Powertrain Blockset™

User's Guide

MATLAB®&SIMULINK®

R2023**a**

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

Powertrain Blockset™ User's Guide

Trademarks

Patents

Revision History

October 2016	Online only	New for Version 1.0 (Release 2016b+)
March 2017	Online only	Revised for Version 1.1 (Release 2017a)
September 2017	Online only	Revised for Version 1.2 (Release 2017b)
March 2018	Online only	Revised for Version 1.3 (Release 2018a)
September 2018	Online only	Revised for Version 1.4 (Release 2018b)
March 2019	Online only	Revised for Version 1.5 (Release 2019a)
September 2019	Online only	Revised for Version 1.6 (Release 2019b)
March 2020	Online only	Revised for Version 1.7 (Release 2020a)
September 2020	Online only	Revised for Version 1.8 (Release 2020b)
March 2021	Online only	Revised for Version 1.9 (Release 2021a)
September 2021	Online only	Revised for Version 1.10 (Release 2021b)
March 2022	Online only	Revised for Version 1.11 (Release 2022a)
September 2022	Online only	Revised for Version 1.12 (Release 2022b)
March 2023	Online only	Revised for Version 2.0 (Release 2023a)

Contents

1 Getting Started

2 Workflows

3 Reference Applications

4 Project Templates

5 Supporting Data

6 Calibration

7 Powertrain Blockset Examples

8 Virtual Vehicle Composer

1

Getting Started

Powertrain Blockset Product Description

Model and simulate automotive powertrain systems

Powertrain Blockset provides preassembled automotive vehicle reference applications for gasoline, diesel, hybrid, fuel cell, and battery electric propulsion systems. The blockset includes a component library for engines, traction motors, batteries, transmissions, tires, and driver models, as well as component and supervisory controllers.

Powertrain Blockset offers the Virtual Vehicle Composer app for configuring and parameterizing models, as well as prebuilt workflows for resizing components, calibrating models from data, optimizing shift schedules, and generating deep learning dynamic plant and state estimators. You can use these models for design tradeoff analysis and component sizing, control parameter optimization, 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

- Fully assembled models for gasoline, diesel, hybrid, and electric powertrains
- Libraries of engine, transmission, traction motor, and battery components
- Basic controllers for powertrain subsystems
- Standard drive cycle data, including FTP75, NEDC, and JC08
- Engine dynamometer model for virtual calibration and testing
- MDF file support for calibration data import

Required and Recommended Products

Required Products

Powertrain Blockset product requires current versions of these products:

- MATLAB
- Simulink

Recommended Products

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

Goal	Recommended Product
Model events	Stateflow®
Use physical modeling blocks	Simscape and Simscape™ add-ons
Optimize powertrain performance and control parameters	Optimization Toolbox™
Generate reports	MATLAB® Report Generator™ Simulink® Report Generator
Optimize powertrain design	Simulink Design Optimization™
Parallel computing	MATLAB Parallel Server™ Parallel Computing Toolbox™
Calibrate engine models	Model-Based Calibration Toolbox™

Getting Started with Powertrain Blockset

The Powertrain Blockset provides reference application projects assembled from blocks and subsystems. Use the reference applications as a starting point to create your own powertrain models.

Objective	For	See
Design tradeoff analysis and component sizing, control parameter optimization, or hardware-in-the-loop (HIL) testing.	Full conventional vehicle with spark-ignition (SI) or combustion-ignition (CI)	"Build a Conventional Vehicle Model" on page 3-5
	Hybrid electric vehicle (HEV) — Multimode	"Build Hybrid Electric Vehicle Multimode Model" on page 3-19
	HEV — Input power-split	"Build Hybrid Electric Vehicle Input Power-Split Model" on page 3-42
	Full electric vehicle	"Build Full Electric Vehicle Model" on page 3-26
Engine and controller calibration, validation, and optimization before integration with the vehicle model.	CI engine plant and controller	"Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness" on page 3-11
	SI engine plant and controller	"Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness" on page 3-15

This example shows how to run the conventional vehicle reference application and examine the final drive gear ratio impact on fuel economy and tailpipe emissions.

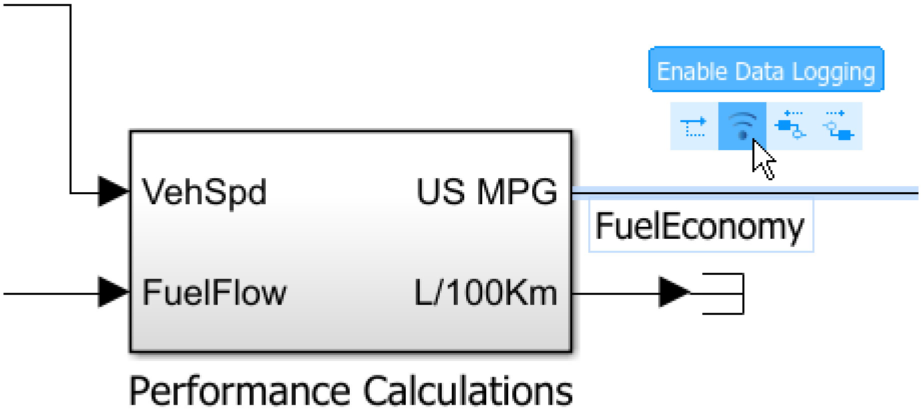
Running this example requires a Stateflow license. You can install a Stateflow trial license using the Add-On Explorer.

1. Open the conventional vehicle reference application project. By default, the application has a 1.5–L spark-ignition (SI) engine and a final drive gear ratio of 3.

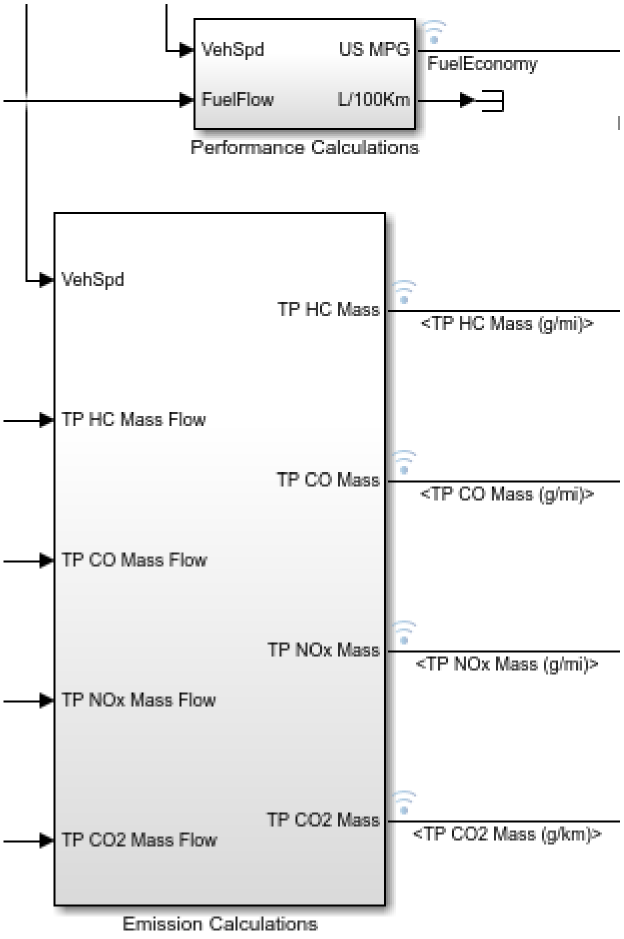
   ```
   autoblkConVehStart
   ```

   Project files open in a writable location.
2. Enable data logging for the fuel economy and tailpipe emissions signals.

   a. In the `Visualization` subsystem, select the `FuelEconomy` signal line and `Enable Data Logging`.



b In the `Visualization` subsystem, enable data logging on the tailpipe emissions signals.



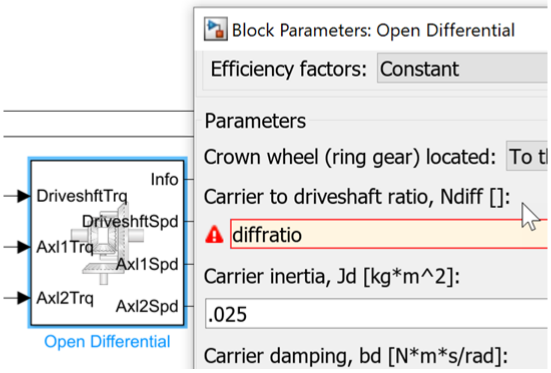
c Save the `SiCiPtReferenceApplication` model.

3 Parameterize the final drive gear ratio.

a In the `Passenger Car` subsystem, navigate to the `DrivetrainConVeh` > `Differential and Compliance` > `Front Wheel Drive` subsystem. Open the Open Differential block.

b In the Open Differential block mask:

- Change the **Carrier to driveshaft ratio, Ndiff** parameter to the variable `diffratio`. The **Carrier to driveshaft ratio, Ndiff** parameter represents the final drive gear ratio.



Block Parameters: Open Differential

Efficiency factors: Constant

Parameters

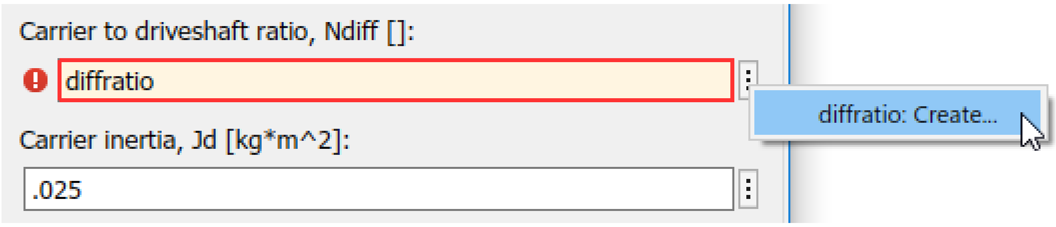
Crown wheel (ring gear) located: To tl

Carrier to driveshaft ratio, Ndiff []: diffratio

Carrier inertia, Jd [kg*m^2]: .025

Carrier damping, bd [N*m*s/rad]:

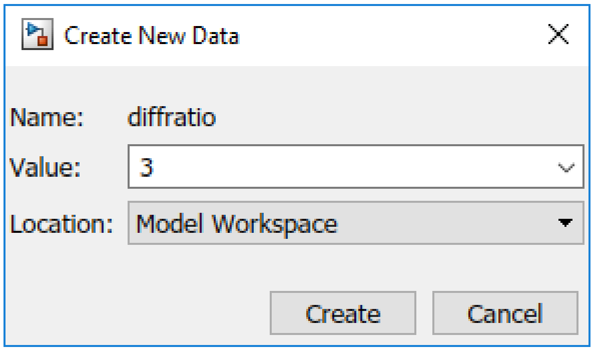
- Use the available actions to create new data.



Carrier to driveshaft ratio, Ndiff []: diffratio

Carrier inertia, Jd [kg*m^2]: .025

- Use the Create New Data dialog box to create a Model Workspace parameter `diffratio` equal to a value of `3`.



Create New Data

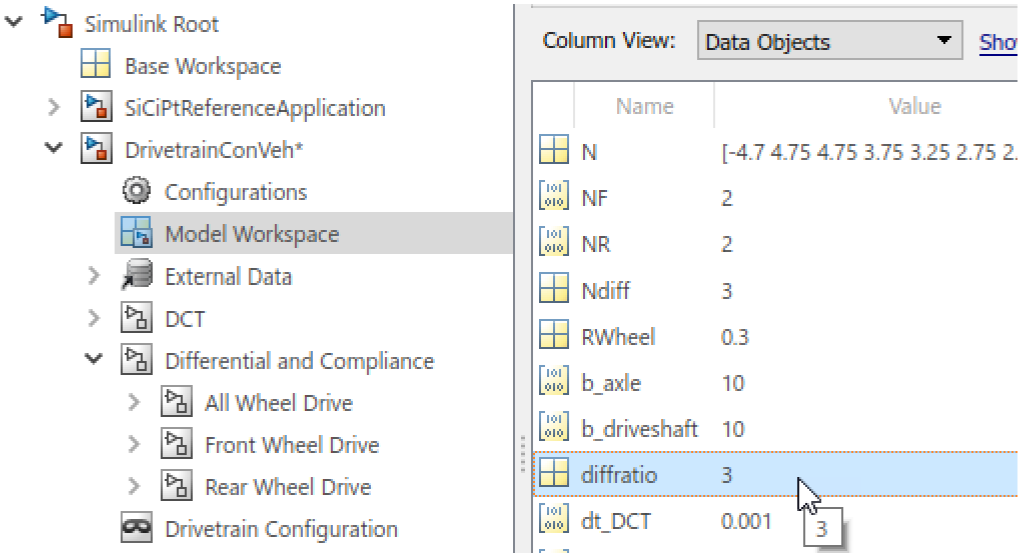
Name: diffratio

Value: 3

Location: Model Workspace

- In the Open Differential block mask, apply the change.

c In the Model Explorer, for the `DrivetrainConVeh` model, confirm that the `diffratio` parameter is set to `3`.



Column View: Data Objects

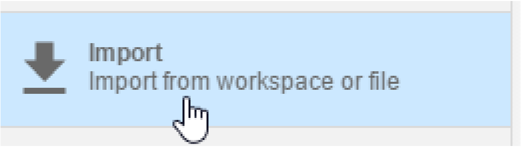
Name	Value
N	[-4.7 4.75 4.75 3.75 3.25 2.75 2.
NF	2
NR	2
Ndiff	3
RWheel	0.3
b_axle	10
b_driveshaft	10
diffratio	3
dt_DCT	0.001

- **d** Save the `DrivetrainConVeh` and `SiCiPtReferenceApplication` models.

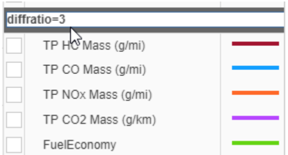
4 Run a baseline conventional vehicle simulation with a final drive gear ratio of `3`. Import the results to the Simulation Data Inspector.

- **a** In the `SiCiPtReferenceApplication` model, run the simulation for the default run time. The simulation can take time to run. View progress in the Simulink window.
- **b** On the Simulink Editor toolbar, click the **Data Inspector** button to open the Simulation Data Inspector.

  - **i** In the Simulation Data Inspector, select **Import**. In the Import dialog box, accept the defaults and select **Import**.



  - **ii** In the results field for the run, right-click to rename the run `diffratio=3`.



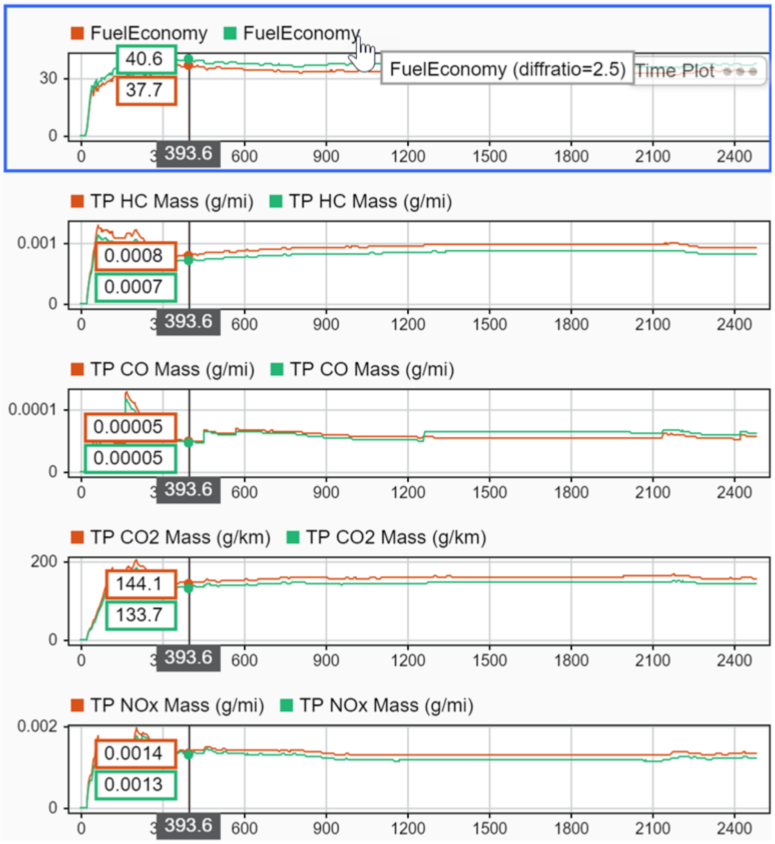
5 Run a conventional vehicle simulation with a final drive gear ratio of `2.5`. Import the results to the Simulation Data Inspector.

- **a** In the Model Explorer, for the `DrivetrainConVeh` model, set the Model Workspace `diffratio` parameter to `2.5`.
- **b** Save the `DrivetrainConVeh` model.
- **c** In the `SiCiPtReferenceApplication` model, run the simulation for the default run time.
- **d** To import the results, on the toolbar, select the Simulation Data Inspector.

- i In the Simulation Data Inspector, select **Import**. In the Import dialog box, accept the defaults and select **Import**.
- ii In the Simulation Data Inspector, in the results field for the run, right-click to rename the run `diffratio=2.5`.

