLocation();
    ActorLocation = ActorLocation * .01f;           // cm -> m
    ActorRotation = GetActorRotation();
    ActorRotation = FMath::DegreesToRadians(ActorRotation);
    ActorScale = GetActorScale3D();
    Translation[BIKE_BODY][X] = ActorLocation.X;
    Translation[BIKE_BODY][Y] = ActorLocation.Y;
    Translation[BIKE_BODY][Z] = ActorLocation.Z;
    Rotation[BIKE_BODY][X] = ActorRotation.Pitch;
    Rotation[BIKE_BODY][Y] = ActorRotation.Roll;
    Rotation[BIKE_BODY][Z] = ActorRotation.Yaw;
    Scale[BIKE_BODY][X] = ActorScale.X;
    Scale[BIKE_BODY][Y] = ActorScale.Y;
    Scale[BIKE_BODY][Z] = ActorScale.Z;

    Translation[FRONT_WHEEL][X] = 0.0f;
    Translation[FRONT_WHEEL][Y] = 0.0f;
    Translation[FRONT_WHEEL][Z] = 0.0f;
    Translation[REAR_WHEEL][X] = 0.0f;
    Translation[REAR_WHEEL][Y] = 0.0f;
```

```
    Translation[REAR_WHEEL][Z] = 0.0f;

    Rotation[FRONT_WHEEL][PITCH] = 0.0f;
    Rotation[FRONT_WHEEL][ROLL] = FMath::DegreesToRadians(Animation->FWheelRotation);
    Rotation[FRONT_WHEEL][YAW] = 0.0f;
    Rotation[REAR_WHEEL][PITCH] = 0.0f;
    Rotation[REAR_WHEEL][ROLL] = FMath::DegreesToRadians(Animation->RWheelRotation);
    Rotation[REAR_WHEEL][YAW] = 0.0f;

    Scale[FRONT_WHEEL][X] = 1.0f;
    Scale[FRONT_WHEEL][Y] = 1.0f;
    Scale[FRONT_WHEEL][Z] = 1.0f;
    Scale[REAR_WHEEL][X] = 1.0f;
    Scale[REAR_WHEEL][Y] = 1.0f;
    Scale[REAR_WHEEL][Z] = 1.0f;

    //Write data back to simulink
    WriteSimulation3DActorTransform(writerTransform, Translation, Rotation, Scale);
}

void ABicycleActor::SetMesh(FString MeshPath) {
    BicycleMesh =
        Cast<USkeletalMesh>(StaticLoadObject(USkeletalMesh::StaticClass(), NULL, *MeshPath));
}

void ABicycleActor::SetAnim(FString AnimPath) {
    BicycleAnimation = StaticLoadClass(USimulinkBikeAnimInst::StaticClass(), NULL, *AnimPath);
}
```

Tip In the code, make sure to use relative paths when you specify the mesh and animation asset locations.

```
void ABicycleActor::Sim3dSetup() {
    SetMesh(TEXT("/MathWorksAutomotiveContent/Vehicles/Bicyclist/Meshes/SK_Bicycle"));
    SetAnim(TEXT("/MathWorksAutomotiveContent/Vehicles/Bicyclist/Meshes/BikeAnimation.BikeAnimation_C"));

    GetMesh()->SetSkeletalMesh(BicycleMesh);
    GetMesh()->SetAnimationMode(EAnimationMode::AnimationBlueprint);
    GetMesh()->SetAnimInstanceClass(BicycleAnimation);
    Transform();
}
```
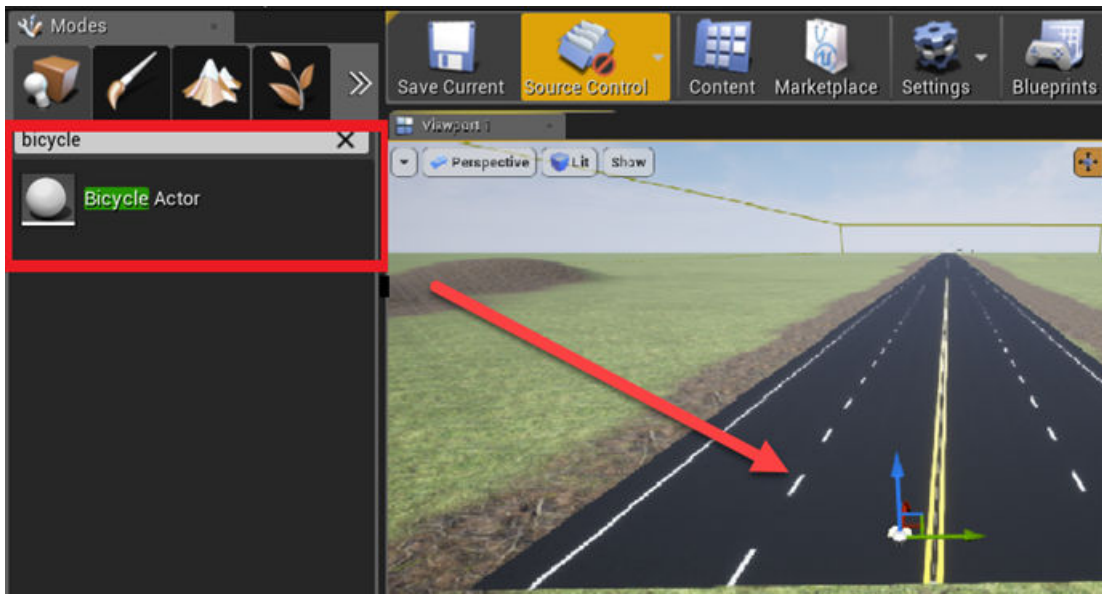
---

5 In the Unreal Editor click **Compile**.
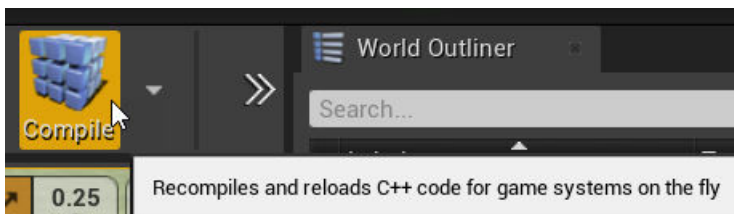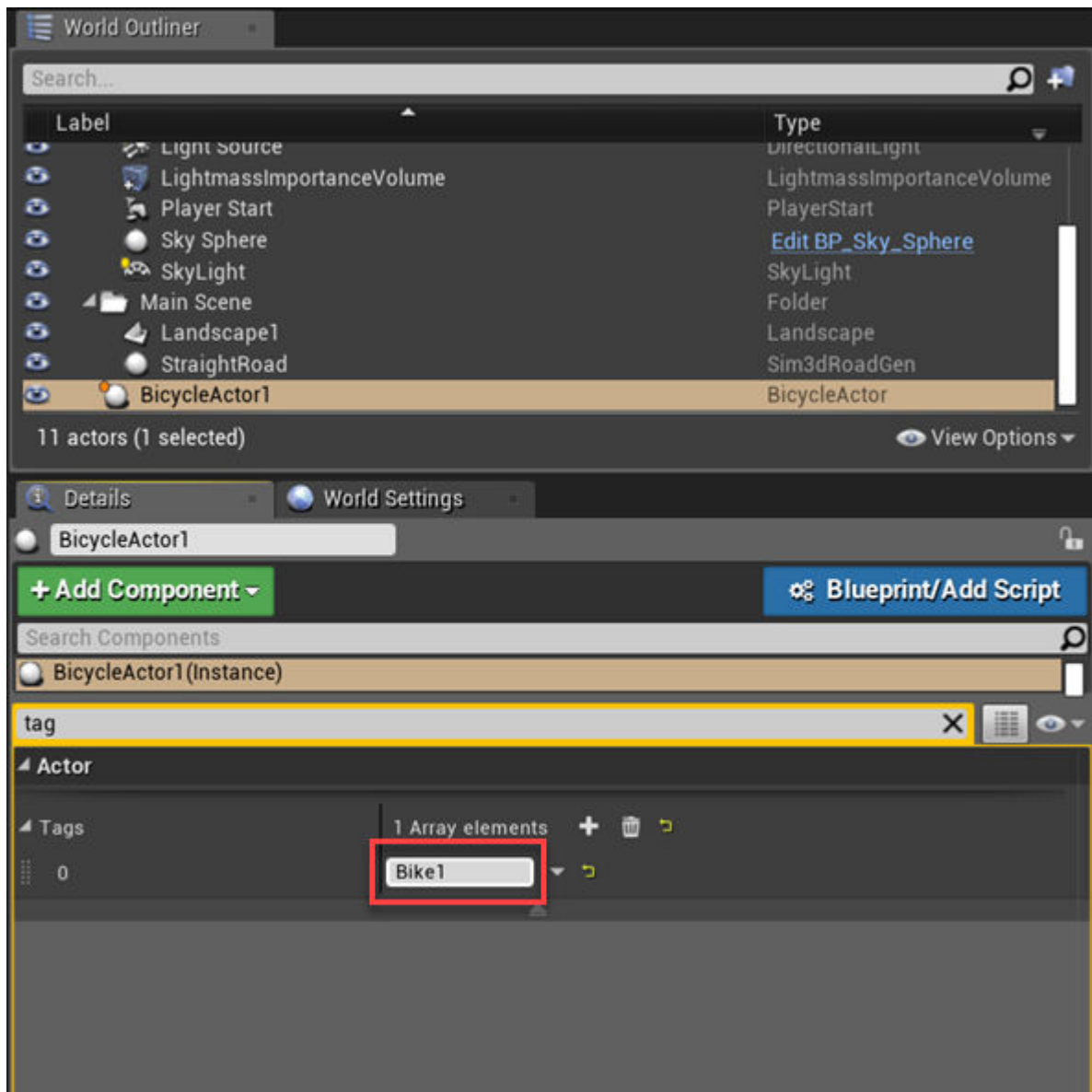

Step 6: Instantiate the Bicycle Actor

1 In your Simulink model, use the Simulation 3D Scene Configuration block **Open Unreal Editor** parameter to open the Unreal Editor.

2 Place the Bicycle Actor in the scene.

3 Set the tag to the same value as the Simulation 3D Actor Transform Set block **Tag for actor in 3D scene, ActorTag**. For this example, set the value to `Bike1`.

Set up Camera View (Optional)

Optionally, set up a camera view to override the default view. You can use either Simulink or a level blueprint to set up the camera view. For the recommended option, use Simulink.
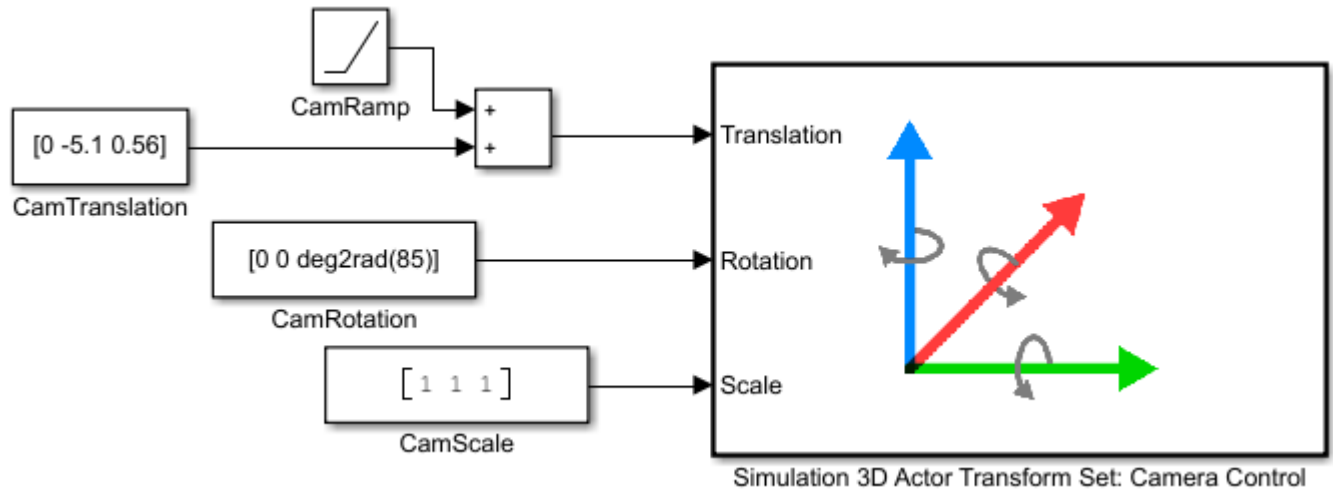
Step 7: Use Simulink (Recommended)

To setup a camera view that follows along with the bicycle:

1   Add these blocks to the model.

- One Ramp block
- One Add block

- Three Constant blocks
- Simulation 3D Actor Transform Set block

Connect and name the blocks as shown.

Simulation 3D Actor Transform Set: Camera Control

2 Set these block parameters.

Block	Parameter Settings
Simulation 3D Actor Transform Set: Camera Control	• **Tag for actor in 3D scene, ActorTag** — `MainCamera1`
CamTranslation	• **Constant value** — `[0 -5.1 0.56]` • **Interpret vector parameters as 1-D** — `off`
CamRotation	• **Constant value** — `[0 0 deg2rad(85)]` • **Interpret vector parameters as 1-D** — `off`
CamScale	• **Constant value** — `[1 1 1]` • **Interpret vector parameters as 1-D** — `off`

Step 7: Use Level Blueprint

To override the default camera view:

1 Add a camera actor. Assign it as a child of the BicycleActor.