6 Use the Simulation Data Inspector to explore the results. To assess the impact of the final drive gear ratio on the fuel economy and tailpipe emissions, view the plots of the simulation results. For example, these simulation results indicate a better powertrain match when the final drive gear ratio is `2.5`:

- Fuel economy increases when the final drive gear ratio changes from `3` to `2.5`.
- Tailpipe emissions (HC, NOx, CO2) decrease when the final drive gear ratio changes from `3` to `2.5`.


🟥 FuelEconomy 🟩 FuelEconomy
40.6
37.7
FuelEconomy (diffratio=2.5)
Time Plot
30
0
393.6
0 600 900 1200 1500 1800 2100 2400
🟥 TP HC Mass (g/mi) 🟩 TP HC Mass (g/mi)
0.001
0
0.0008
0.0007
🟥 TP CO Mass (g/mi) 🟩 TP CO Mass (g/mi)
0.0001
0
0.00005
0.00005
🟥 TP CO2 Mass (g/km) 🟩 TP CO2 Mass (g/km)
200
0
144.1
133.7
🟥 TP NOx Mass (g/mi) 🟩 TP NOx Mass (g/mi)
0.002
0
0.0014
0.0013

Next Steps

Assess the impact of the final drive gear ratio on vehicle performance. Although the fuel economy and tailpipe emissions indicate a better powertrain match when the final drive gear ratio is 2.5, the ratio also impacts performance.

To assess the vehicle performance, examine `0` to `100` km/hr acceleration times for each axle setting. You can use the Drive Cycle Source block to output a constant velocity of (`100/3.6`) m/s.

See Also

Related Examples

- “Conventional Vehicle Spark-Ignition Engine Fuel Economy and Emissions” on page 1-10
- “Conventional Vehicle Powertrain Efficiency” on page 1-15

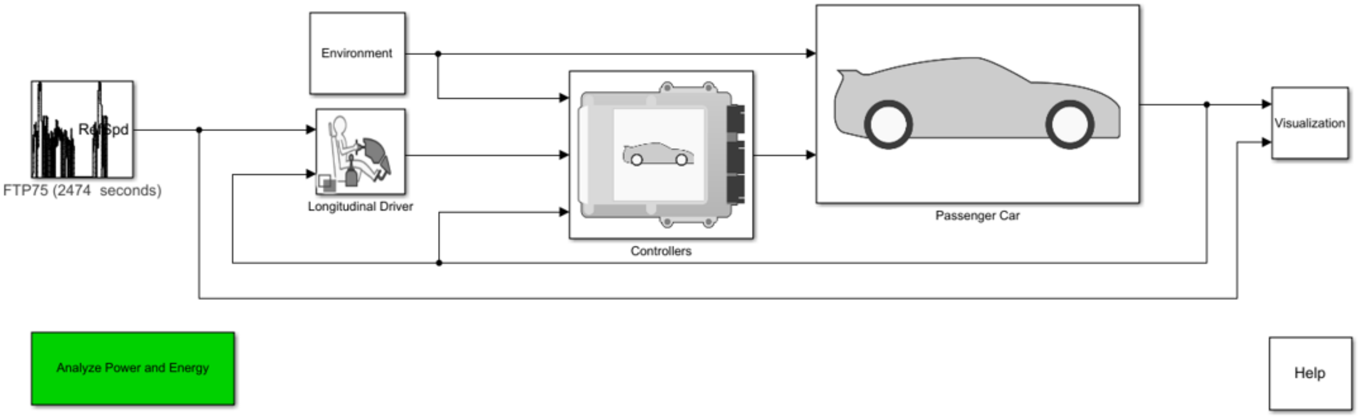
More About

- “Build a Conventional Vehicle Model” on page 3-5
- Simulation Data Inspector

Conventional Vehicle Spark-Ignition Engine Fuel Economy and Emissions

This example shows how to calculate the city and highway fuel economy and the emissions for a conventional vehicle with a 1.5-L spark-ignition (SI) engine. To run this example, make sure you have the city (FTP75) and the highway (HWFET) drive cycles installed. After you open the conventional vehicle reference application, open the Drive Cycle Source block and click **Install additional drive cycles**. For more information, see “Install Drive Cycle Data” on page 5-2.

```
setupconvehMPG;
```



Copyright 2015-2022 The MathWorks, Inc.

Prepare the Conventional Vehicle Reference Application For Simulation

Name the Drive Cycle Source block and Visualization subsystem.

```
model = 'SiCiPtReferenceApplication';
dcs = [model, '/Drive Cycle Source'];
vis_sys = [model, '/Visualization'];
```

In the Visualization subsystem, log the emissions signal data.

```
pt_set_logging([vis_sys, '/Performance Calculations'], 'US MPG', 'Fuel Economy [mpg]', 'both');
pt_set_logging([vis_sys, '/Emission Calculations'], 'TP HC Mass (g/mi)', 'HC [g/mi]', 'both');
pt_set_logging([vis_sys, '/Emission Calculations'], 'TP CO Mass (g/mi)', 'CO [g/mi]', 'both');
pt_set_logging([vis_sys, '/Emission Calculations'], 'TP NOx Mass (g/mi)', 'NOx [g/mi]', 'both');
pt_set_logging([vis_sys, '/Emission Calculations'], 'TP CO2 Mass (g/km)', 'CO2 [g/km]', 'both');
```

Run City Drive Cycle Simulation

Configure the Drive Cycle Source block to run the city drive cycle (FTP75).

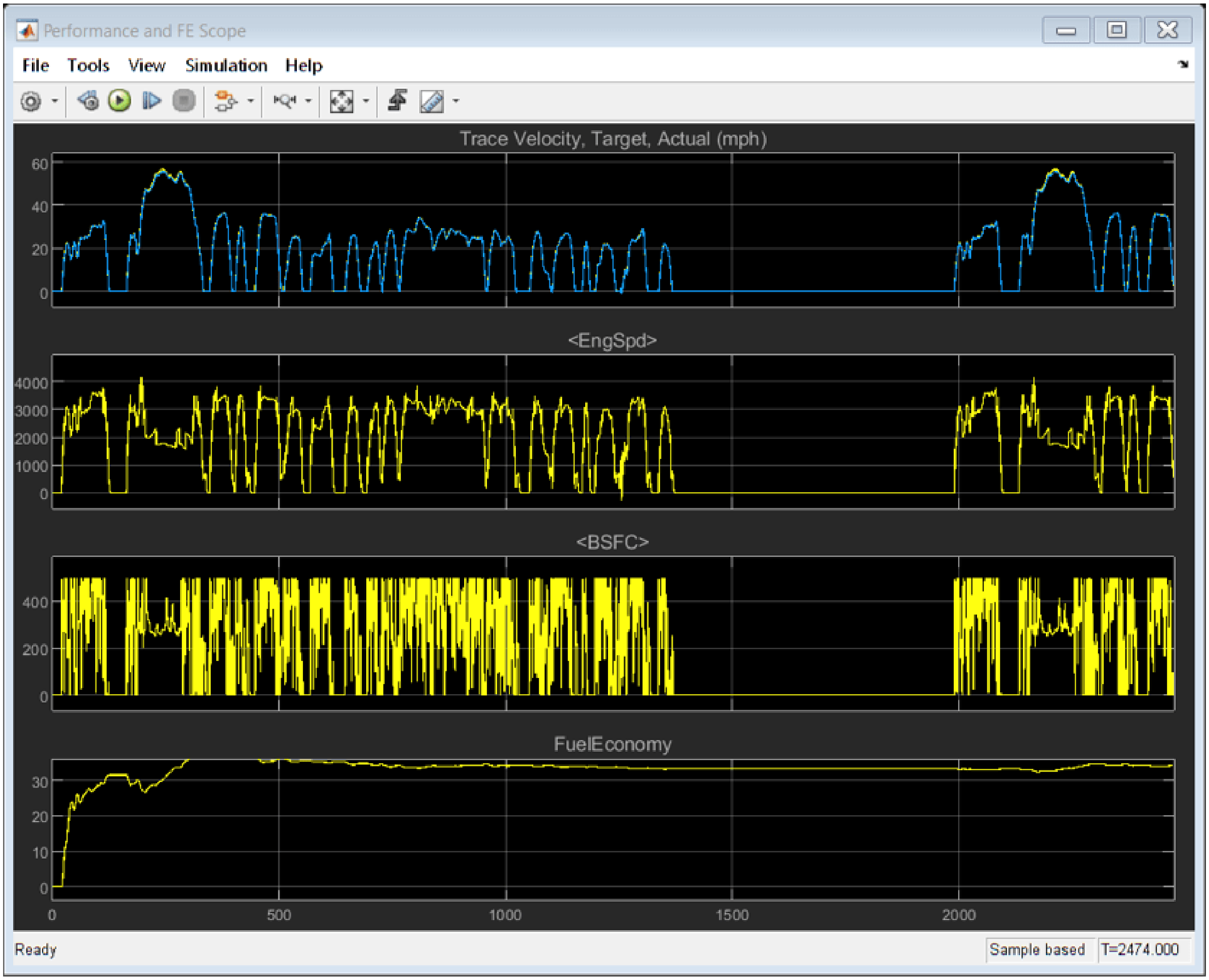
```
set_param(dcs,'cycleVar','FTP75');
```

Run a simulation of the city drive cycle. View the results in the Performance and FE Scope.

```
tfinal = get_param(dcs, 'tfinal');
tf = tfinal(1:strfind(tfinal,' '));
```

```
simout1 = sim(model,'ReturnWorkspaceOutputs','on', 'StopTime', tf);
open_system('SiCiPtReferenceApplication/Visualization/Performance and FE Scope')
```

```
### Starting serial model reference simulation build.
### Model reference simulation target for DrivetrainConVeh is up to date.
### Model reference simulation target for PowertrainBestFuelController is up to date.
### Model reference simulation target for SiEngineController is up to date.
### Model reference simulation target for SiMappedEngine is up to date.

Build Summary

0 of 4 models built (4 models already up to date)
Build duration: 0h 0m 12.229s
```



The results indicate that the fuel economy is approximately 30 mpg at the end of the drive cycle. The scope also provides the target velocity, engine speed, and brake specific fuel consumption (BSFC).

Run Highway Drive Cycle Simulation

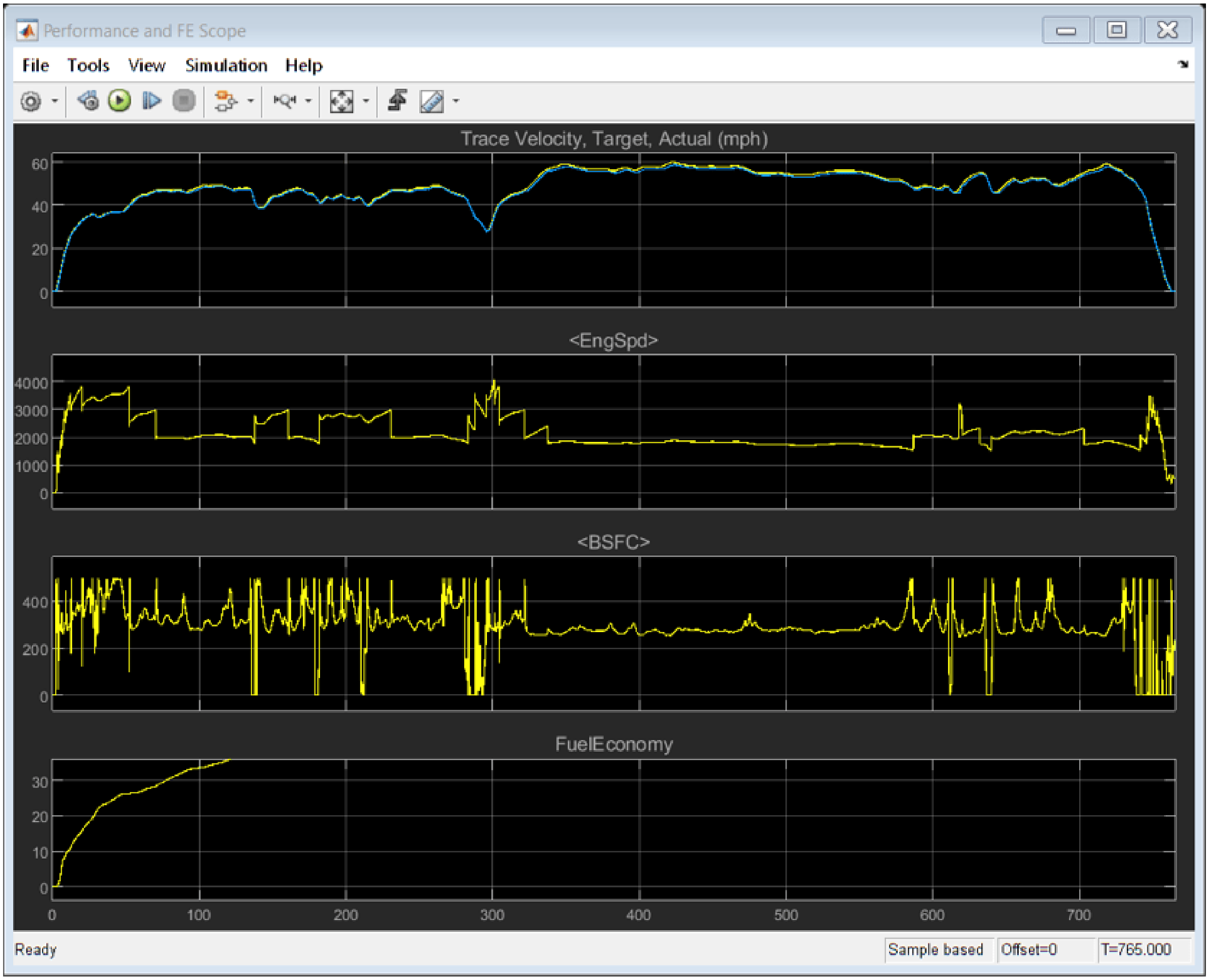
Configure the Drive Cycle Source block to run the highway drive cycle (HWFET). Make sure that you have installed the highway drive cycle.

```
set_param(dcs,'cycleVar','HWFET');
```

Run a simulation of the highway drive cycle. View the results in the Performance and FE Scope.

```
tfinal = get_param(dcs, 'tfinal');
tf = tfinal(1:strfind(tfinal,' '));
simout2 = sim(model,'ReturnWorkspaceOutputs','on', 'StopTime', tf);
open_system('SiCiPtReferenceApplication/Visualization/Performance and FE Scope')
```

```
### Starting serial model reference simulation build.
### Model reference simulation target for DrivetrainConVeh is up to date.
### Model reference simulation target for PowertrainBestFuelController is up to date.
### Model reference simulation target for SiEngineController is up to date.
### Model reference simulation target for SiMappedEngine is up to date.

Build Summary

0 of 4 models built (4 models already up to date)
Build duration: 0h 0m 1.5696s
```



The results indicate that the fuel economy is approximately 34 mpg at the end of the drive cycle. The scope also provides the target velocity, engine speed, and brake specific fuel consumption (BSFC).

Extract Results

Extract the city and highway fuel economy results for the city and highway drive cycles from the logged data.

```
logsout1 = simout1.get('logsout');
FE_urban = logsout1.get('Fuel Economy [mpg]').Values.Data(end);
logsout2 = simout2.get('logsout');
FE_hwy = logsout2.get('Fuel Economy [mpg]').Values.Data(end);
```

Use the city and highway fuel economy results to compute the combined sticker mpg.

```
FE_combined = 0.55*FE_urban + 0.45*FE_hwy;
```

Extract the tailpipe emissions from the city drive cycle.

```
HC = logsout1.get('HC [g/mi]').Values.Data(end);
CO = logsout1.get('CO [g/mi]').Values.Data(end);
NOx = logsout1.get('NOx [g/mi]').Values.Data(end);
CO2 = logsout1.get('CO2 [g/km]').Values.Data(end);
```

Display the fuel economy and city drive cycle tailpipe emissions results in the command window.

```
fprintf('\n***********************\n')
fprintf('FUEL ECONOMY\n');
fprintf('   City:     %4.2f mpg\n', FE_urban);
fprintf('   Highway:  %4.2f mpg\n', FE_hwy);
fprintf('   Combined: %4.2f mpg\n', FE_combined);
fprintf('\nTAILPIPE EMISSIONS\n');
fprintf('   HC:   %4.3f [g/mi]\n',HC);
fprintf('   CO:   %4.3f [g/mi]\n',CO);
fprintf('   NOx:  %4.3f [g/mi]\n',NOx);
fprintf('   CO2:  %4.1f [g/km]\n',CO2);
fprintf('   NMOG: %4.3f [g/mi]',HC+NOx);
fprintf('\n***********************\n');
```

```
***********************
FUEL ECONOMY
   City:     34.26 mpg
   Highway:  45.18 mpg
   Combined: 39.18 mpg

TAILPIPE EMISSIONS
   HC:   0.001 [g/mi]
   CO:   0.000 [g/mi]
   NOx:  0.001 [g/mi]
   CO2:  158.1 [g/km]
   NMOG: 0.002 [g/mi]
***********************
```

See Also

Drive Cycle Source

Related Examples

- “Install Drive Cycle Data” on page 5-2

More About

- “Build a Conventional Vehicle Model” on page 3-5

Conventional Vehicle Powertrain Efficiency

The Powertrain Blockset vehicle reference applications include live scripts that you can run to evaluate and report energy and power losses at the component- and subsystem-level. This example shows how to examine the impact of the conventional vehicle transmission efficiency on the powertrain efficiency.