2 Use the **Transform** settings to specify the location and viewing angle.
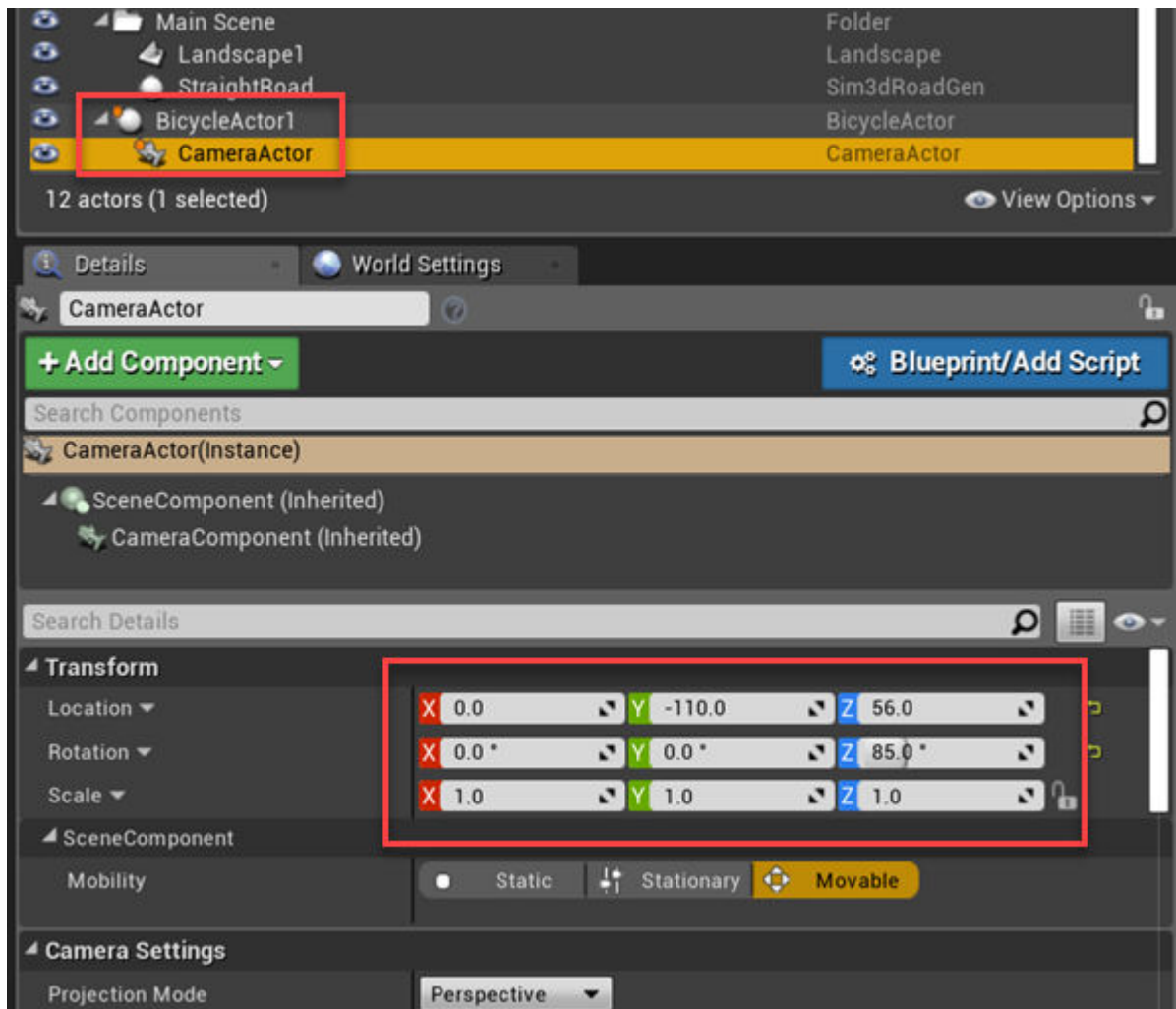
3 Open the level blueprint.
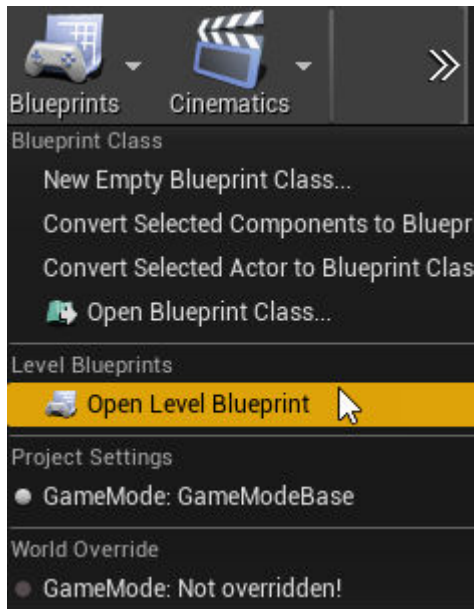
4 In the level blueprint, make these connections. If you right-click on the Event Graph to find nodes, clear **Context Sensitive**. If you have a CameraActor, you can drag it to the Event Graph from the World Outliner view in the editor.
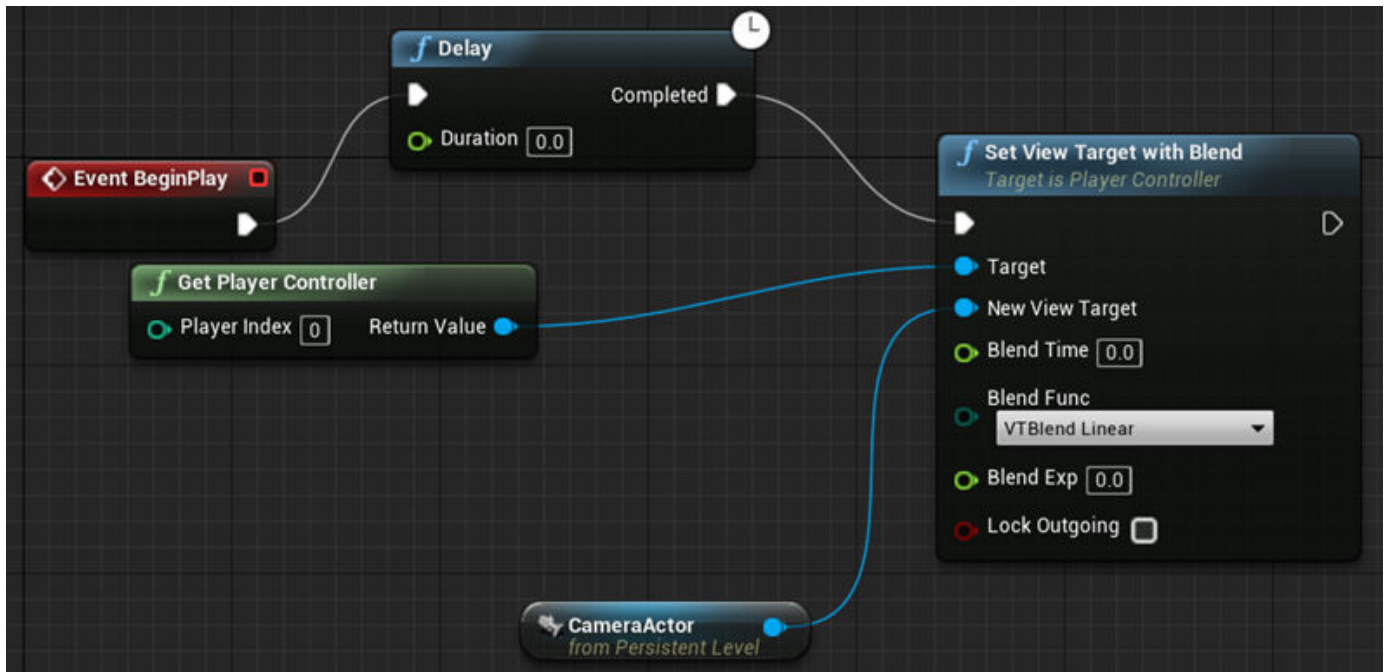
5 Save the blueprint and project. Close the Unreal Editor.

Run Simulation

After you configure the Simulink model and Unreal Editor environment, run a simulation.

1 In your Simulink model, make sure that you have set the Simulation 3D Scene Configuration parameters to these values:

- **Scene source** — `Unreal Editor`
- **Project** — Name and location of the installed support package project file, for example, `C:\Local\AutoVrtlEnv\AutoVrtlEnv.uproject`.
- **Scene view** — `Scene Origin`

2 Use the Simulation 3D Scene Configuration block **Open Unreal Editor** parameter to open the Unreal Editor.

3 Run the simulation.

    a In the Simulink model, click **Run**.

       Because the source of the scenes is the project opened in the Unreal Editor, the simulation does not start.

    b Verify that the Diagnostic Viewer window in Simulink displays this message:

```
In the Simulation 3D Scene Configuration block, you set the scene
source to 'Unreal Editor'. In Unreal Editor, select 'Play' to view the
scene.
```

       This message confirms that Simulink has instantiated the vehicles and other assets in the Unreal Engine 3D environment.

    c In the Unreal Editor, click **Play**. The simulation runs in the scene currently open in the Unreal Editor.

See Also

Simulation 3D Actor Transform Set | Simulation 3D Scene Configuration

More About

- "Get Started Communicating with the Unreal Engine Visualization Environment" on page 6-25
- "Place Cameras on Actors in the Unreal Editor" on page 8-10

External Websites

- Unreal Engine

Virtual Vehicle Composer

Get Started with the Virtual Vehicle Composer

The **Virtual Vehicle Composer** app enables you to quickly configure and build a virtual vehicle that you can use for system-level performance testing and analysis, including component sizing, fuel economy, drive cycle tracking, vehicle handling maneuvers, software integration testing, and hardware-in-the-loop (HIL) testing. Use the app to enter your vehicle parameter data, build a virtual vehicle model, run test scenarios, and analyze the results.

The virtual vehicle model utilizes sets of blocks and reference application subsystems available with Powertrain Blockset, Vehicle Dynamics Blockset, and Simscape add-ons. **Virtual Vehicle Composer** simplifies the task of configuring the architecture and entering parameter data.

Open the Virtual Vehicle Composer App

To open the app, do either of the following:

- MATLAB Toolstrip: On the **Apps** tab, under **Automotive**, click the **Virtual Vehicle Composer** app icon .
- MATLAB command prompt: Enter `virtualVehicleComposer`.

Virtual Vehicle Composer Workflow

To build, operate, and analyze your virtual vehicle, use the **Virtual Vehicle Composer** app **Composer** tab options. To get started with an example, follow the workflow steps to build a four-wheeled electric vehicle (EV), test it with FTP—75 drive cycle, and analyze the results.

Step		Button		Description
1	"Setup Virtual Vehicle" on page 9-4		**Setup**	Select **New**, then specify: - Project, folder, and model name - Powertrain architecture - Model template - Vehicle dynamics
2	"Configure Virtual Vehicle Data" on page 9-7		**Data and Calibration**	Specify the chassis, tire, brake type, powertrain, and driver. For each selection, enter the vehicle parameter data.
3	"Configure Virtual Vehicle Scenario and Test" on page 9-10		**Scenario and Test**	Select the virtual vehicle test scenario. Options include a drive cycle scenario for fuel economy and energy management analysis.
4	"Configure Virtual Vehicle Data Logging" on page 9-12		**Logging**	Select the model signal data to log when operating your virtual vehicle. Options include vehicle position, velocity, and acceleration.

Step		Button		Description
5	"Build Virtual Vehicle" on page 9-14		**Virtual Vehicle**	Build your virtual vehicle. When you build, the Virtual Vehicle Composer creates a Simulink model that contains the vehicle architecture and data that you specified in the configuration steps.
6	"Operate Virtual Vehicle" on page 9-15		**Run Test Plan**	Simulate your model in the scenario that you specified in step 2.
7	"Analyze Virtual Vehicle" on page 9-16		**Simulation Data Inspector**	Use the Simulation Data Inspector to view and inspect the simulation signals.