Running this example requires a Stateflow license. You can install a Stateflow trial license using the Add-On Explorer.

1. Open the conventional vehicle reference application project. By default, the application has a mapped 1.5–L spark-ignition (SI) engine and a dual clutch transmission.

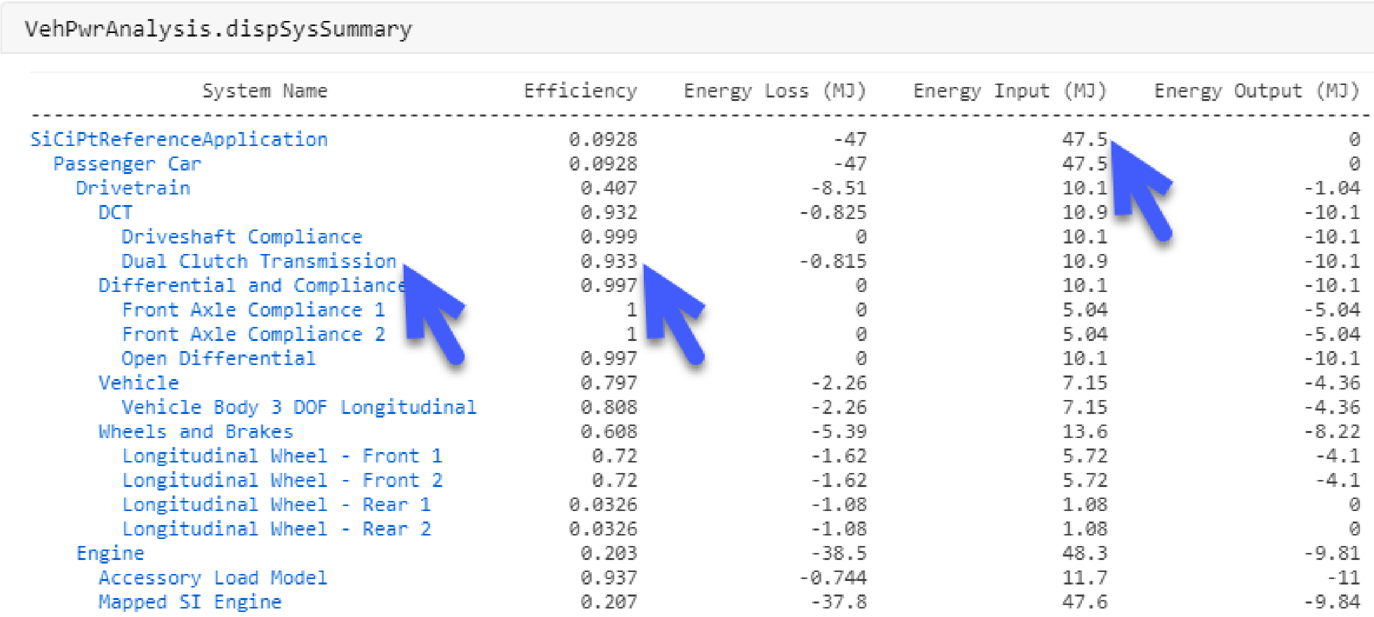
   ```
   autoblkConVehStart
   ```

   Project files open in a writable location.
2. Double-click **Analyze Power and Energy** to open the live script. To generate the energy report, select **Run**.

   The live script provides:

   - An overall energy summary and exported Excel® spreadsheet containing the data. For example, this is similar to the Overall Summary report for the conventional vehicle. The results indicate that the:

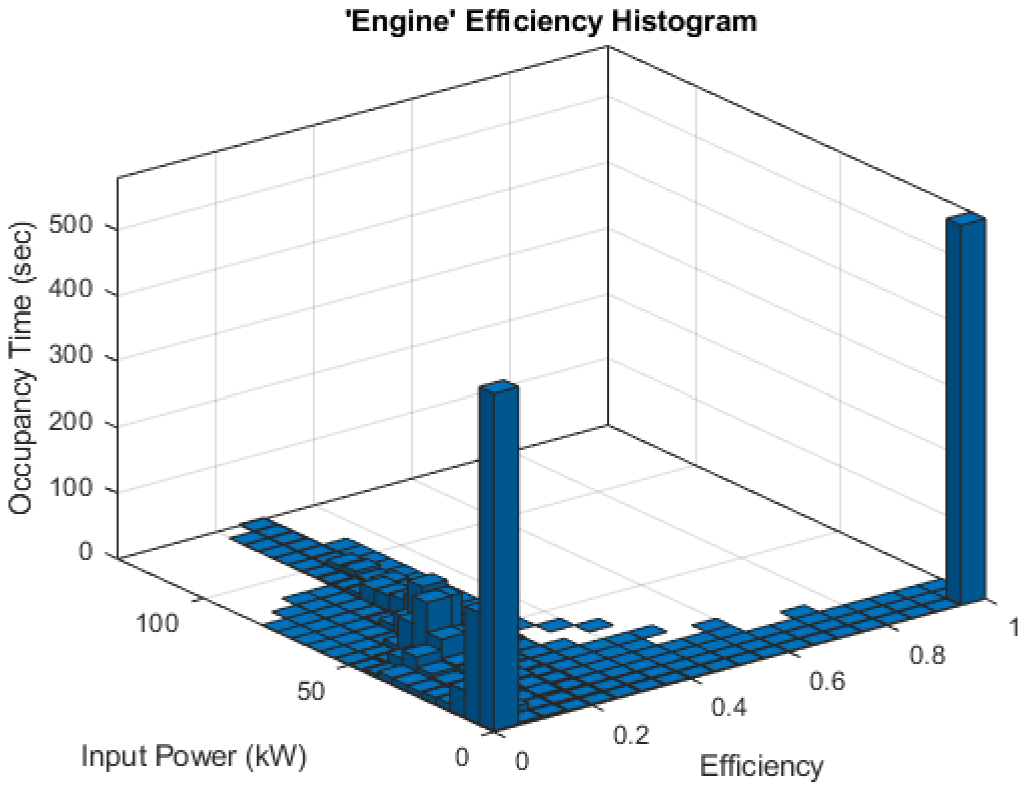
     - Overall powertrain input energy is `47.5` MJ
     - Dual clutch transmission average efficiency is `0.933`



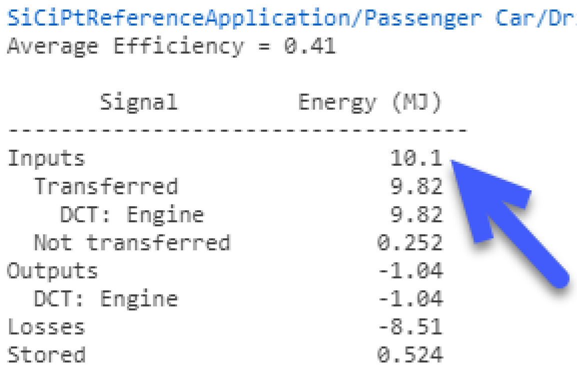
```
VehPwrAnalysis.dispSysSummary
```

```
                System Name                   Efficiency    Energy Loss (MJ)    Energy Input (MJ)    Energy Output (MJ)
-----------------------------------------------------------------------------------------------------------------------
SiCiPtReferenceApplication                      0.0928            -47                  47.5                    0
  Passenger Car                                 0.0928            -47                  47.5                    0
    Drivetrain                                   0.407          -8.51                  10.1                -1.04
      DCT                                        0.932         -0.825                  10.9                -10.1
        Driveshaft Compliance                    0.999              0                  10.1                -10.1
        Dual Clutch Transmission                 0.933         -0.815                  10.9                -10.1
      Differential and Complianc                 0.997              0                  10.1                -10.1
        Front Axle Compliance 1                      1              0                  5.04                -5.04
        Front Axle Compliance 2                      1              0                  5.04                -5.04
        Open Differential                        0.997              0                  10.1                -10.1
      Vehicle                                    0.797          -2.26                  7.15                -4.36
        Vehicle Body 3 DOF Longitudinal          0.808          -2.26                  7.15                -4.36
      Wheels and Brakes                          0.608          -5.39                  13.6                -8.22
        Longitudinal Wheel - Front 1              0.72          -1.62                  5.72                 -4.1
        Longitudinal Wheel - Front 2              0.72          -1.62                  5.72                 -4.1
        Longitudinal Wheel - Rear 1             0.0326          -1.08                  1.08                    0
        Longitudinal Wheel - Rear 2             0.0326          -1.08                  1.08                    0
    Engine                                       0.203          -38.5                  48.3                -9.81
      Accessory Load Model                       0.937         -0.744                  11.7                  -11
      Mapped SI Engine                           0.207          -37.8                  47.6                -9.84
```

   - Engine and drivetrain efficiencies, including an engine histogram of time spent at the different engine efficiencies. For example, this is similar to the engine efficiency histogram for the conventional vehicle.

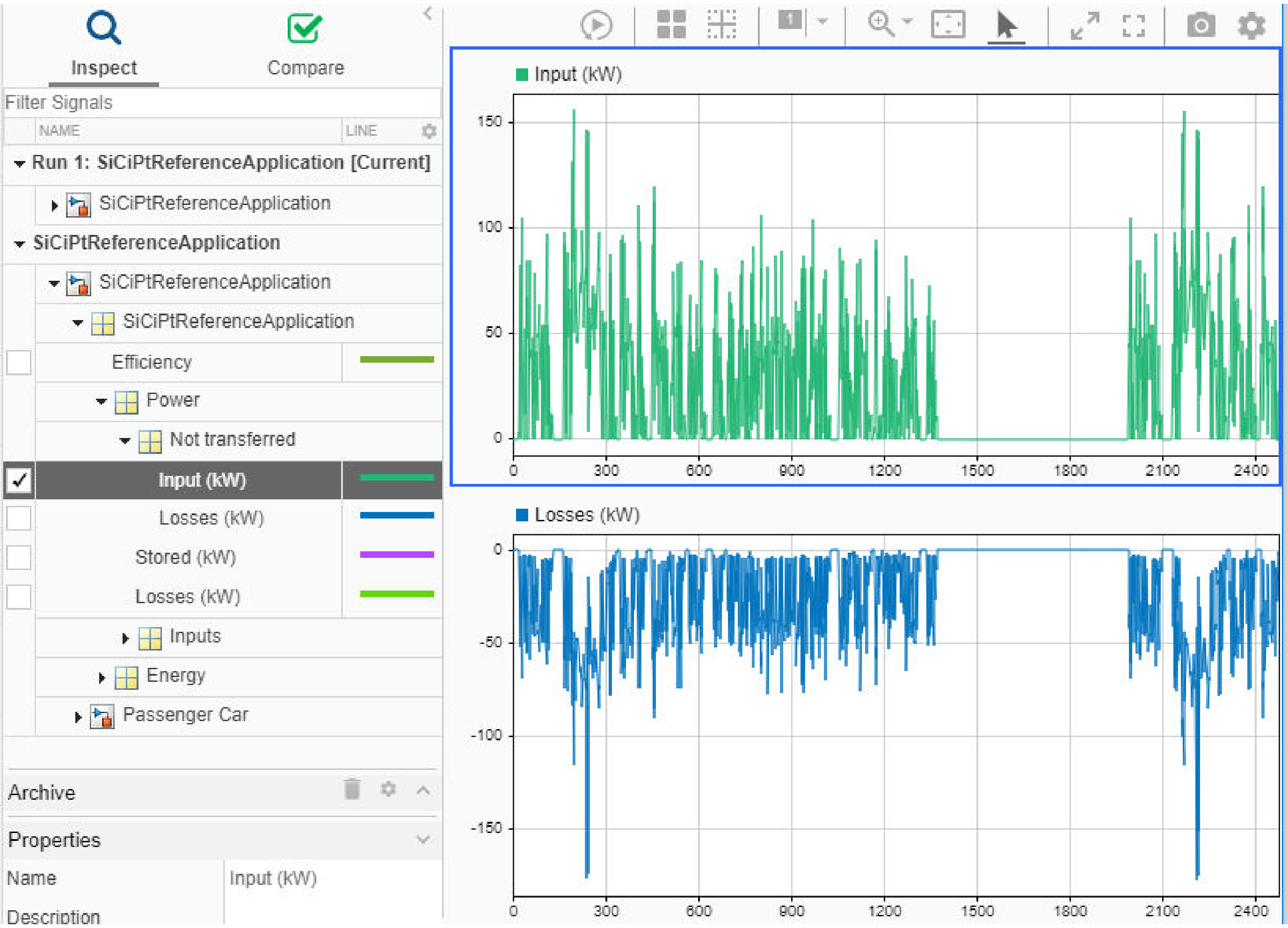


- Drivetrain plant summary that provides the average efficiency, energy input, output, loss, and stored. For example, this is similar to the Drivetrain Plant Summary for the conventional vehicle. The results indicate that the drivetrain input energy is `10.1` MJ.



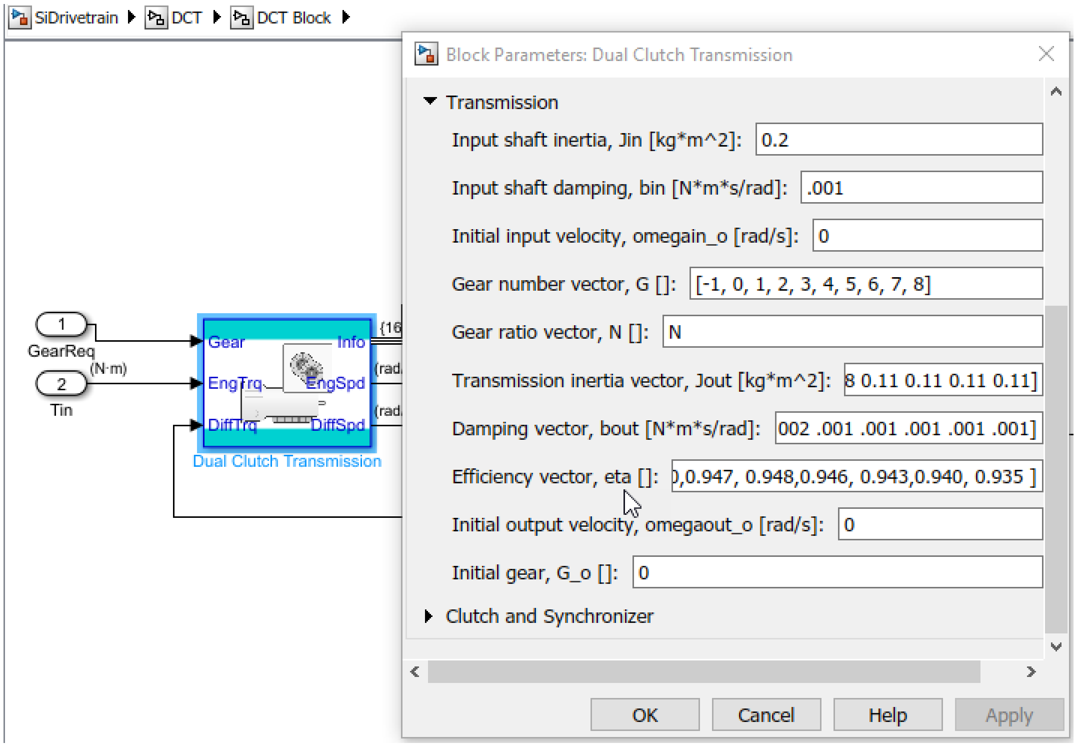
```
SiCiPtReferenceApplication/Passenger Car/Drivetrain
Average Efficiency = 0.41

       Signal         Energy (MJ)
-----------------------------------
Inputs                      10.1
  Transferred               9.82
    DCT: Engine             9.82
  Not transferred          0.252
Outputs                    -1.04
  DCT: Engine              -1.04
Losses                     -8.51
Stored                     0.524
```

- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals. For example, these are similar to the power input and loss plots for the conventional vehicle.



3 In the Overall Summary section of the report:

- **a** Select Dual Clutch Transmission to open the DCT Block subsystem.
- **b** Select the Dual Clutch Transmission block.
- **c** In the block mask, open the **Transmission** parameters.



Block Parameters: Dual Clutch Transmission

Transmission

Input shaft inertia, Jin [kg*m^2]: 0.2

Input shaft damping, bin [N*m*s/rad]: .001

Initial input velocity, omegain_o [rad/s]: 0

Gear number vector, G []: [-1, 0, 1, 2, 3, 4, 5, 6, 7, 8]

Gear ratio vector, N []: N

Transmission inertia vector, Jout [kg*m^2]: 8 0.11 0.11 0.11 0.11]

Damping vector, bout [N*m*s/rad]: 002 .001 .001 .001 .001 .001]

Efficiency vector, eta []:),0.947, 0.948,0.946, 0.943,0.940, 0.935]

Initial output velocity, omegaout_o [rad/s]: 0

Initial gear, G_o []: 0

Clutch and Synchronizer

4 Change the dual clutch transmission so that it is less efficient. By default, the Dual Clutch Transmission block **Efficiency vector, eta** parameter value is `[0.930, 0.930, 0.930, 0.940,0.947, 0.948,0.946, 0.943,0.940, 0.935]`.

  a Set the **Efficiency vector, eta** parameter to `.9*[0.930, 0.930, 0.930, 0.940,0.947, 0.948,0.946, 0.943,0.940, 0.935]`.