See Also
Virtual Vehicle Composer

Setup Virtual Vehicle

Use the **Virtual Vehicle Composer** app to configure your virtual vehicle. First, specify the project path and configuration name, powertrain architecture, model template, and vehicle dynamics.
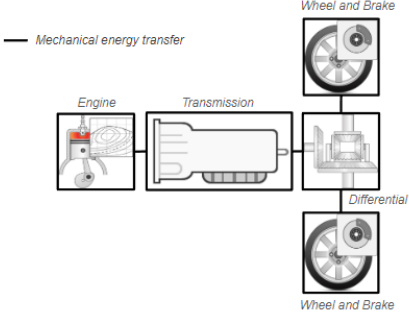
In the **Virtual Vehicle Composer** app, on the **Composer** tab, click **New** ![plus icon]. The app opens a default virtual vehicle template and creates virtual vehicle project files.

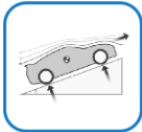
For this example, configure a four-wheeled electric vehicle (EV) with a single motor, using a Simulink model template with longitudinal vehicle dynamics. Set:

1 **Project path** to C:\Users\<*user_name*>\MATLAB\Projects\examples.
2 **Configuration name** to ConfiguredVirtual_EV.
3 **Vehicle class** to Passenger car.
4 **Powertrain Architecture** to Electric Vehicle 1EM.
5 **Model template** to Simulink .
6 **Vehicle dynamics** to Longitudinal vehicle dynamics.

Then click **Configure**.

After completing this step, see "Configure Virtual Vehicle Data" on page 9-7.

More About

Vehicle class

Use the **Vehicle class** parameter to choose between a four-wheeled `Passenger car` or a two-wheeled `Motorcycle`.

Powertrain Architecture

Set the **Powertrain Architecture** parameter. The options depend on the **Vehicle class**.

For a `Passenger car` the default architecture is `Conventional Vehicle` equipped with an internal combustion engine and a transmission. You can also select `Electric Vehicle 1EM` to specify a single-motor EV powertrain. If you have Powertrain Blockset, you can specify other EV architectures and hybrid electric vehicle (HEV) architectures.

For a `Motorcycle` the default architecture is `Conventional Motorcycle with Chain Drive` having a spark-ignition (SI) engine. You can also select `Electric Motorcycle with Chain Drive` to specify an EV architecture. Motorcycles require Vehicle Dynamics Blockset, Simscape, and Simscape add-ons.

Model Template

Use the **Model template** parameter to specify a `Simulink` or `Simscape` vehicle plant and powertrain architecture. The default for a `Passenger car` is a `Simulink` model template. For a `Motorcycle` the only option is a `Simscape` model template.

For a `Passenger car`, if you have Simscape Driveline™, you can configure the vehicle plant and powertrain architecture with Simscape subsystems that model a conventional vehicle. If you have Simscape Driveline and Simscape Electrical™, you can configure the vehicle plant and powertrain architecture with Simscape subsystems that model EVs and HEVs.

Configuring a `Motorcycle` requires Vehicle Dynamics Blockset, Simscape, Simscape Driveline, and Simscape Electrical.

Vehicle Dynamics

Use the **Vehicle Dynamics** parameter to configure the virtual vehicle dynamics.

For a `Passenger car` select one of the following options.

-

  `Longitudinal vehicle dynamics` — Suitable for fuel economy and energy management analysis.

- `Combined longitudinal and lateral vehicle dynamics` — If you have Vehicle Dynamics Blockset, you can specify dynamics suitable for vehicle handling, stability, and ride comfort analysis.

For a `Motorcycle` select one of the following options.

- `In-plane motorcycle dynamics` — Suitable for fuel economy and energy management analysis.

- `Out-of-plane motorcycle dynamics` — Suitable for vehicle handling, stability, and ride comfort analysis.

Note The virtual vehicle uses the Z-up coordinate system as defined in SAE J670 and ISO 8855.

See Also
Virtual Vehicle Composer

Related Examples
- "Coordinate Systems in Vehicle Dynamics Blockset" on page 2-2
- "Create Projects"
- "Get Started with the Virtual Vehicle Composer" on page 9-2

Configure Virtual Vehicle Data

Before completing this step, see "Setup Virtual Vehicle" on page 9-4, which began the setup of an EV with one electric motor.

Next, use the **Data and Calibration** options to configure the virtual vehicle chassis, tire, brake type, powertrain, environment and driver. The available options depend on the virtual vehicle **Powertrain architecture** and **Model template** parameter settings.

Chassis

Use the **Chassis** parameter to select the body dynamics and mass properties. The available options depend on the virtual vehicle **Setup > Vehicle dynamics** parameter.

For this example, set **Chassis** to `Vehicle Body 3DOF Longitudinal`, and use the default values for the mass properties.

Tire and Brake

Use the **Tire** and **Brake Type** options to specify the tire and brake parameters.

1 Set **Tire** to `MF Tires Longitudinal` to configure a tire model suitable for drive cycle analysis.

   • On the **Tire Data** tab, enter the tire parameters for your virtual vehicle, including:

- `PlntWhlLdRadius` — Tire loaded radius
- `PlntWhlPrsFrnt` — Front tire pressure

For this example, use the default parameter values.

2 Set **Brake Type** to `Disc`.

- Enter the brake parameters for your virtual vehicle, including:

  - `PlntBrkStcFricCffFrnt` — Static friction coefficient for front brakes
  - `PlntBrkKinFricCffFrnt` — Kinetic friction coefficient for front brakes

  For this example, use the default parameter values.

Powertrain

Under the **Powertrain** tab, set parameters for the powertrain systems such as the engine, electric motor, transmission, drivetrain, differential system, and electrical system for your virtual vehicle. The available options depend on the virtual vehicle **Powertrain architecture** and **Model template** parameter settings.

For this example, under **Powertrain**, set:

1 **Vehicle Control Unit** to `EV 1EM with BMS`.

2 **Drivetrain** to `Front Wheel Drive`.

3 **Drivetrain** > **Front Differential System** to `Open Differential`.

4 **Electrical System** to `Electrical System`.

- Specify the **DC-DC Converter** parameters, such as electrical conversion losses and measured efficiency:

  - `PlntDCDCEff` — Overall converter efficiency
  - `PlntDCDCLossTbl` — Conversion losses

  For this example, use the default parameter values.

- Click **Electric Machine 1** > **Parameters** to specify the motor parameters, including:

  - `PlntEM1Spd` — Vector of rotational speeds in torque table
  - `PlntEM1Trq` — Corresponding vector of maximum torque values

  For this example, use the default parameter values.

- Click **Energy Storage**, then select `Mapped Battery (Electric Vehicle 1EM)` to specify parameters for a mapped lithium-ion battery model, including:

  - `PlntBattOpenCirctVolt` — Open circuit voltage table data
  - `PlntBattVoltSocBpt` — Open circuit voltage breakpoints

  For this example, use the default parameter values.

Driver

Under the **Driver** tab, set the driver parameters. The available options depend on the virtual vehicle **Powertrain architecture** and **Model template** parameter settings.

For this example, set **Driver** to `Longitudinal Driver` to implement a driver suitable for drive-cycle tracking.

- Enter the driver data for your virtual vehicle, including:

  - `DriverPreviewDist` — Preview distance
  - `DriverTimeConst` — Time constant

  For this example, use the default parameter values.

Environment

Under the **Environment** tab, set the environment parameters. Set **Environment** to `Standard Ambient`.

- Enter the environment data for your virtual vehicle, including:

  - `EnvAirTemp` — Ambient air temperature
  - `EnvWindVelX` — Ambient wind velocity in X direction

  For this example, use the default parameter values.

After completing this step, see "Configure Virtual Vehicle Scenario and Test" on page 9-10.

See Also
Virtual Vehicle Composer

Related Examples
- "Get Started with the Virtual Vehicle Composer" on page 9-2

Configure Virtual Vehicle Scenario and Test

Before completing this step, see "Configure Virtual Vehicle Data" on page 9-7.

Next, use the **Scenario and Test** tab to configure your virtual vehicle test plan. The available options depend on the virtual vehicle **Powertrain architecture** and **Model template** parameter settings.

If you set **Scenario** to `Drive Cycle`, you can use:

- Drive cycles from predefined sources. By default, the block includes the FTP–75 drive cycle. To install additional drive cycles from the support package, see "Support Package for Maneuver and Drive Cycle Data" on page 6-2. The support package has drive cycles that include the gear shift schedules, for example, `JC08` and `CUEDC`.

- Workspace variables that define your own drive cycles.

- .mat, .xls, .xlsx, or .txt files.

- Wide open throttle (WOT) parameters, including initial and nominal reference speeds, deceleration start time, and final reference speed.

To choose one or more existing drive cycles, click the **Scenario and Test** tab and follow these steps:

1   Under **Scenario**, select `Drive Cycle`.

2   Under **Drive cycle**, select the desired drive cycle.

3   Click **Add to Test Plan**.

If you have selected `Combined longitudinal and lateral vehicle dynamics` (not applicable in this example), you can also choose from several vehicle dynamics tests under the **Scenario** tab:

- `Increasing Steer`
- `Swept Sine`
- `Sine with Dwell`
- `Fishhook`

For this example, use the default `FTP75` drive cycle.

After completing this step, see "Configure Virtual Vehicle Data Logging" on page 9-12.

See Also
Virtual Vehicle Composer

Related Examples
-

Configure Virtual Vehicle Data Logging

Before completing this step, see "Configure Virtual Vehicle Scenario and Test" on page 9-10.

Next, use the **Virtual Vehicle Composer** app to configure the data set that you want to log, such as energy-related quantities and vehicle position, velocity, and acceleration. The signals available for logging depend on your **Powertrain architecture**, **Model template**, and **Scenario and Test** parameter settings.

Under the **Logging** tab, the app has a default set of signals in the **Selected Signals** list. You can add or remove signals to suit your task. For this example, log the default signals in the list.
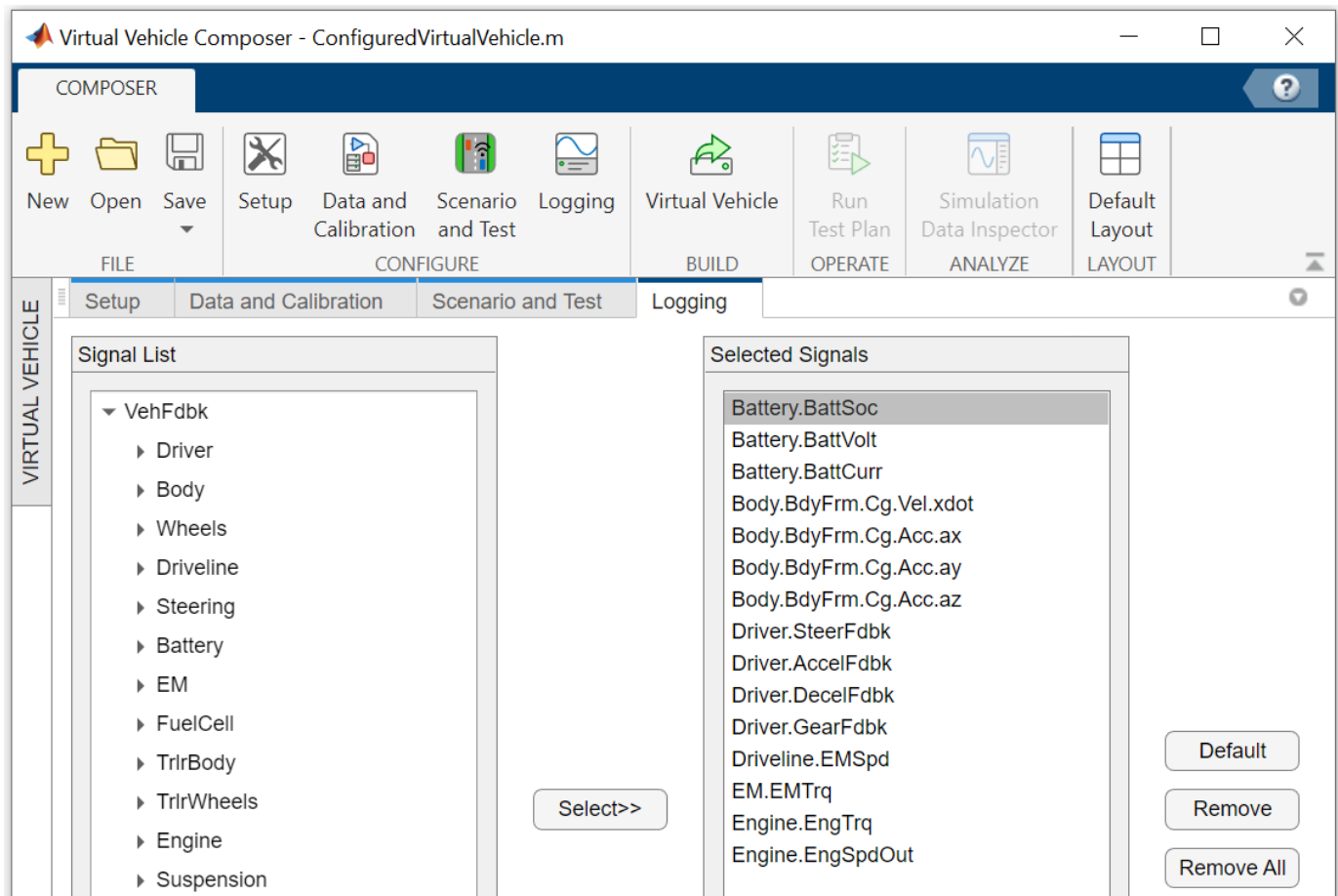
After completing this step, see "Build Virtual Vehicle" on page 9-14.

See Also
Virtual Vehicle Composer

Related Examples
- "Get Started with the Virtual Vehicle Composer" on page 9-2

More About

- "Simulation Data Inspector"

Build Virtual Vehicle

Before completing this step, see "Configure Virtual Vehicle Data Logging" on page 9-12.

Next, use the **Virtual Vehicle Composer** app to build your virtual vehicle. When you build, the app creates a model that incorporates the vehicle architecture and parameters that you have specified, and associates it with the test plan you configured.

In the app **Build** section, click **Virtual Vehicle** to start the build.

The build takes time to complete. View progress in the MATLAB Command Window.

The app names the model `ConfiguredVirtualVehicleModel`.
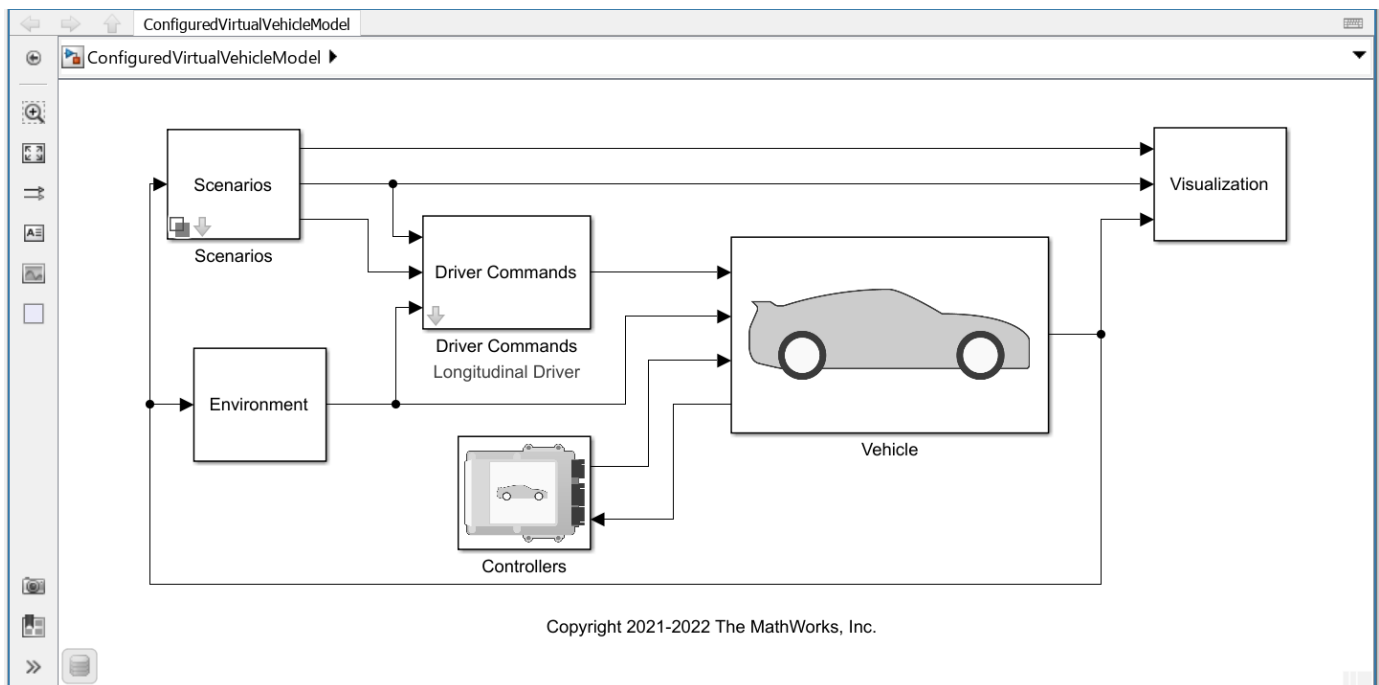
After completing this step, see "Operate Virtual Vehicle" on page 9-15.