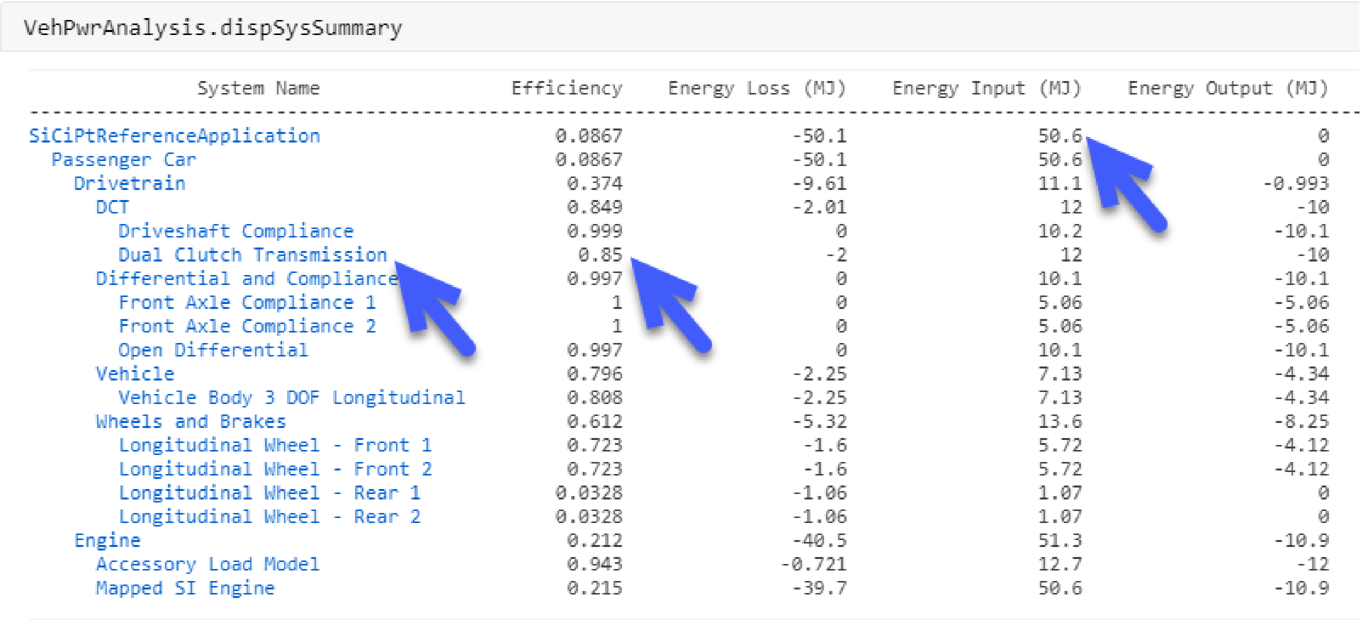
  b Save the `DrivetrainConVeh` model.

5 In the SiCIPtReferenceApplication model window, click **Analyze Power and Energy** to open the live script. To generate the energy report, select **Run**.

6 After you run the live script, in the Overall Summary, examine the efficiencies. For example, these results indicate that the:

- Overall powertrain input energy is `50.6` MJ
- Dual clutch transmission efficiency is `0.85`

When the dual clutch transmission is less efficient, the powertrain requires more energy to complete the drive cycle.



```
VehPwrAnalysis.dispSysSummary
```

```
                System Name                    Efficiency   Energy Loss (MJ)   Energy Input (MJ)   Energy Output (MJ)
----------------------------------------------------------------------------------------------------------------------
SiCiPtReferenceApplication                       0.0867          -50.1               50.6                  0
  Passenger Car                                  0.0867          -50.1               50.6                  0
    Drivetrain                                    0.374          -9.61               11.1             -0.993
      DCT                                         0.849          -2.01                 12                -10
        Driveshaft Compliance                     0.999              0               10.2              -10.1
        Dual Clutch Transmission                   0.85             -2                 12                -10
      Differential and Compliance                 0.997              0               10.1              -10.1
        Front Axle Compliance 1                       1              0               5.06              -5.06
        Front Axle Compliance 2                       1              0               5.06              -5.06
        Open Differential                         0.997              0               10.1              -10.1
      Vehicle                                     0.796          -2.25               7.13              -4.34
        Vehicle Body 3 DOF Longitudinal           0.808          -2.25               7.13              -4.34
      Wheels and Brakes                           0.612          -5.32               13.6              -8.25
        Longitudinal Wheel - Front 1              0.723           -1.6               5.72              -4.12
        Longitudinal Wheel - Front 2              0.723           -1.6               5.72              -4.12
        Longitudinal Wheel - Rear 1              0.0328          -1.06               1.07                  0
        Longitudinal Wheel - Rear 2              0.0328          -1.06               1.07                  0
    Engine                                        0.212          -40.5               51.3              -10.9
      Accessory Load Model                        0.943         -0.721               12.7                -12
      Mapped SI Engine                            0.215          -39.7               50.6              -10.9
```

See Also

`autoblks.pwr.PlantInfo`

Related Examples

- “Getting Started with Powertrain Blockset” on page 1-4
- “Conventional Vehicle Spark-Ignition Engine Fuel Economy and Emissions” on page 1-10

More About

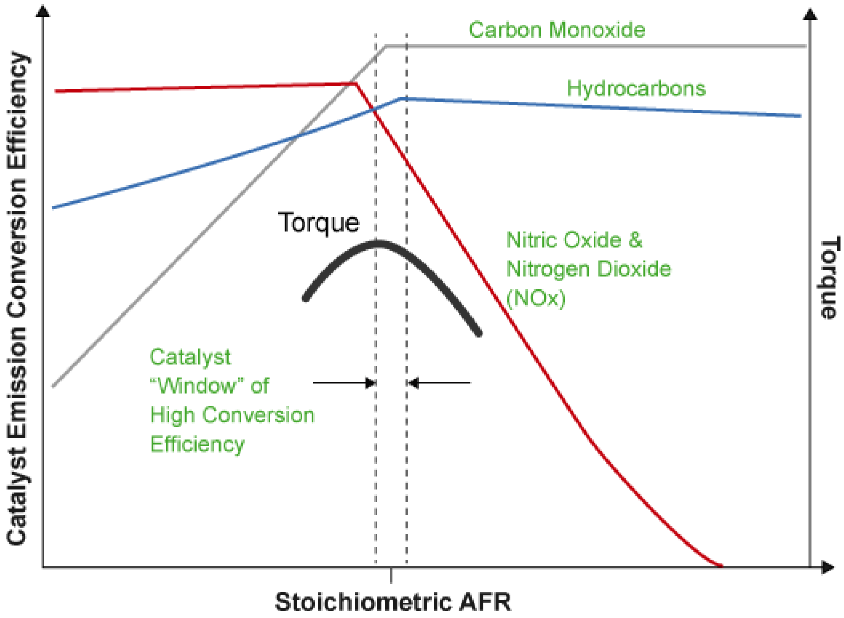
- “Analyze Power and Energy” on page 3-129
- “Build a Conventional Vehicle Model” on page 3-5
- Simulation Data Inspector

Workflows

SI Core Engine Air Mass Flow and Torque Production

A spark-ignition (SI) engine produces torque by controlling the net airflow into the engine using throttle, turbocharger wastegate, and cam-phasing actuators.

While producing torque, the engine must comply with emission standards. To meet the tailpipe emission standards, the ECU operates a three-way-catalyst (TWC) at the stoichiometric air-fuel ratio (AFR).



In addition to emission controls, the ECU:

- Maximizes torque at middle speeds and high loads by operating rich of stoichiometry.
- Limits piston crown temperature at high speeds and high loads by running rich of stoichiometry.

Air Mass Flow Models

To calculate engine air mass flow, configure the SI engine to use either of these air mass flow models.

Air Mass Flow Model	Description
“SI Engine Speed-Density Air Mass Flow Model” on page 2-11	Uses the speed-density equation to calculate the engine air mass flow, relating the engine air mass flow to the intake manifold pressure and engine speed. Consider using this air mass flow model in engines with fixed valvetrain designs.

Air Mass Flow Model	Description
“SI Engine Dual-Independent Cam Phaser Air Mass Flow Model” on page 2-5	To calculate the engine air mass flow, the dual-independent cam phaser model uses: • Empirical calibration parameters developed from engine mapping measurements • Desktop calibration parameters derived from engine computer-aided design (CAD) data In contrast to typical embedded air mass flow calculations based on direct air mass flow measurement with an air mass flow (MAF) sensor, this air mass flow model offers: • Elimination of MAF sensors in dual cam-phased valvetrain applications • Reasonable accuracy with changes in altitude • Semiphysical modeling approach • Bounded behavior • Suitable execution time for electronic control unit (ECU) implementation • Systematic development of a relatively small number of calibration parameters

Torque Models

To calculate the brake torque, configure the SI engine to use either of these torque models.

Brake Torque Model	Description
“SI Engine Torque Structure Model” on page 2-14	For the structured brake torque calculation, the SI engine uses tables for the inner torque, friction torque, optimal spark, spark efficiency, and lambda efficiency. If you select **Crank angle pressure and torque** on the block **Torque** tab, you can: • Simulate advanced closed-loop engine controls in desktop simulations and on HIL bench, based on cylinder pressure recorded from a model or laboratory test as a function of crank angle. • Simulate driveline vibrations downstream of the engine due to high-frequency crankshaft torsionals. • Simulate engine misfires due to lean operation or spark plug fouling by using the injector pulse width input. • Simulate cylinder deactivation effect (closed intake and exhaust valves, no injected fuel) on individual cylinder pressures, mean-value airflow, mean-value torque, and crank-angle-based torque. • Simulate the fuel-cut effect on individual cylinder pressure, mean-value torque, and crank-angle-based torque.
“SI Engine Simple Torque Model” on page 2-20	For the simple brake torque calculation, the SI engine block uses a torque lookup table map that is a function of engine speed and load.

See Also

SI Controller | SI Core Engine

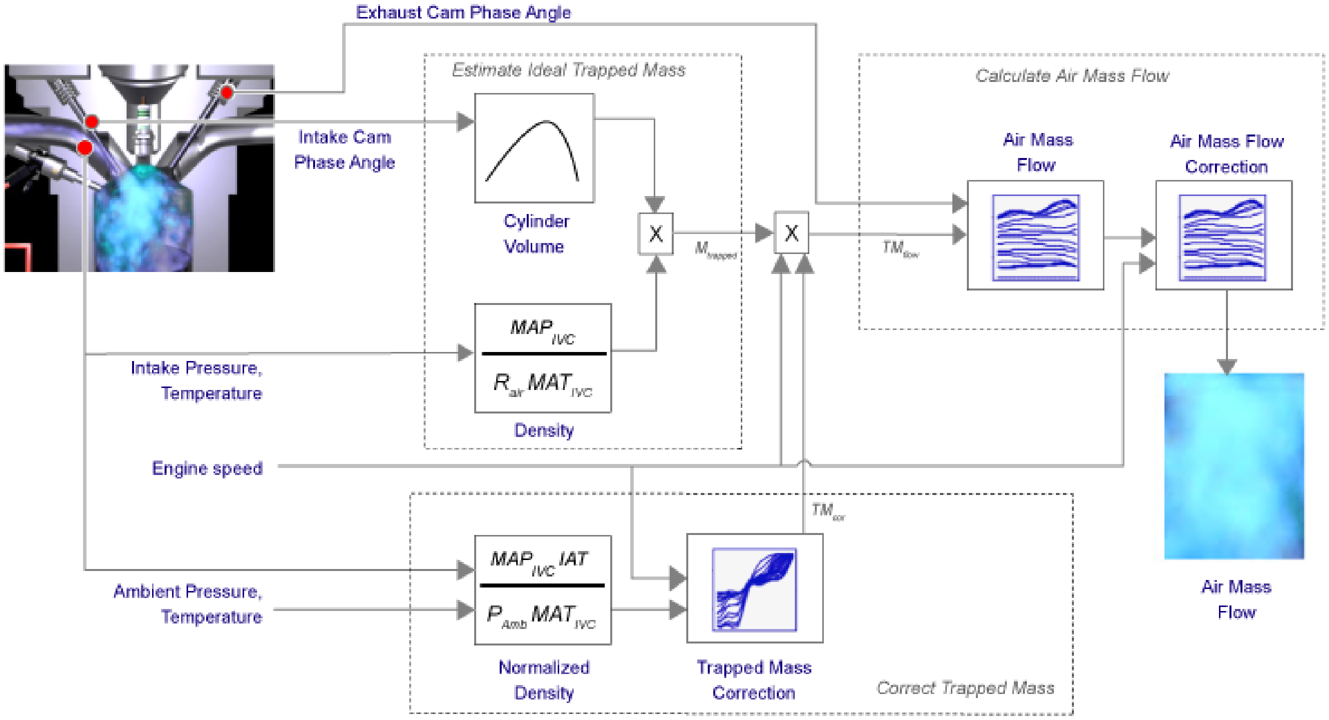
More About

- “Engine Calibration Maps” on page 2-31

SI Engine Dual-Independent Cam Phaser Air Mass Flow Model

To calculate intake air mass flow for an engine equipped with cam phasers, you can configure the spark-ignition (SI) engine with a dual-independent cam phaser intake air mass flow model. As illustrated, the spark-ignition (SI) engine intake air mass flow calculation consists of these steps:

- Collecting physical measurements
- Estimating the ideal trapped mass
- Correcting the trapped mass
- Calculating the intake air mass flow



The dual-independent cam phaser intake air mass flow model implements equations that use these variables.

$M_{trapped}$	Estimated ideal trapped mass
TM_{corr}	Trapped mass correction multiplier
TM_{flow}	Flow rate equivalent to corrected trapped mass at the current engine speed
$\dot{m}_{intkideal}$	Engine intake air mass flow at arbitrary cam phaser angles
$\dot{m}_{intkideal}$	Engine intake port mass flow at arbitrary cam phaser angles
$\dot{m}_{air}$	Engine intake air mass flow final correction at steady-state cam phaser angles
$\dot{m}_{intk}$	Engine intake port mass flow at steady-state cam phaser angles
$y_{intk,air}$	Engine intake manifold air mass fraction

MAP_{IVC}	Intake manifold pressure at IVC
MAT_{IVC}	Intake manifold temperature at IVC
M_{Nom}	Nominal engine cylinder intake air mass at standard temperature and pressure, piston at bottom dead center (BDC) maximum volume
IAT	Intake air temperature
N	Engine speed
N_{cyl}	Number of engine cylinders
V_{IVC}	Cylinder volume at IVC
V_d	Displaced volume
R_{air}	Ideal gas constant
P_{Amb}	Ambient pressure
T_{std}	Standard temperature
P_{std}	Standard pressure
ρ_{norm}	Normalized density
φ_{ICP}	Measured intake cam phaser angle
φ_{ECP}	Exhaust cam phaser angle
L_{ideal}	Engine load (normalized cylinder air mass) at arbitrary cam phaser angles, uncorrected for final steady-state cam phaser angles
L	Engine load (normalized cylinder air mass) at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles
Cps	Crankshaft revolutions per power stroke
f_{Vivc}	Cylinder volume at IVC table
f_{TMcorr}	Trapped mass correction table
$f_{airideal}$	Intake air mass flow table
$f_{aircorr}$	Intake air mass flow correction table

Collect Physical Measurements

In the SI engine model, the dual-independent cam phaser intake air mass flow model requires these physical measurements:

- Intake manifold temperature and pressure at intake valve closing (IVC) condition
- Intake cam phase angle
- Exhaust cam phase angle
- Engine speed
- Ambient pressure and temperature
- Intake air mass flow, from one or more of the following
  - Tank air meter
  - Wide range air-fuel sensor and fuel-flow meter

- Wide range air-fuel sensor and injector pulse-width

Estimate Ideal Trapped Mass

The dual-independent cam phaser intake air mass flow model uses the Ideal Gas Law to estimate the ideal trapped mass at intake manifold conditions. The calculation assumes the cylinder pressure and temperature at IVC equal the intake manifold pressure and temperature.

$$M_{trapped} \cong \frac{MAP_{IVC}V_{IVC}}{R_{air}MAT_{IVC}}$$

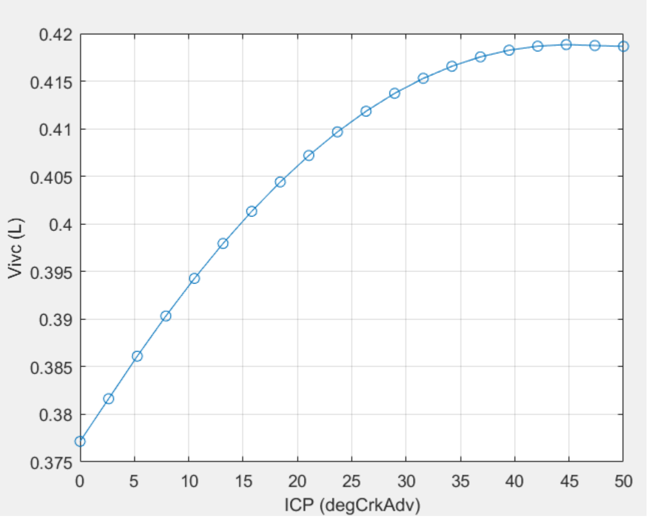
For engines with variable intake cam phasing, the trapped volume at IVC varies.

The cylinder volume at intake valve close table (IVC), f_{Vivc} is a function of the intake cam phaser angle

$$V_{IVC} = f_{Vivc}(\varphi_{ICP})$$

where:

- V_{IVC} is cylinder volume at IVC, in L.
- φ_{ICP} is intake cam phaser angle, in crank advance degrees.



Correct Trapped Mass

The dual-independent cam phaser intake air mass flow model uses a correction factor to account for the difference between the ideal trapped mass in the cylinder and the actual trapped mass. The trapped mass correction factor is a lookup table that is a function of the normalized density and engine speed.

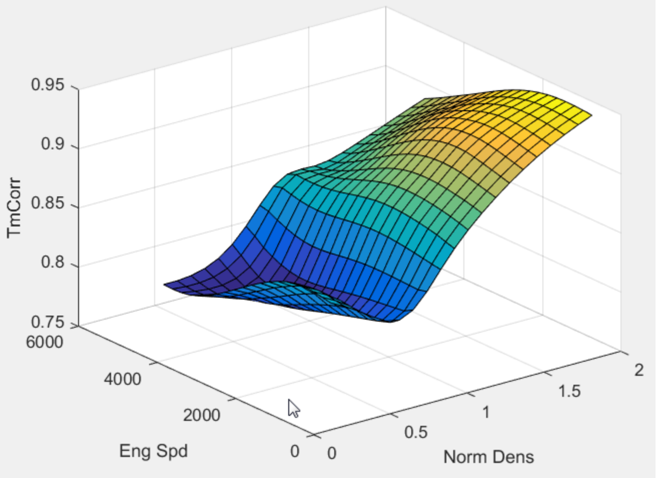
$$\rho_{norm} = \frac{MAP_{IVC}IAT}{P_{Amb}MAT_{IVC}}$$

The trapped mass correction factor table, f_{TMcorr}, is a function of the normalized density and engine speed

$$TM_{corr} = f_{TMcorr}(\rho_{norm}, \quad N)$$

where:

- TM_{corr}, is trapped mass correction multiplier, dimensionless.
- ρ_{norm} is normalized density, dimensionless.
- N is engine speed, in rpm.



- Normalized density accounts for the throttle position independent of a given altitude.
- Engine speed accounts for the pulsation effects of the piston movement.
- Ambient pressure is measured by a sensor on the electronic control unit (ECU) or estimated using an inverse throttle valve model.
- The ECU estimates or measures intake air temperature (IAT) upstream of the throttle.

Trapped mass flow is expressed as a flow rate in grams per second (g/s). The trapped mass flow is the maximum gas mass flow through the engine when no residual gases remain in the cylinder at the end of the exhaust stroke.

$$TM_{flow} = \frac{\left(1000\frac{g}{kg}\right)N_{cyl}TM_{corr}M_{trapped}N}{\left(\frac{60s}{min}\right)Cps}$$

Calculate Air Mass Flow

To determine the engine intake air mass flow at arbitrary cam phase angles, the dual-independent cam phaser air mass flow model uses a lookup table.

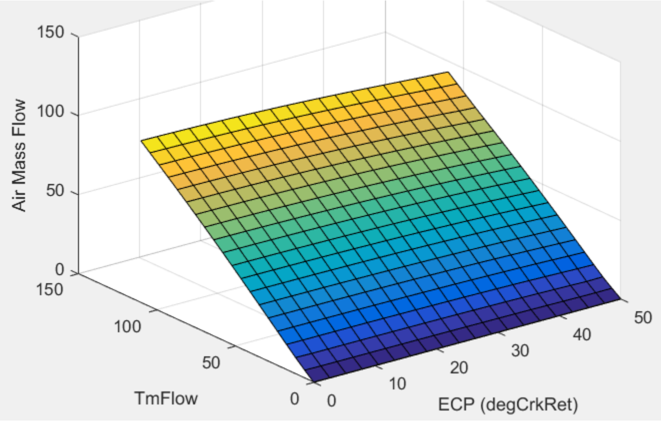
The phaser intake mass flow model lookup table is a function of exhaust cam phaser angles and trapped air mass flow

$$\dot{m}_{intkideal} = f_{intkideal}(\varphi_{ECP}, TM_{flow})$$

where:

- $\dot{m}_{intkideal}$ is engine intake port mass flow at arbitrary cam phaser angles, in g/s.

- φ_{ECP} is exhaust cam phaser angle, in degrees crank retard.
- TM_{flow} is flow rate equivalent to corrected trapped mass at the current engine speed, in g/s.



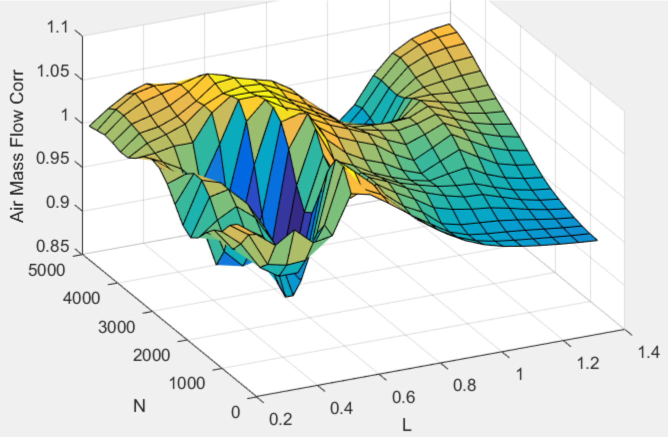
- The exhaust cam phasing has a significant effect on the fraction of burned gas. During the exhaust stroke, exhaust cam-phasing affects the exhaust valve position at exhaust valve closing (EVC) relative to the piston position. A retarded (late) exhaust cam phase angle moves EVC past piston top dead center (TDC), causing the exhaust gas to flow back from the manifold runner into the cylinder. This pull-back triggers the reburn of crevice volume gasses, reducing nitric oxide and nitrogen dioxide emissions (NOx) via charge temperature reduction and hydrocarbon (HC) emissions. Exhaust temperature and back pressure affect exhaust gas back-flow and exhaust cam phaser timing. Exhaust gas temperature and pressure correlate to trapped mass flow. Since at least 80% of trapped mass flow is unburned air, air mass flow is highly correlated to trapped mass flow.
- The unburned air mass flow determines the engine load and open-loop fuel control to achieve a target air-fuel ratio (AFR).
- The lookup table allows arbitrary cam phaser position combinations that can occur during transient engine operations when the phasers are moving from one target position to another.

The intake air mass flow correction lookup table, $f_{aircorr}$, is a function of ideal load and engine speed

$$\dot{m}_{air} = \dot{m}_{intkideal} f_{aircorr}(L_{ideal}, N)$$

where:

- L_{ideal} is engine load (normalized cylinder air mass) at arbitrary cam phaser angles, uncorrected for final steady-state cam phaser angles, dimensionless.
- N is engine speed, in rpm.
- $\dot{m}_{air}$ is engine intake air mass flow final correction at steady-state cam phaser angles, in g/s.
- $\dot{m}_{intkideal}$ is engine intake port mass flow at arbitrary cam phaser angles, in g/s.



- To calculate the engine intake port mass flow, the engine model uses this equation.

$$\dot{m}_{intk} = \frac{\dot{m}_{air}}{y_{intk,\,air}}$$

- Ideal load is the normalized engine cylinder unburned intake air mass before the final correction. To calculate ideal load, the model divides the unburned intake air mass by the nominal cylinder intake air mass. The nominal cylinder intake air mass is the intake air mass (kg) in a cylinder at piston bottom dead center (BDC) with air at standard temperature and pressure:

$$M_{Nom} = \frac{P_{std}V_d}{N_{cyl}R_{air}T_{std}}$$

$$L_{ideal} = \frac{\left(\frac{60s}{min}\right)Cps\dot{m}_{intkideal}y_{intk,\,air}}{\left(\frac{1000g}{kg}\right)N_{cyl}NM_{Nom}}$$

- The final engine load is expressed by

$$L = \frac{\left(\frac{60s}{min}\right)Cps\dot{m}_{air}}{\left(\frac{1000g}{Kg}\right)N_{cyl}NM_{Nom}}$$

See Also

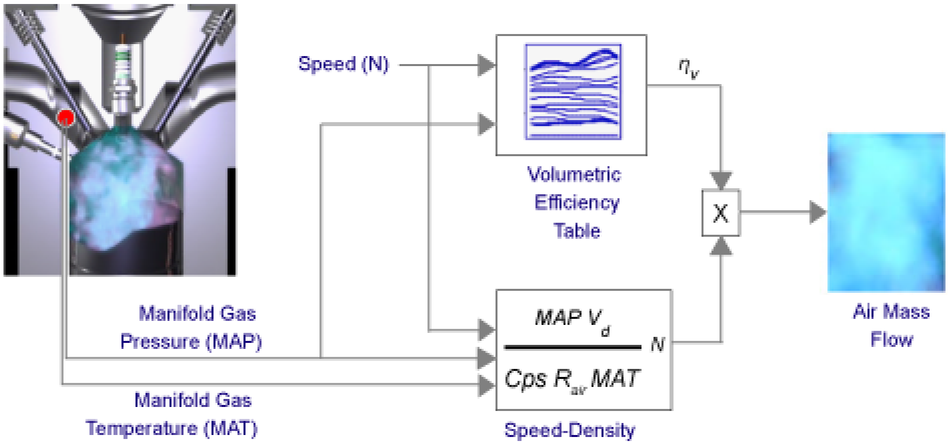
SI Controller | SI Core Engine

More About

- “SI Core Engine Air Mass Flow and Torque Production” on page 2-2
- “SI Engine Speed-Density Air Mass Flow Model” on page 2-11
- “Engine Calibration Maps” on page 2-31

SI Engine Speed-Density Air Mass Flow Model

To calculate the air mass flow in the spark-ignition (SI) engine, you can configure the Spark Ignition Core Engine block to use a speed-density air mass flow model. The speed-density model uses the speed-density equation to calculate the engine air mass flow. The equation relates the engine air mass flow to the intake manifold gas pressure, intake manifold gas temperature, and engine speed. Consider using this air mass flow model in simple conventional engine designs, where variable valvetrain technologies are not in use.


Speed (N)
η_v
Volumetric Efficiency Table
X
Manifold Gas Pressure (MAP)
$\frac{MAP\ V_d}{Cps\ R_{air}\ MAT} N$
Manifold Gas Temperature (MAT)
Speed-Density
Air Mass Flow

To determine the air mass flow, the speed-density air mass flow model applies these speed-density equations at the intake manifold gas pressure and gas temperature states.

$$\dot{m}_{intk} = \frac{MAPV_dN\left[\frac{1\quad min}{60s}\right]}{CpsR_{air}MAT}\eta_v$$

$$\dot{m}_{air} = y_{intk,\,air}\dot{m}_{intk}$$

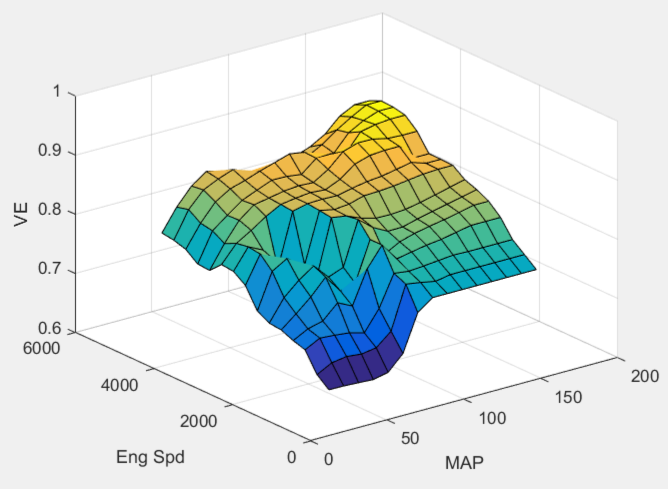
The speed-density air mass flow model uses a volumetric efficiency lookup table to correct the ideal air mass flow.

The engine volumetric efficiency lookup table, f_{η_v}, is a function of intake manifold absolute pressure and engine speed

$$\eta_v = f_{\eta_v}(MAP, N)$$

where:

- η_v is engine volumetric efficiency, dimensionless.
- *MAP* is intake manifold absolute pressure, in KPa.
- *N* is engine speed, in rpm.



To develop the volumetric efficiency table, use the measured air mass flow rate, intake manifold gas pressure, intake manifold gas temperature, and engine speed from engine performance testing.

$$\eta_v = \frac{CpsR_{air}MAT}{MAPV_dN\left[\frac{1\quad min}{60s}\right]}\dot{m}_{air}$$

The air mass flow model implements equations that use these variables.

MAP	Cycle average intake manifold pressure
$\dot{m}_{intk}$	Engine intake port mass flow
$\dot{m}_{air}$	Engine intake air mass flow
V_d	Displaced volume
N	Engine speed
Cps	Crankshaft revolutions per power stroke
MAT	Cycle average intake manifold gas absolute temperature
R_{air}	Ideal gas constant for air and burned gas mixture
f_{η_v}	Engine volumetric efficiency lookup table
η_v	Engine volumetric efficiency

References

[1] Heywood, John B. *Internal Combustion Engine Fundamentals*. New York: McGraw-Hill, 1988.

See Also

SI Controller | SI Core Engine

More About

- “SI Core Engine Air Mass Flow and Torque Production” on page 2-2

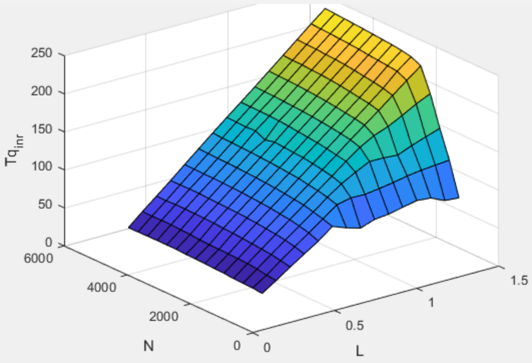
- “SI Engine Dual-Independent Cam Phaser Air Mass Flow Model” on page 2-5
- “Engine Calibration Maps” on page 2-31

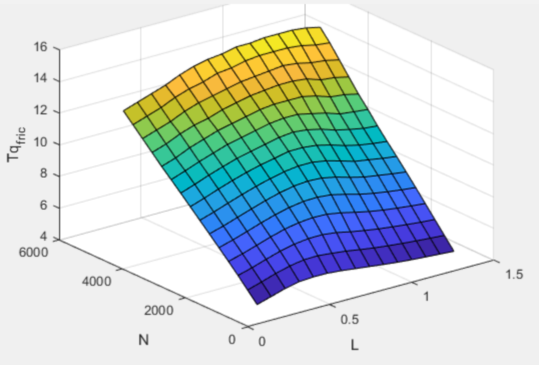
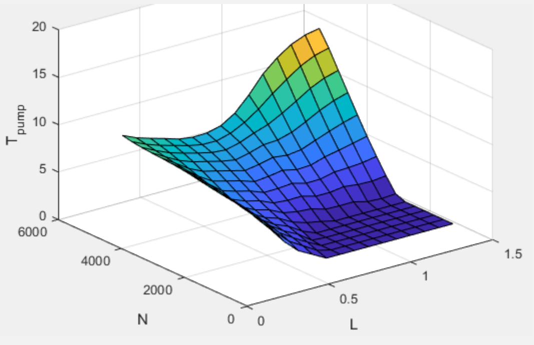
SI Engine Torque Structure Model

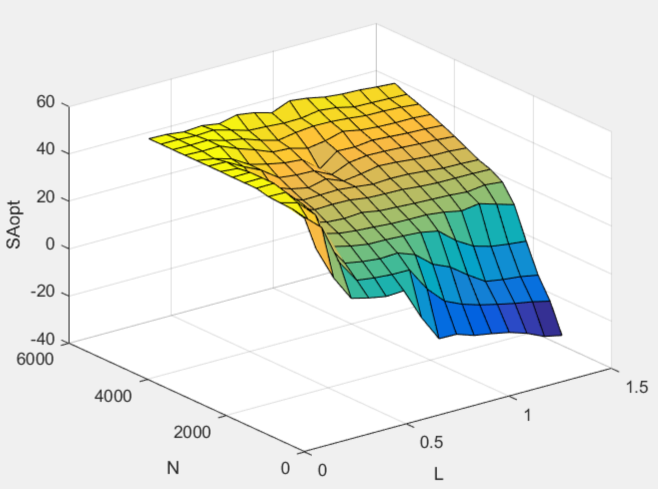
The spark-ignition (SI) engine implements a simplified version of the SI engine torque structure calculation used in a Bosch Engine Management System (EMS). For the torque structure estimation calculation, the block requires calibration tables for:

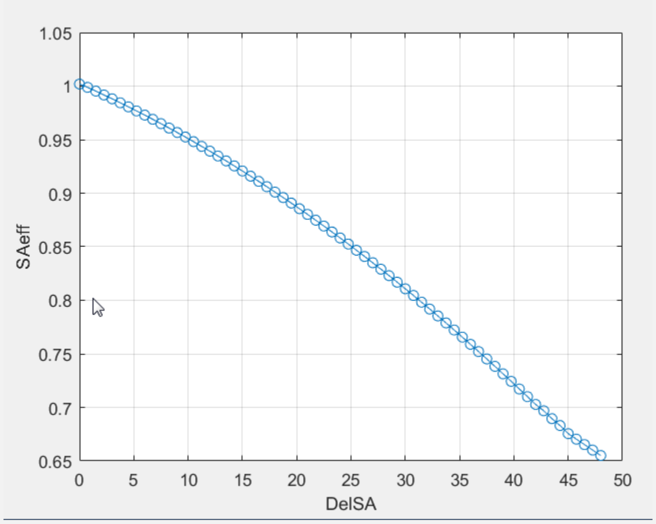
- Inner torque — Maximum torque potential of the engine at a given speed and load
- Friction torque — Torque losses due to friction
- Optimal spark — Spark advance for optimal inner torque
- Spark efficiency — Torque loss due to spark retard from optimal
- Lambda efficiency — Torque loss due to lambda change from optimal
- Pumping torque — Torque loss due to pumping

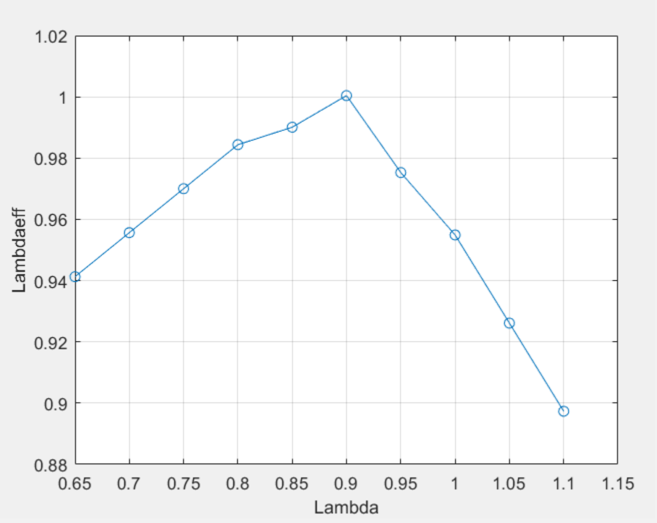
The tables available with Powertrain Blockset were developed with the Model-Based Calibration Toolbox.