See Also
Virtual Vehicle Composer

Related Examples
*   "Get Started with the Virtual Vehicle Composer" on page 9-2

Operate Virtual Vehicle

Before completing this step, see "Build Virtual Vehicle" on page 9-14.

Next, use the **Virtual Vehicle Composer** app to operate your virtual vehicle. When you operate the vehicle, the app simulates the model executing the test plan that you specified in **Scenario and Test**, which in this example is an electric vehicle (EV) executing an FTP75 drive cycle.

To operate the model, click the **Run Test Plan** button in the **Operate** section on the **Composer** tab.

The simulations take time to complete. View progress in the MATLAB Command Window.

After completing this step, see "Analyze Virtual Vehicle" on page 9-16.

See Also
Virtual Vehicle Composer

Related Examples
*   "Get Started with the Virtual Vehicle Composer" on page 9-2

Analyze Virtual Vehicle

Before completing this step, see "Operate Virtual Vehicle" on page 9-15.

Next, use the **Virtual Vehicle Composer** app to analyze the results of operating your virtual vehicle. The app uses the Simulation Data Inspector to display signals from the list that you configured on the **Logging** tab. For this example, you can analyze the signals logged while your electric vehicle (EV) executed the FTP75 drive cycle.

In the app **Analyze** section, click **Simulation Data Inspector** ▯.
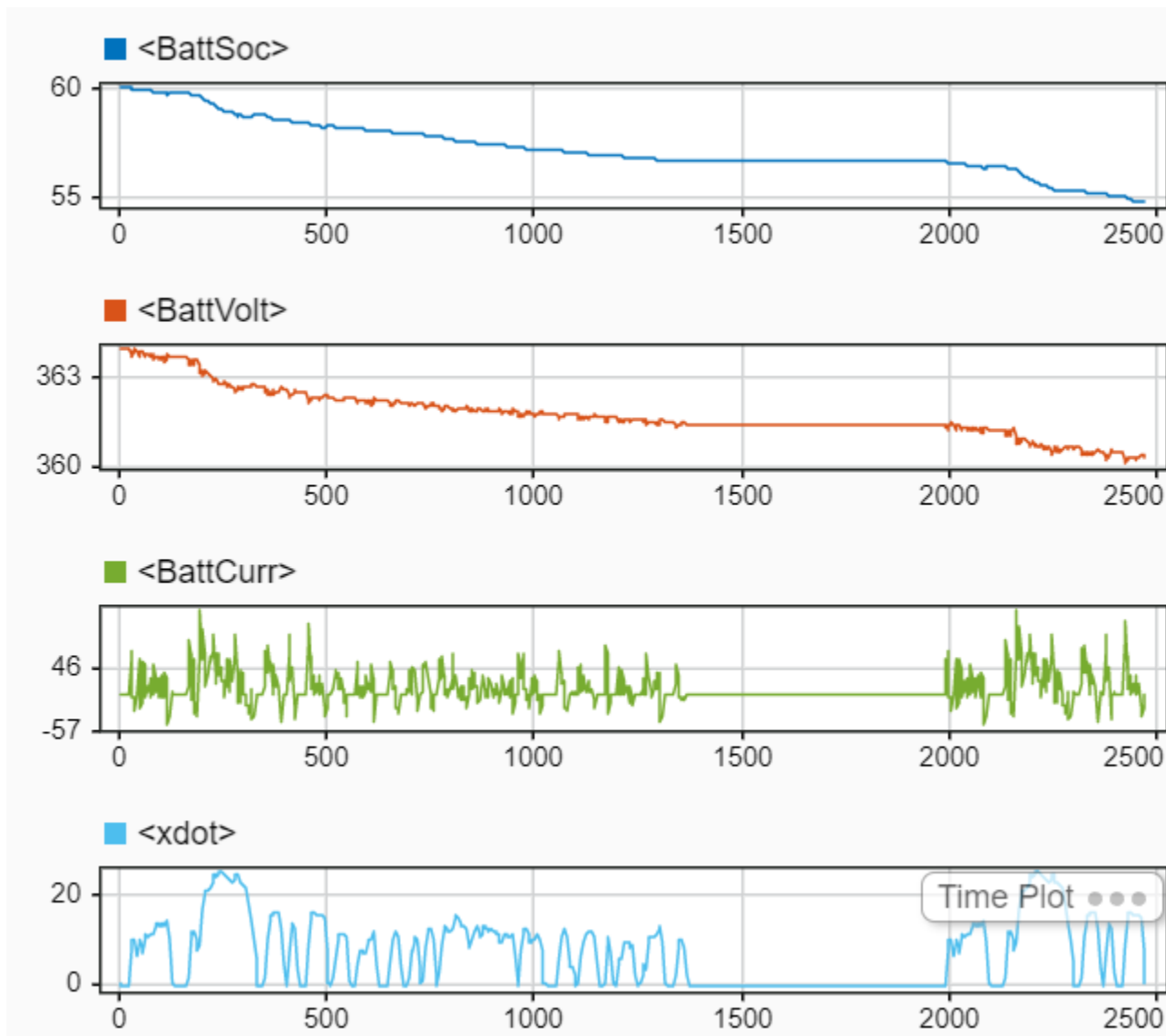
See Also
Virtual Vehicle Composer

Related Examples

- "Get Started with the Virtual Vehicle Composer" on page 9-2

More About

- "Simulation Data Inspector"

Resize Mapped Motor

Use the **Virtual Vehicle Composer** app to resize a motor based on high-level specifications.

For virtual vehicles configured with an electric vehicle (EV) or HEV powertrain architecture, you can resize the mapped motor based on a desired maximum motor power.

After you setup your virtual vehicle, in the **Virtual Vehicle Composer** app, follow these steps.

1   In the **Virtual Vehicle** pane, select **Electric Machine (Motor)**, then click the **Data and Calibration** tab.
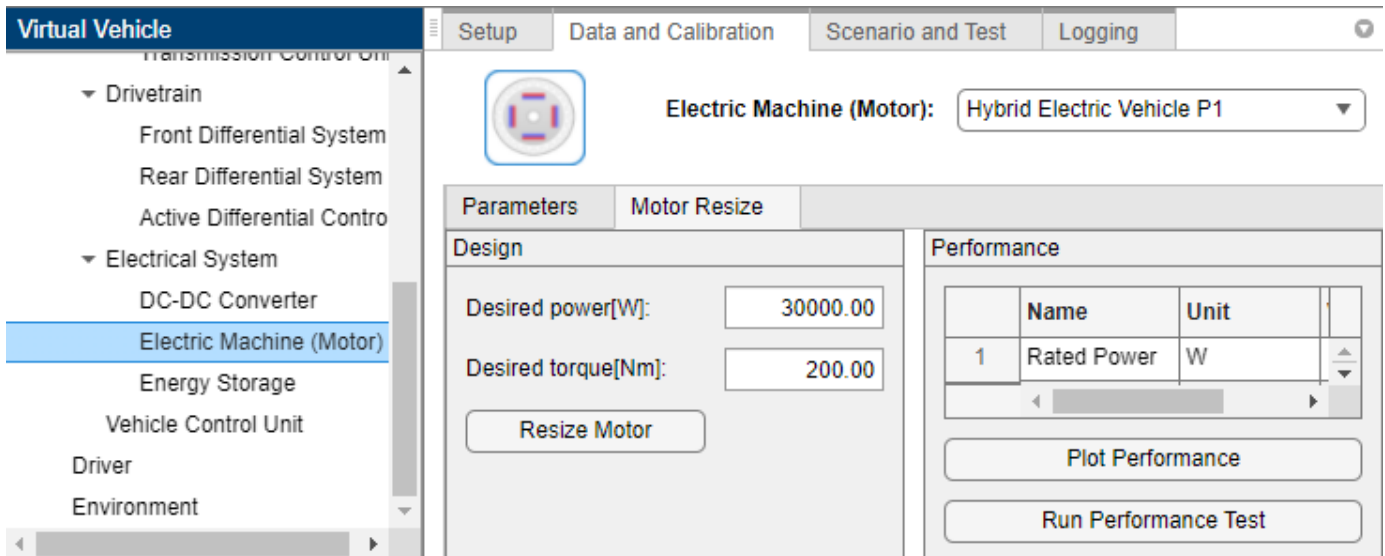
2   Specify the **Motor Resize** motor parameter options, including desired power, and desired torque.

3   Select **Resize Motor**. The Virtual Vehicle Composer resizes the mapped motor. The **Performance** pane provides the updated performance characteristics and plots based on the resized motor. If you do not resize the motor, the pane provides the performance characteristics and plots of the existing motor.