Lookup Table	Used to Determine	Plot
Inner torque, f_{Tqinr}	$Tq_{inr} = f_{Tqinr}(L, N)$	The inner torque lookup table, f_{Tqinr}, is a function of engine speed and engine load, $Tq_{inr} = f_{Tqinr}(L, N)$, where: • Tq_{inr} is inner torque based on gross indicated mean effective pressure, in N·m. • L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. • N is engine speed, in rpm. 

Lookup Table	Used to Determine	Plot
Friction torque, f_{Tfric}	$T_{fric} = f_{Tfric}(L, N)$	The friction torque lookup table, f_{Tfric}, is a function of engine speed and engine load, $T_{fric} = f_{Tfric}(L, N)$, where: - T_{fric} is friction torque offset to inner torque, in N·m. - L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. - N is engine speed, in rpm. 
Pumping torque, f_{Tpump}	T_{pump}=$f_{Tpump}(L,N)$	The pumping work lookup table, f_{Tpump}, is a function of engine load and engine speed, T_{pump}=$f_{Tpump}(L,N)$, where: - T_{pump} is pumping work, in N·m. - L is engine load, as a normalized cylinder air mass, dimensionless. - N is engine speed, in rpm. 

Lookup Table	Used to Determine	Plot
Optimal spark, f_{SAopt}	$SA_{opt} = f_{SAopt}(L, N)$	The optimal spark lookup table, f_{SAopt}, is a function of engine speed and engine load, $SA_{opt} = f_{SAopt}(L, N)$, where: - SA_{opt} is optimal spark advance timing for maximum inner torque at stoichiometric air-fuel ratio (AFR), in deg. - L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. - N is engine speed, in rpm. 

Lookup Table	Used to Determine	Plot
Spark efficiency, f_{Msa}	$M_{sa} = f_{Msa}(\Delta SA)$ $\Delta SA = SA_{opt} - SA$	The spark efficiency lookup table, f_{Msa}, is a function of the spark retard from optimal $M_{sa} = f_{Msa}(\Delta SA)$ $\Delta SA = SA_{opt} - SA$ where: • M_{sa} is the spark retard efficiency multiplier, dimensionless. • $\Delta SA$is the spark retard timing distance from optimal spark advance, in deg. 

Lookup Table	Used to Determine	Plot
Lambda efficiency, $f_{M\lambda}$	$M_\lambda = f_{M\lambda}(\lambda)$	The lambda efficiency lookup table, $f_{M\lambda}$, is a function of lambda, $M_\lambda = f_{M\lambda}(\lambda)$, where: • M_λ is the lambda multiplier on inner torque to account for the air-fuel ratio (AFR) effect, dimensionless. • λ is lambda, AFR normalized to stoichiometric fuel AFR, dimensionless. 

The engine brake torque is a based on inner torque with lambda efficiency, spark retard efficiency multipliers, pumping torque, and a friction torque offset

$$T_{brake} = M_\lambda M_{sa} Tq_{inr} - T_{fric} - T_{pump}$$

To account for thermal effects, the torque structure model corrects the friction torque calculation as a function of coolant temperature.

$$T_{fric} = M_{fric} f_{Tfric}(L, N)$$
$$M_{fric} = f_{fric,\,temp}(T_{coolant})$$

The pumping torque is a function of engine speed and engine speed.

$$T_{pump} = f_{Tpump}(L, N)$$

SA_{opt}	Optimal spark advance timing for maximum inner torque at stoichiometric air-fuel ratio (AFR)
ΔSA	Spark retard timing distance from optimal spark advance
SA	Spark advance timing
L	Engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles
N	Engine speed

M_{λ}	Lambda multiplier on inner torque to account for the AFR effect
λ	Lambda, AFR normalized to stoichiometric fuel AFR
M_{sa}	Spark retard efficiency multiplier
f_{Msa}	Spark efficiency lookup table to account for torque loss due to spark retard from optimal
f_{Tfric}	Friction torque lookup table to account for torque losses due to friction
$f_{M\lambda}$	Lambda efficiency lookup table to account for torque loss due to lambda change from optimal
f_{SAopt}	Optimal spark lookup table, for maximum inner torque as a function of engine speed and load
f_{Tqinr}	Inner torque lookup table, for maximum torque potential of the engine at a given speed and load
T_{brake}	Engine brake torque after accounting for spark advance, AFR, and friction effects
T_{fric}	Friction torque offset to inner torque
Tq_{inr}	Inner torque based on gross indicated mean effective pressure
T_{pump}	Pumping torque
M_{fric}	Friction torque modifier
$T_{coolant}$	Coolant temperature

References

[1] Gerhardt, J., Hönninger, H., and Bischof, H., *A New Approach to Functional and Software Structure for Engine Management Systems – BOSCH ME7*. SAE Technical Paper 980801, 1998.

See Also

SI Controller | SI Core Engine

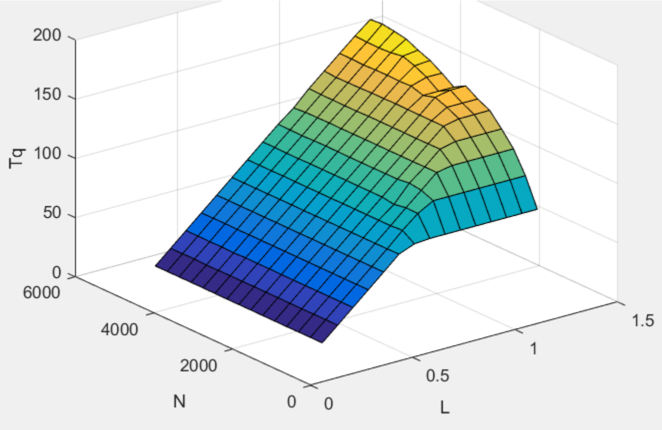
More About

- “SI Core Engine Air Mass Flow and Torque Production” on page 2-2
- “SI Engine Simple Torque Model” on page 2-20

SI Engine Simple Torque Model

For the simple torque lookup table model, the SI engine uses a lookup table map that is a function of engine speed and load, $T_{brake} = f_{TnL}(L, N)$, where:

- T_{brake} is engine brake torque after accounting for spark advance, AFR, and friction effects, in N·m.
- L is engine load, as a normalized cylinder air mass, dimensionless.
- N is engine speed, in rpm.



See Also

SI Controller | SI Core Engine

More About

- “SI Core Engine Air Mass Flow and Torque Production” on page 2-2
- “SI Engine Torque Structure Model” on page 2-14

CI Core Engine Air Mass Flow and Torque Production

A compression-ignition (CI) engine produces mechanical power by injecting fuel into the combustion chamber near the end of the compression stroke. Since the combustion chamber pressure and temperature exceeds the fuel ignition limit, spontaneous ignition occurs after injection. Heat released during combustion increases the cylinder pressure. During the power stroke, the engine converts the pressure to mechanical torque.

Torque production relates to injected fuel mass, fuel injection timing, fuel pressure, and air system states. CI engines operate at lean air-fuel ratio (AFR) conditions, so the AFR is greater than the stoichiometric AFR. CI engines use exhaust gas recirculation (EGR). The exhaust gases recirculate back to the intake manifold, reducing engine-out nitric oxide and nitrogen dioxide (NOx) emissions.

Air Mass Flow

To calculate the air mass flow, the compression-ignition (CI) engine uses the "CI Engine Speed-Density Air Mass Flow Model" on page 2-22. The speed-density model uses the speed-density equation to calculate the engine air mass flow, relating the engine intake port mass flow to the intake manifold pressure, intake manifold temperature, and engine speed.

Torque

To calculate the engine torque, you can configure the block to use either of these torque models.

Brake Torque Model	Description
"CI Engine Torque Structure Model" on page 2-25	The CI core engine torque structure model determines the engine torque by reducing the maximum engine torque potential as these engine conditions vary from nominal: • Start of injection (SOI) timing • Exhaust back-pressure • Burned fuel mass • Intake manifold gas pressure, temperature, and oxygen percentage • Fuel rail pressure To account for the effect of post-inject fuel on torque, the model uses a calibrated torque offset table.
"CI Engine Simple Torque Model" on page 2-30	For the simple engine torque calculation, the CI engine uses a torque lookup table map that is a function of engine speed and injected fuel mass.

See Also

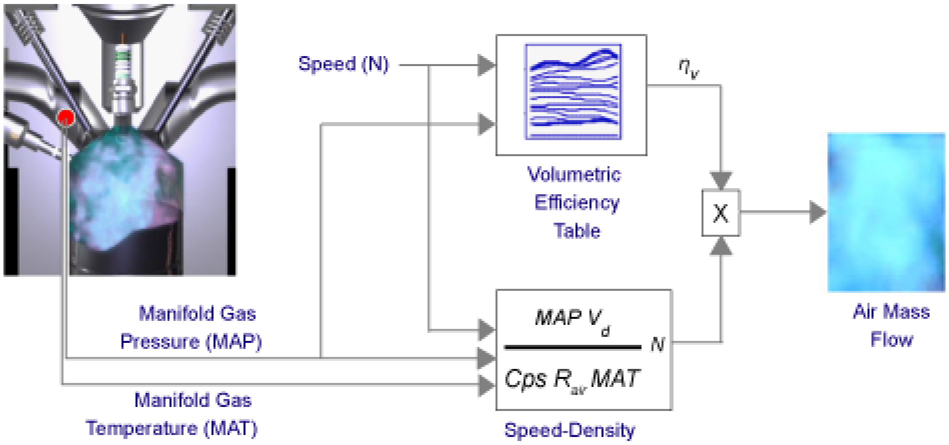
CI Core Engine | CI Controller

More About

- "Engine Calibration Maps" on page 2-31

CI Engine Speed-Density Air Mass Flow Model

To calculate the air mass flow in the compression-ignition (CI) engine, the CI Core Engine block uses a speed-density air mass flow model. The speed-density model uses the speed-density equation to calculate the engine air mass flow. The equation relates the engine air mass flow to the intake manifold gas pressure, intake manifold gas temperature, and engine speed. In the CI Core Engine block, the air mass flow and the cylinder air mass determine the engine load.


Speed (N)
η_v
Volumetric Efficiency Table
X
Air Mass Flow
Manifold Gas Pressure (MAP)
$\frac{MAP\ V_d}{Cps\ R_{air}\ MAT} N$
Manifold Gas Temperature (MAT)
Speed-Density

To determine the air mass flow, the speed-density air mass flow model uses this speed-density equation at the intake manifold and the volumetric efficiency. The model subtracts the exhaust gas recirculation (EGR) burned gas from the mass flow at the intake port.

$$\dot{m}_{port} = \frac{MAPV_dN\left[\frac{1 \quad min}{60s}\right]}{CpsR_{air}MAT}\eta_v$$

$$\dot{m}_{air} = \dot{m}_{port} - \dot{m}_{egr}$$

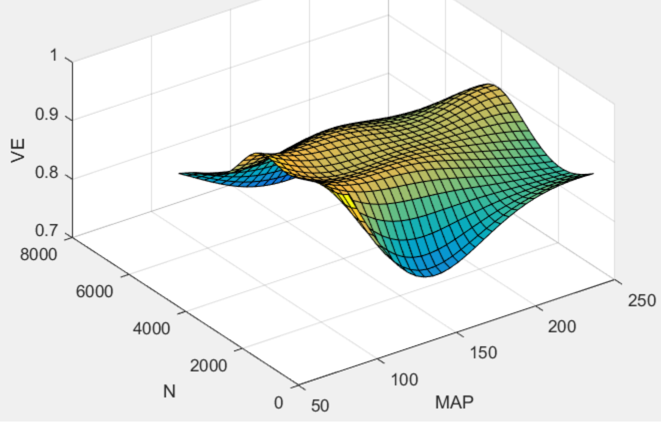
The speed-density air mass flow model uses a volumetric efficiency lookup table to determine the volumetric efficiency.

The volumetric efficiency lookup table is a function of the intake manifold absolute pressure at intake valve closing (IVC) and engine speed

$$\eta_v = f_{\eta_v}(MAP, N)$$

where:

- η_v is engine volumetric efficiency, dimensionless.
- *MAP* is intake manifold absolute pressure, in KPa.
- *N* is engine speed, in rpm.



To create the volumetric efficiency table, use the air mass flow rate from measured engine performance data and the speed-density equation.

$$\eta_v = \frac{Cps R_{air} MAT}{MAP V_d N\left[\frac{1 \quad min}{60s}\right]} \dot{m}_{air}$$

To calculate the engine load, the block divides the calculated unburned air mass by the nominal cylinder air mass. The nominal cylinder air mass is the mass of air (in kg) in a cylinder with the piston at bottom dead center (BDC), at standard air temperature and pressure.

$$M_{Nom} = \frac{P_{std} V_d}{N_{cyl} R_{air} T_{std}}$$

$$L = \frac{\left(\frac{60s}{min}\right) Cps \dot{m}_{air}}{\left(\frac{1000g}{kg}\right) N_{cyl} N M_{Nom}}$$

The model implements equations that use these variables.

$\dot{m}_{air}$	Engine air mass flow
MAP	Cycle average intake manifold pressure
$\dot{m}_{port}$	Total engine air mass flow at intake ports, including EGR flow
$\dot{m}_{egr}$	Recirculated burned gas mass flow entering engine intake port
V_d	Displaced volume
N	Engine speed
Cps	Crankshaft revolutions per power stroke
R_{air}	Ideal gas constant for air and burned gas mixture
MAT	Cycle average intake manifold gas absolute temperature
η_v	Engine volumetric efficiency
f_{η_v}	Engine volumetric efficiency lookup table
L	Engine load (normalized cylinder air mass) at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles

M_{Nom} Nominal engine cylinder air mass at standard temperature and pressure; piston at bottom dead center (BDC) maximum volume

P_{std} Standard pressure

T_{std} Standard temperature

References

[1] Heywood, John B. *Internal Combustion Engine Fundamentals*. New York: McGraw-Hill, 1988.

See Also

CI Core Engine | CI Controller

More About

- “CI Core Engine Air Mass Flow and Torque Production” on page 2-21
- “Engine Calibration Maps” on page 2-31

CI Engine Torque Structure Model

The CI core engine torque structure model determines the engine torque by reducing the maximum engine torque potential as these engine conditions vary from nominal:

- Start of injection (SOI) timing
- Exhaust back-pressure
- Burned fuel mass
- Intake manifold gas pressure, temperature, and oxygen percentage
- Fuel rail pressure

To account for the effect of post-inject fuel on torque, the model uses a calibrated torque offset table.

To determine the engine torque, the CI core engine torque structure model implements the equations specified in these steps.

Step	Description
Step 1: Determine nominal engine inputs and states	Model uses lookup tables to determine these nominal engine inputs and states as a function of compression stroke injected fuel mass, *F*, and engine speed, *N*: • Main start of injection timing, $SOI = f_{SOIc}(F,N)$ • Intake manifold gas temperature, $MAT = f_{MAT}(F,N)$ • Intake manifold gas pressure, $MAP = f_{MAP}(F,N)$ • Intake manifold oxygen percentage, $O2PCT = f_{O2}(F,N)$ • Fuel rail pressure, $FUELP = f_{fuelp}(F,N)$
Step 2: Calculate relative engine states	To determine these relative engine states, the model calculates deviations from their nominal values. • Main start of injection timing delta, $\Delta SOI_c= f_{SOI}(F,N)- SOI$ • Intake manifold gas temperature delta, $\Delta MAT = f_{MAT}(F,N) - MAT$ • Intake manifold oxygen percentage delta, $\Delta O2PCT = f_{O2}(F,N) - O2PCT$ • Fuel rail pressure delta, $\Delta FUELP = f_{fuelp}(F,N) - FUELP$ For the intake manifold gas pressure, the block uses a pressure ratio to determine the relative state. The pressure ratio is the intake manifold gas pressure to the steady-state operating point gas pressure. $MAP_{ratio} = \frac{MAP}{f_{MAP}(F,N)}$

Step	Description
Step 3: Determine efficiency multipliers	Model uses gross indicated mean effective pressure (IMEPG)[1] efficiency multipliers to reduce the maximum average pressure potential of combustion. The efficiency multipliers are lookup tables that are functions of the relative engine states. • Main start of injection timing efficiency multiplier, $SOI_{eff} = f_{SOIeff}(\Delta SOI,N)$ • Intake manifold gas temperature efficiency multiplier, $MAT_{eff} = f_{MATeff}(\Delta MAT,N)$ • Intake manifold gas pressure efficiency multiplier, $MAP_{eff} = f_{MAPeff}(MAP_{ratio},\lambda)$ • Intake manifold oxygen percentage efficiency multiplier, $O2P_{eff} = f_{O2Peff}(\Delta O2P,N)$ • Fuel rail pressure efficiency multiplier, $FUELP_{eff} = f_{FUELPeff}(\Delta FUELP,N)$
Step 4: Determine indicated mean effective cylinder pressure (IMEP) available for torque production	To determine the IMEP available for torque production, the model implements these equations. $IMEP = SOI_{eff}MAP_{eff}MAT_{eff}O2p_{eff}FUELP_{eff}IMEPG$ $IMEPG = f_{IMEP_g}(F,N)$ The model multiplies the efficiency multipliers from step 3 by the IMEPG. The model implements IMEPG as lookup table that is a function of the compression stroke injected fuel mass, *F*, and engine speed, *N*.
Step 5: Account for losses due to friction	To account for friction effects, the model uses the nominal friction mean effective pressure (FMEP)[1] to implement this equation. $FMEP = f_{FMEP}(F,N)f_{fmod}(T_{oil},N)$ The model implements FMEP as lookup table that is a function of the compression stroke injected fuel mass, *F*, and engine speed, *N*. To account for the temperature effect on friction, the model use a lookup table that is a function of oil temperature, T_{oil}, and *N*.
Step 6: Account for pressure loss due to pumping	To account for pressure losses due to pumping, the model uses the nominal pumping mean effective pressure (PMEP)[1] to implement these equations. $\Delta MAP = f_{MAP}(F,N) - MAP$ $\Delta EMAP = f_{EMAP}(F,N) - EMAP$ $PMEP = f_{PMEP}(F,N) - \Delta MAP + \Delta EMAP$ The model implements MAP and EMAP as lookup tables that are functions of the compression stroke injected fuel mass, *F*, and engine speed, *N*. Under normal operating conditions, PMEP is negative, indicating a loss of cylinder pressure.

Step	Description
Step 7: Account for late fuel injection SOI timing on IMEP	To account for late fuel injection SOI timing on IMEP, $\Delta IMEP_{post}$, the model uses a lookup table that is a function of the effective pressure post inject SOI timing centroid, SOI_{post}, and the post inject mass sum, F_{post}. $\Delta IMEP_{post} = f_{\Delta IMEP_{post}}(SOI_{post}, F_{post})$
Step 8: Calculate engine brake torque	To calculate the engine brake torque, T_{brake}, the model converts the brake mean effective pressure (BMEP)[1] to engine brake torque using these equations. The BMEP calculation accounts for all gross mean effective pressure losses. V_d is displaced cylinder volume. Cps is the number of power strokes per revolution. $BMEP = IMEPG + \Delta IMEP_{post} - FMEP + PMEP$ $T_{brake} = \frac{V_d}{2\pi Cps} BMEP$

Fuel Injection

In the CI Core Engine and CI Controller blocks, you can represent multiple injections with the start of injection (SOI) and fuel mass inputs to the model. To specify the type of injection, use the **Fuel mass injection type identifier** parameter.

Type of Injection	Parameter Value
Pilot	0
Main	1
Post	2
Passed	3

The model considers `Passed` fuel injections and fuel injected later than a threshold to be unburned fuel. Use the **Maximum start of injection angle for burned fuel, f_tqs_f_burned_soi_limit** parameter to specify the threshold.

Percent Oxygen

The model uses this equation to calculate the oxygen percent, $O2p$. $y_{in,air}$ is the unburned air mass fraction.

$$O2p = 23.13 y_{in,\,air}$$

Exhaust Temperature

The exhaust temperature calculation depends on the torque model. For both torque models, the block implements lookup tables.

Torque Model	Description	Equations
`Simple Torque Lookup`	Exhaust temperature lookup table is a function of the injected fuel mass and engine speed.	$T_{exh} = f_{Texh}(F, N)$
`Torque Structure`	The nominal exhaust temperature, *Texh*$_{nom}$, is a product of these exhaust temperature efficiencies: • SOI timing • Intake manifold gas pressure • Intake manifold gas temperature • Intake manifold gas oxygen percentage • Fuel rail pressure • Optimal temperature The exhaust temperature, *Texh*$_{nom}$, is offset by a post temperature effect, ΔT_{post}, that accounts for post and late injections during the expansion and exhaust strokes.	$T_{exhnom} = SOI_{exhteff} MAP_{exhteff} MAT_{exhteff} O2p_{exhteff} FU$ $T_{exh} = T_{exhnom} + \Delta T_{post}$ $SOI_{exhteff} = f_{SOI_{exhteff}}(\Delta SOI, N)$ $MAP_{exhteff} = f_{MAP_{exhteff}}(MAP_{ratio}, \lambda)$ $MAT_{exhteff} = f_{MAT_{exhteff}}(\Delta MAT, N)$ $O2p_{exhteff} = f_{O2p_{exhteff}}(\Delta O2p, N)$ $Texh_{opt} = f_{Texh}(F, N)$

The equations use these variables.

F	Compression stroke injected fuel mass
N	Engine speed
$Texh$	Exhaust manifold gas temperature
$Texh_{opt}$	Optimal exhaust manifold gas temperature
ΔT_{post}	Post injection temperature effect
$Texh_{nom}$	Nominal exhaust temperature
$SOI_{exhteff}$	Main SOI exhaust temperature efficiency multiplier
ΔSOI	Main SOI timing relative to optimal timing
MAP_{exheff}	Intake manifold gas pressure exhaust temperature efficiency multiplier
MAP_{ratio}	Intake manifold gas pressure ratio relative to optimal pressure ratio
λ	Intake manifold gas lambda
MAT_{exheff}	Intake manifold gas temperature exhaust temperature efficiency multiplier
ΔMAT	Intake manifold gas temperature relative to optimal temperature
$O2P_{exheff}$	Intake manifold gas oxygen exhaust temperature efficiency multiplier
$\Delta O2P$	Intake gas oxygen percent relative to optimal
$FUELP_{exheff}$	Fuel rail pressure exhaust temperature efficiency multiplier
$\Delta FUELP$	Fuel rail pressure relative to optimal

References

[1] Heywood, John B. *Internal Engine Combustion Fundamentals*. New York: McGraw-Hill, 1988.

See Also

CI Controller | CI Core Engine

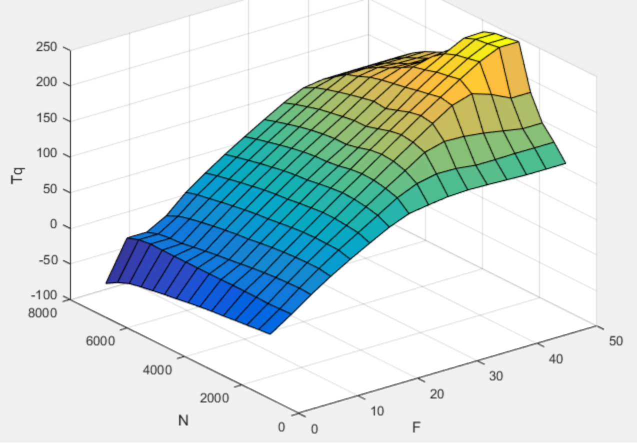
More About

- “CI Core Engine Air Mass Flow and Torque Production” on page 2-21
- “CI Engine Simple Torque Model” on page 2-30

CI Engine Simple Torque Model

For the simple torque lookup table model, the CI engine uses a lookup table is a function of engine speed and injected fuel mass, $T_{brake} = f_{Tnf}(F, N)$, where:

- $Tq = T_{brake}$ is engine brake torque after accounting for engine mechanical and pumping friction effects, in N·m.
- F is injected fuel mass, in mg per injection.
- N is engine speed, in rpm.



See Also

CI Controller | CI Core Engine

More About

- “CI Core Engine Air Mass Flow and Torque Production” on page 2-21
- “CI Engine Torque Structure Model” on page 2-25

Engine Calibration Maps

Calibration maps are a key part of the engine plant and controller models available in the Powertrain 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 and controller 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 and controllers 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 Powertrain Blockset SI and CI reference applications affect the engine response to control inputs (for example, spark timing, throttle position, and cam phasing).

To develop the calibration maps in the Powertrain Blockset 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.

Engine Controller Calibration Maps

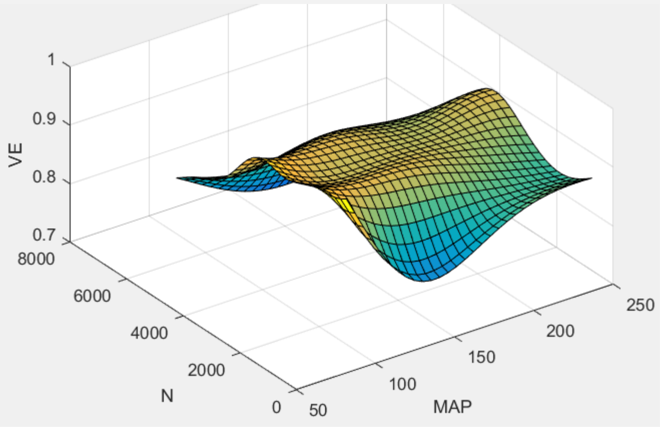
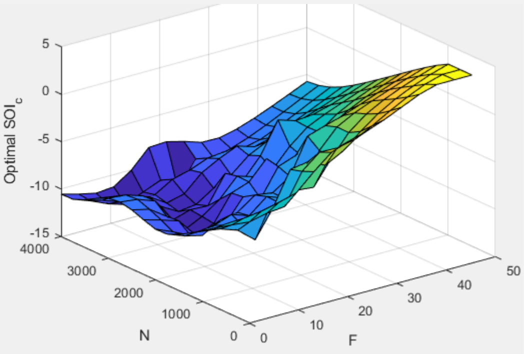
The engine controller model calibration maps in the reference applications represent the optimal open-loop control commands for given engine operating points.

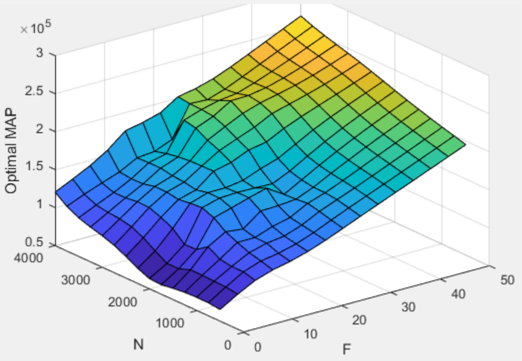
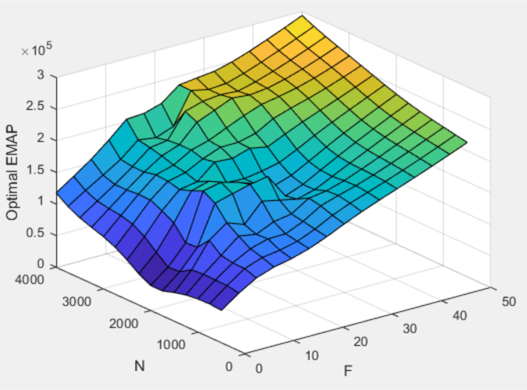
To develop the calibration maps for the SI engine controller, MathWorks used the GT-POWER reference engine models in a virtual engine calibration optimization (VECO) process. The process optimized the open-loop control commands for 1.5–L SI engine, subject to engine operating constraints for knock, turbocharger speed, and exhaust temperature.

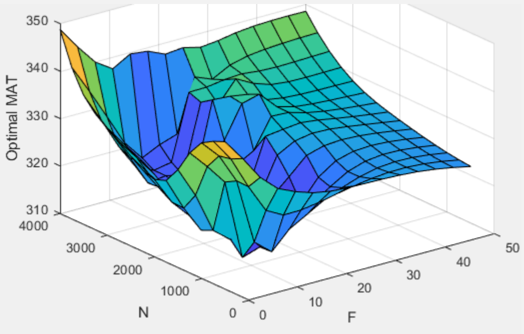
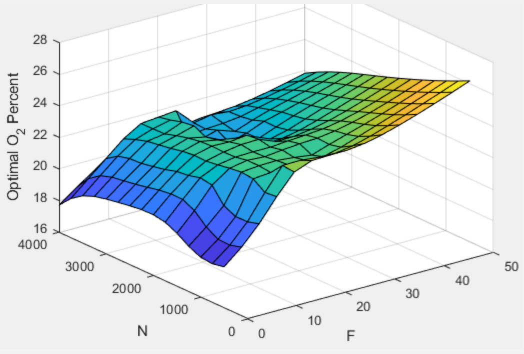
To develop the calibration maps for the CI engine controller, MathWorks used the DOE test data from the GT-POWER 1.5–L CI reference model operated at minimum brake-specific fuel consumption (BSFC).

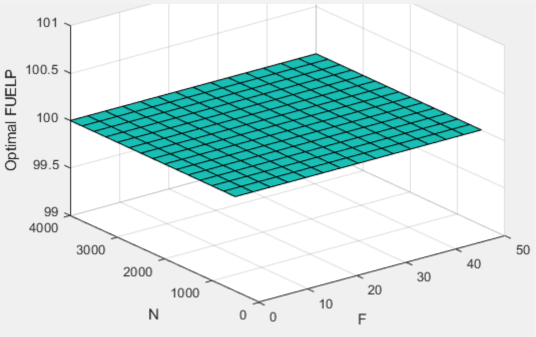
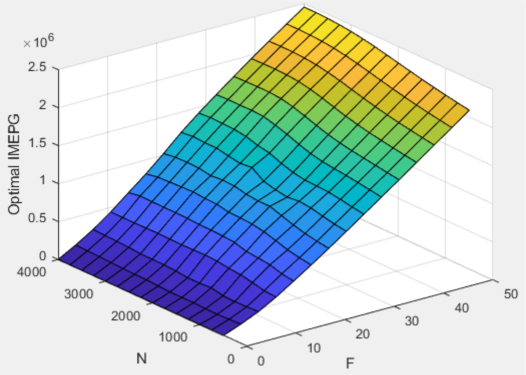
Calibration Maps in Compression-Ignition (CI) Blocks

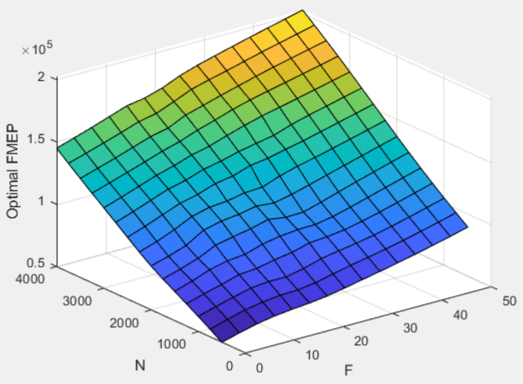
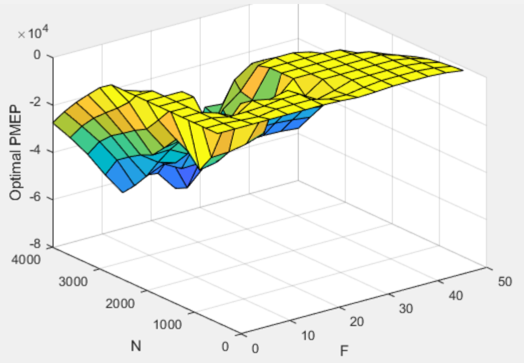
In the engine models, the Powertrain Blockset blocks implement these calibration maps.