See Also
Virtual Vehicle Composer

Related Examples
*   "Get Started with the Virtual Vehicle Composer" on page 9-2

Calibrate Mapped CI Engine Using Data

If you have Model-Based Calibration Toolbox, the **Virtual Vehicle Composer** can use a dataset to calibrate a mapped compression-ignition (CI) engine in the vehicle.

1   Open the **Virtual Vehicle Composer** app.

2   On the **Setup** tab:

   a   Specify your virtual vehicle options, including **Vehicle class**, **Model template**, and **Vehicle dynamics**.

   b   Set **Powertrain architecture** to a configuration that uses a CI engine, for example, `Conventional Vehicle`.

   c   Click **Configure**.

3   On the **Data and Calibration** tab, select **Powertrain > Engine**.

4   In the **Engine** list, select `CI Mapped Engine`.

5   Select the **Calibrate from Data** tab.

6   Use the **Firing Data** and **Nonfiring Data** boxes to provide data files. By default, the app uses the file `CiEngineData.xlsx`, which contains required and optional data. The tables summarize the data file requirements for generating calibrated tables that are functions of either injected fuel mass or engine torque and engine speed.

   Firing data contains data collected at different engine torques and speeds.

Firing Data	Description	Data Requirements for Generating Mapped Engine Tables	
		Function of Fuel Mass and Engine Speed	Function of Torque and Engine Speed
FuelMassCmd	Injected fuel mass, in mg per injection	*Required*	*Not used*
Torque	Engine torque command, in N·m	*Required*	*Required*
EngSpd	Engine speed, in rpm	*Required*	*Required*
AirMassFlwRate	Air mass flow, in kg/s	*Optional*	*Optional*
FuelMassFlwRate	Fuel mass flow, in kg/s	*Optional*	*Optional*
ExhTemp	Exhaust temperature, in K	*Optional*	*Optional*
BSFC	Engine brake-specific fuel consumption (BSFC), in g/kWh	*Optional*	*Optional*
HCMassFlwRate	Hydrocarbon emission mass flow, in kg/s	*Optional*	*Optional*

Firing Data	Description	Data Requirements for Generating Mapped Engine Tables	
		Function of Fuel Mass and Engine Speed	**Function of Torque and Engine Speed**
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s	*Optional*	*Optional*
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s	*Optional*	*Optional*
CO2MassFlwRate	Carbon dioxide emission mass flow, in kg/s	*Optional*	*Optional*
PMMassFlwRate	Particulate matter emission mass flow, in kg/s	*Optional*	*Optional*

Nonfiring data contains data collected at different engine speeds without fuel consumption.

Nonfiring Data	Description	Data Requirements for Generating Mapped Engine Tables	
		Function of Fuel Mass and Engine Speed	**Function of Torque and Engine Speed**
FuelMassCmd	Injected fuel mass, in mg per injection	*Not used*	*Not used*
Torque	Engine torque command, in N·m	*Required*	*Required*
EngSpd	Engine speed, in rpm	*Required*	*Required*
AirMassFlwRate	Air mass flow, in kg/s	*Optional*	*Optional*

7   Click **Calibrate** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). To calibrate the data, Model-Based Calibration Toolbox uses templates.

   If prompted, select the firing or non-firing data sheets. Click **OK**.

   When the process completes, the app updates the powertrain subsystem Mapped CI Engine block parameters with the calibrated data.

8   Review the engine characteristics response surface models, for example, air mass flow.
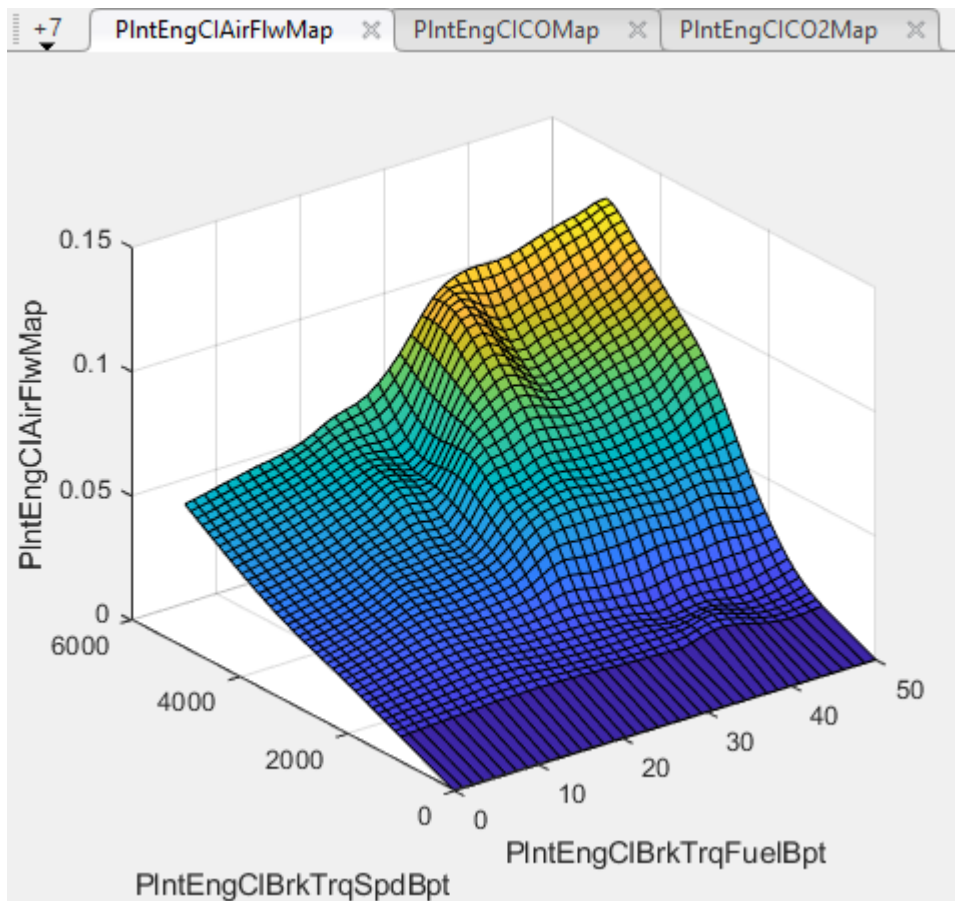
9 Optionally, to use additional calibration options, click **Open Calibration Tool**. The Model-Based Calibration Toolbox opens.

- The Model Browser provides the response model fits for the data contained in the data file.
- The CAGE Browser provides the calibrated data.

For information, see "Model-Based Calibration Toolbox".

See Also
Mapped CI Engine

More About
- "What Is CAGE?" (Model-Based Calibration Toolbox)
- "Mapped CI Lookup Tables as Functions of Fuel Mass and Engine Speed" (Model-Based Calibration Toolbox)
- "Mapped CI Lookup Tables as Functions of Engine Torque and Speed" (Model-Based Calibration Toolbox)
- "Generate Mapped CI Engine from a Spreadsheet" (Powertrain Blockset)
- "Model Assessment" (Model-Based Calibration Toolbox)
- "Using Data" (Model-Based Calibration Toolbox)

Calibrate Mapped SI Engine Using Data

If you have Model-Based Calibration Toolbox, the **Virtual Vehicle Composer** can use a dataset to calibrate a mapped spark-ignition (SI) engine in the vehicle.

1   Open the **Virtual Vehicle Composer** app.

2   On the **Setup** tab:

   a   Specify your virtual vehicle options, including **Vehicle class**, **Model template**, and **Vehicle dynamics**.

   b   Set **Powertrain architecture** to a configuration that uses a SI engine, for example, `Hybrid Electric Vehicle P2`.

   c   Click **Configure**.

3   On the **Data and Calibration** tab, select **Powertrain > Engine**.

4   Set **Engine** to `SI Mapped Engine`.

5   Select the **Calibrate from Data** tab.

6   Use the **Firing Data** and **Nonfiring Data** boxes to provide data files. By default, the app uses the file `SiEngineData.xlsx`, which contains required and optional data. The tables summarize the data file requirements for generating calibrated tables that are functions of engine torque and engine speed.

   Firing data contains data collected at different engine torques and speeds.

Firing Data	Description	Data Requirements for Generating Mapped Engine Tables
FuelMassCmd	Injected fuel mass, in mg per injection	*Not used*
Torque	Engine torque command, in N·m	*Required*
EngSpd	Engine speed, in rpm	*Required*
AirMassFlwRate	Air mass flow, in kg/s	*Optional*
FuelMassFlwRate	Fuel mass flow, in kg/s	*Optional*
ExhTemp	Exhaust temperature, in K	*Optional*
BSFC	Engine brake-specific fuel consumption (BSFC), in g/kWh	*Optional*
HCMassFlwRate	Hydrocarbon emission mass flow, in kg/s	*Optional*
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s	*Optional*
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, in kg/s	*Optional*
CO2MassFlwRate	Carbon dioxide emission mass flow, in kg/s	*Optional*

Firing Data	Description	Data Requirements for Generating Mapped Engine Tables
PMMassFlwRate	Particulate matter emission mass flow, in kg/s	*Optional*

Nonfiring data contains data collected at different engine speeds without fuel consumption.

Nonfiring Data	Description	Data Requirements for Generating Mapped Engine Tables
FuelMassCmd	Injected fuel mass, in mg per injection	*Not used*
Torque	Engine torque command, in N·m	*Required*
EngSpd	Engine speed, in rpm	*Required*
AirMassFlwRate	Air mass flow, in kg/s	*Optional*

7 Click **Calibrate** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). To calibrate the data, Model-Based Calibration Toolbox uses templates.

If prompted, select the firing or non-firing data sheets. Click **OK**.

When the process completes, the app updates the powertrain subsystem Mapped SI Engine block parameters with the calibrated data.

8 Review the engine characteristics response surface models, for example, air mass flow.
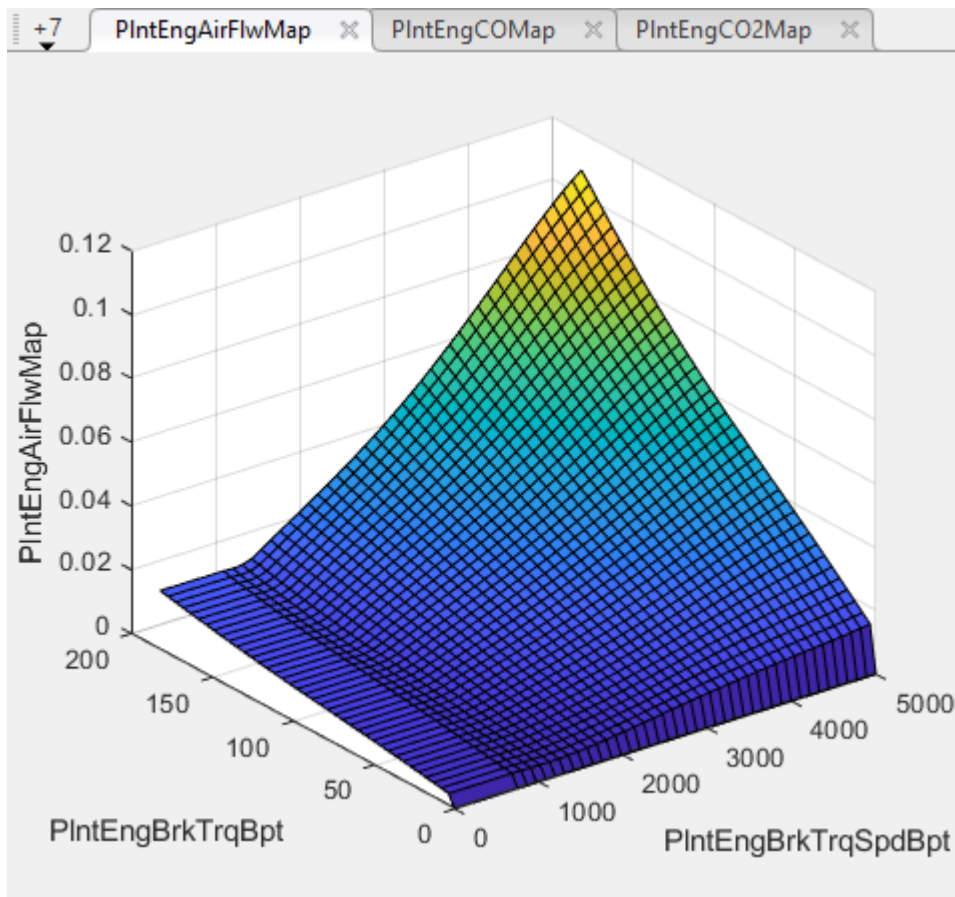
9   Optionally, to use additional calibration options, click **Open Calibration Tool**. The Model-Based Calibration Toolbox opens.

- The Model Browser provides the response model fits for the data contained in the data file.
- The CAGE Browser provides the calibrated data.

For information, see "Model-Based Calibration Toolbox".

See Also
Mapped SI Engine

More About
- "What Is CAGE?" (Model-Based Calibration Toolbox)
- "Mapped SI Lookup Tables as Functions of Engine Torque and Speed" (Model-Based Calibration Toolbox)
- "Generate Mapped SI Engine from a Spreadsheet" (Powertrain Blockset)
- "Model Assessment" (Model-Based Calibration Toolbox)
- "Using Data" (Model-Based Calibration Toolbox)

Calibrate Mapped Electric Motors Using Data

If you have Model-Based Calibration Toolbox, the **Virtual Vehicle Composer** can use a dataset to calibrate a mapped electric motor in an electric vehicle (EV) or hybrid electric vehicle (HEV).

1   Open the **Virtual Vehicle Composer** app.

2   On the **Setup** tab:

    a   Specify your virtual vehicle options, including **Vehicle class**, **Model template**, and **Vehicle dynamics**.

    b   Set **Powertrain architecture** to a configuration that uses an EV or HEV, for example, `Electric Vehicle 3EM Dual Front`.

    c   Click **Configure**.

3   On the **Data and Calibration** tab, select the electric motor that you want to calibrate. Specifically, select **Powertrain > Electrical System > Electric Machine x**, where x is 1, 2, 3, or 4 and corresponds to the motor.

    For example, select **Electric Machine 1**.

4   Select the **Calibrate from Data** tab.

5   Use **Browse** to specify the data file. By default, the app uses the file `MappedMotorDataset.xlsx`, which contains the required data.

    The table summarizes the data file requirements for generating calibrations from measured motor power loss data at steady-state operating conditions.

Calibration Type	Required Data
`Tabulated loss data`	• Motor speed, rad/s • Motor torque, N·m • Power loss, W

6   Click **Calibrate** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). To calibrate the data, Model-Based Calibration Toolbox uses templates.

    When the process completes, the app updates the powertrain subsystem Mapped Motor block parameters with the calibrated data.

7   Review the electric motor loss characteristics response surface model.
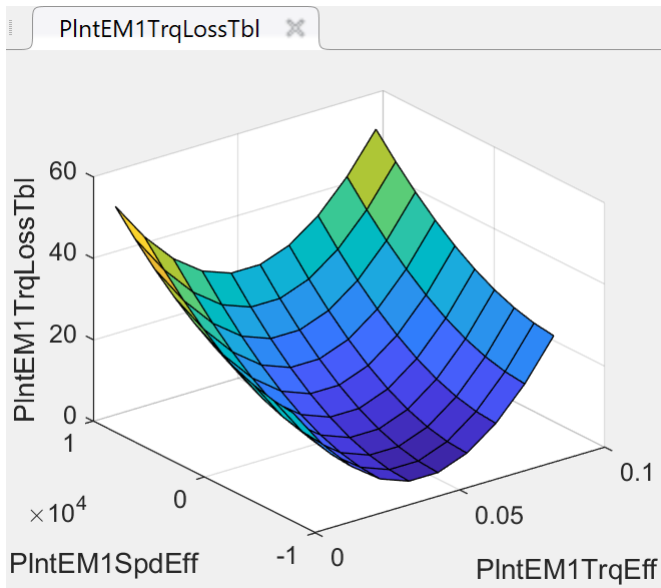
8   Optionally, to adjust the calibration, click **Open Calibration Tool**. The Model-Based Calibration Toolbox opens.

- The Model Browser provides the response model fits for the data contained in the data file.
- The CAGE Browser provides the calibrated data.

For information, see "Model-Based Calibration Toolbox".

See Also
Mapped Motor

More About
- "What Is CAGE?" (Model-Based Calibration Toolbox)
- "Model Assessment" (Model-Based Calibration Toolbox)
- "Using Data" (Model-Based Calibration Toolbox)