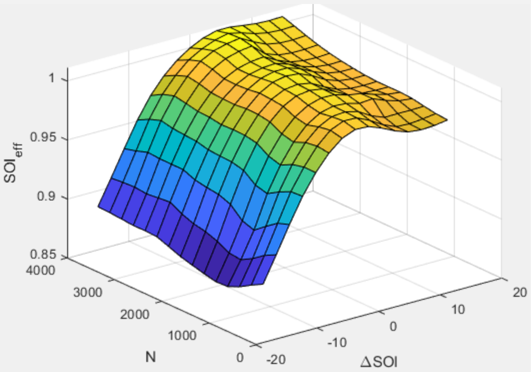
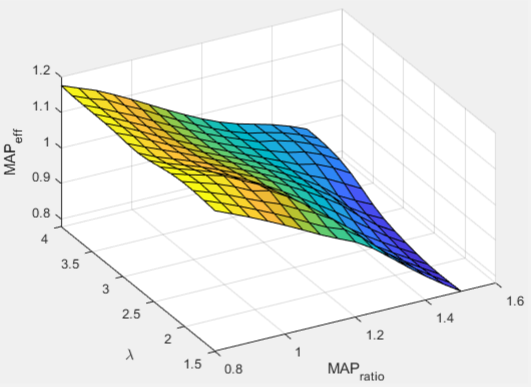
Map	Used For	In	Description
Volumetric efficiency	"CI Engine Speed-Density Air Mass Flow Model" on page 2-22	CI Core Engine CI Controller	The volumetric efficiency lookup table is a function of the intake manifold absolute pressure at intake valve closing (IVC) and engine speed $\eta_v = f_{\eta_v}(MAP, N)$ where: • η_v is engine volumetric efficiency, dimensionless. • *MAP* is intake manifold absolute pressure, in KPa. • *N* is engine speed, in rpm. 
Optimal main start of injection (SOI) timing	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The optimal main start of injection (SOI) timing lookup table, f_{SOIc}, is a function of the engine speed and injected fuel mass, $SOI_c = f_{SOIc}(F,N)$, where: • SOI_c is optimal SOI timing, in degATDC. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 

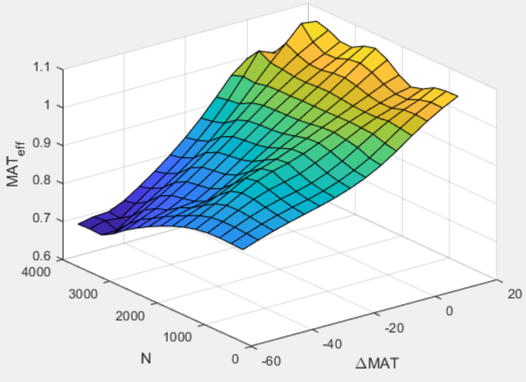
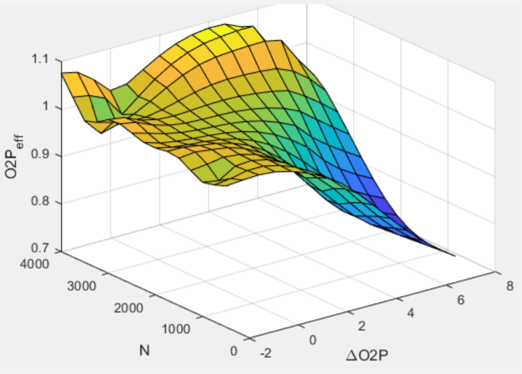
Map	Used For	In	Description
Optimal intake manifold gas pressure	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The optimal intake manifold gas pressure lookup table, f_{MAP}, is a function of the engine speed and injected fuel mass, $MAP = f_{MAP}(F,N)$, where: • *MAP* is optimal intake manifold gas pressure, in Pa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 
Optimal exhaust manifold gas pressure	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The optimal exhaust manifold gas pressure lookup table, f_{EMAP}, is a function of the engine speed and injected fuel mass, $EMAP = f_{EMAP}(F,N)$, where: • *EMAP* is optimal exhaust manifold gas pressure, in Pa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 

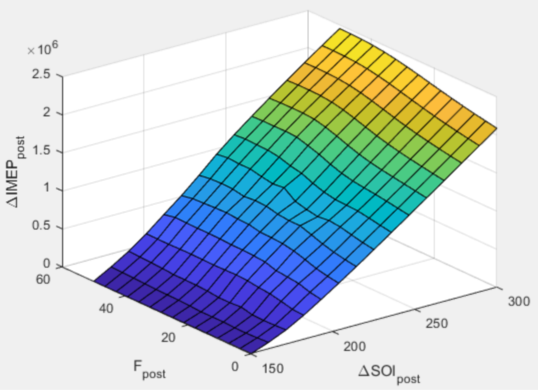
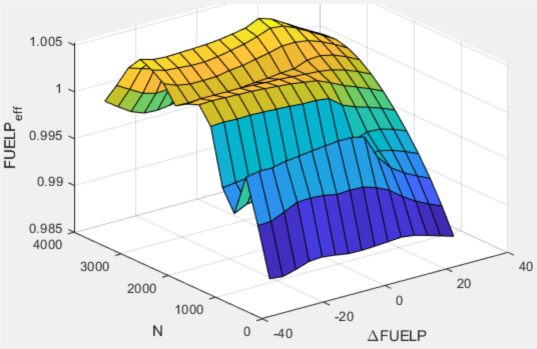
Map	Used For	In	Description
Optimal intake manifold gas temperature	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The optimal intake manifold gas temperature lookup table, f_{MAT}, is a function of the engine speed and injected fuel mass, $MAT = f_{MAT}(F,N)$, where: • *MAT* is optimal intake manifold gas temperature, in K. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 
Optimal intake gas oxygen percent	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The optimal intake gas oxygen percent lookup table, f_{O2}, is a function of the engine speed and injected fuel mass, $O2PCT = f_{O2}(F,N)$, where: • *O2PCT* is optimal intake gas oxygen, in percent. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 

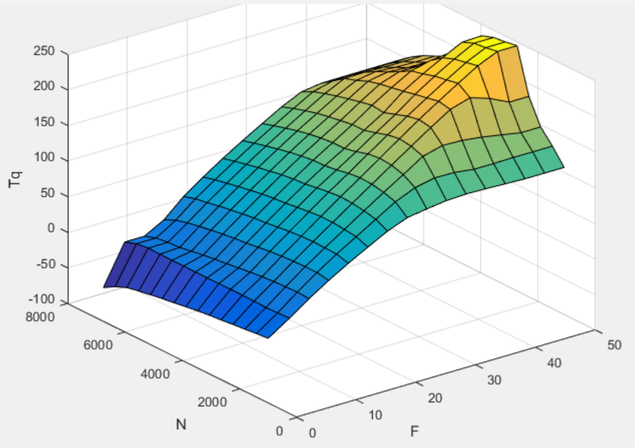
Map	**Used For**	**In**	**Description**
Optimal fuel rail pressure	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The optimal fuel rail pressure lookup table, f_{fuelp}, is a function of the engine speed and injected fuel mass, $FUELP = f_{fuelp}(F,N)$, where: • *FUELP* is optimal fuel rail pressure, in MPa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 
Optimal gross indicated mean effective pressure	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The optimal gross indicated mean effective pressure lookup table, f_{imepg}, is a function of the engine speed and injected fuel mass, $IMEPG = f_{imepg}(F,N)$, where: • *IMEPG* is optimal gross indicated mean effective pressure, in Pa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 

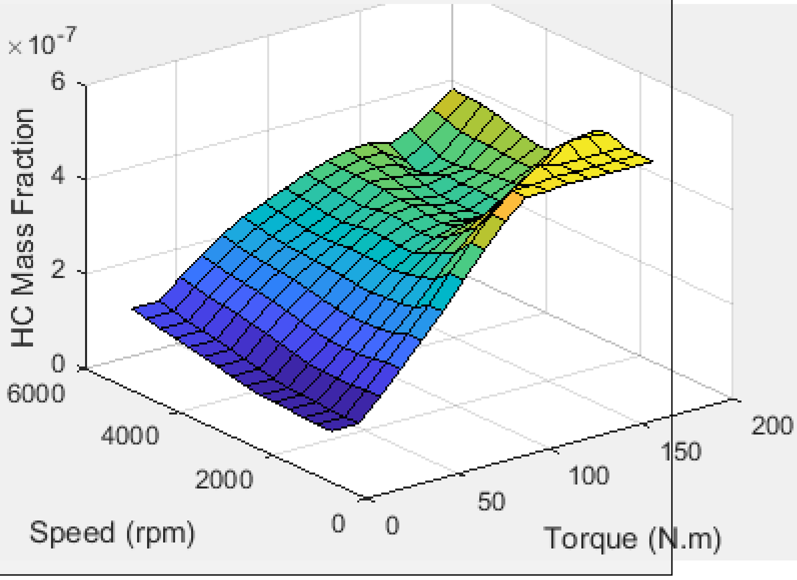
Map	Used For	In	Description
Optimal friction mean effective pressure	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The optimal friction mean effective pressure lookup table, f_{fmep}, is a function of the engine speed and injected fuel mass, $FMEP = f_{fmep}(F,N)$, where: • *FMEP* is optimal friction mean effective pressure, in Pa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 
Optimal pumping mean effective pressure	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The optimal pumping mean effective pressure lookup table, f_{pmep}, is a function of the engine speed and injected fuel mass, $PMEP = f_{pmep}(F,N)$, where: • *PMEP* is optimal pumping mean effective pressure, in Pa. • *F* is compression stroke injected fuel mass, in mg per injection. • *N* is engine speed, in rpm. 

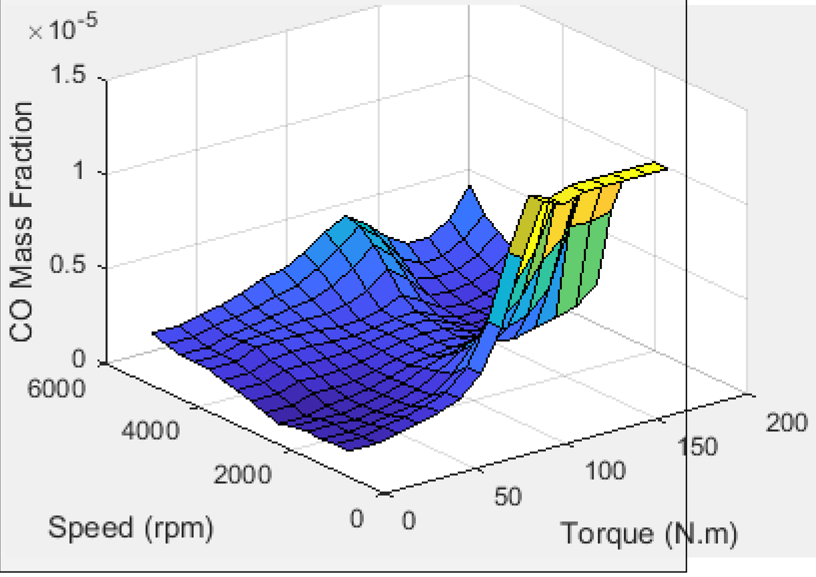
Map	Used For	In	Description
Main SOI timing efficiency multiplier	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The main start of injection (SOI) timing efficiency multiplier lookup table, f_{SOIeff}, is a function of the engine speed and main SOI timing relative to optimal timing, $SOI_{eff} = f_{SOIeff}(\Delta SOI,N)$, where: • SOI_{eff} is main SOI timing efficiency multiplier, dimensionless. • ΔSOI is main SOI timing relative to optimal timing, in degBTDC. • N is engine speed, in rpm. 
Intake manifold gas pressure efficiency multiplier	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The intake manifold gas pressure efficiency multiplier lookup table, f_{MAPeff}, is a function of the intake manifold gas pressure ratio relative to optimal pressure ratio and lambda, $MAP_{eff} = f_{MAPeff}(MAP_{ratio},\lambda)$, where: • MAP_{eff} is intake manifold gas pressure efficiency multiplier, dimensionless. • MAP_{ratio} is intake manifold gas pressure ratio relative to optimal pressure ratio, dimensionless. • λ is intake manifold gas lambda, dimensionless. 

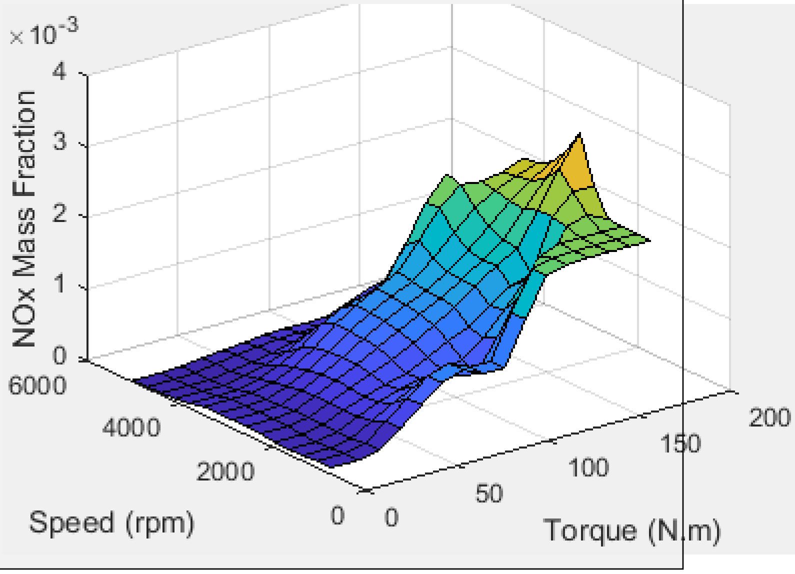
Map	Used For	In	Description
Intake manifold gas temperature efficiency multiplier	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The intake manifold gas temperature efficiency multiplier lookup table, f_{MATeff}, is a function of the engine speed and intake manifold gas temperature relative to optimal temperature, $MAT_{eff} = f_{MATeff}(\Delta MAT,N)$, where: • MAT_{eff} is intake manifold gas temperature efficiency multiplier, dimensionless. • ΔMAT is intake manifold gas temperature relative to optimal temperature, in K. • N is engine speed, in rpm. 
Intake manifold gas oxygen efficiency multiplier	“CI Engine Torque Structure Model” on page 2-25	CI Core Engine CI Controller	The intake manifold gas oxygen efficiency multiplier lookup table, f_{O2Peff}, is a function of the engine speed and intake manifold gas oxygen percent relative to optimal, $O2P_{eff} = f_{O2Peff}(\Delta O2P,N)$, where: • $O2P_{eff}$ is intake manifold gas oxygen efficiency multiplier, dimensionless. • $\Delta O2P$ is intake gas oxygen percent relative to optimal, in percent. • N is engine speed, in rpm. 

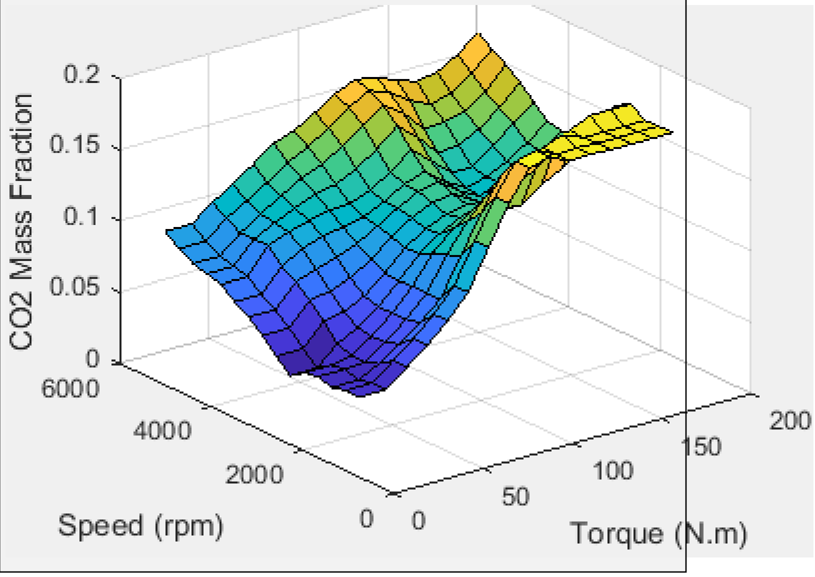
Map	Used For	In	Description
Indicated mean effective pressure post inject correction	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The indicated mean effective pressure post inject correction lookup table, $f_{IMEPpost}$, is a function of the engine speed and fuel rail pressure relative to optimal breakpoints, $\Delta IMEP_{post} = f_{IMEPpost}(\Delta SOI_{post},F_{post})$, where: - $\Delta IMEP_{post}$ is indicated mean effective pressure post inject correction, in Pa. - ΔSOI_{post} is indicated mean effective pressure post inject start of inject timing centroid, in degATDC. - F_{post} is indicated mean effective pressure post inject mass sum, in mg per injection. 
Fuel rail pressure efficiency multiplier	"CI Engine Torque Structure Model" on page 2-25	CI Core Engine CI Controller	The fuel rail pressure efficiency multiplier lookup table, $f_{FUELPeff}$, is a function of the engine speed and fuel rail pressure relative to optimal breakpoints, $FUELP_{eff} = f_{FUELPeff}(\Delta FUELP,N)$, where: - $FUELP_{eff}$ is fuel rail pressure efficiency multiplier, dimensionless. - $\Delta FUELP$ is fuel rail pressure relative to optimal, in MPa. - N is engine speed, in rpm. 

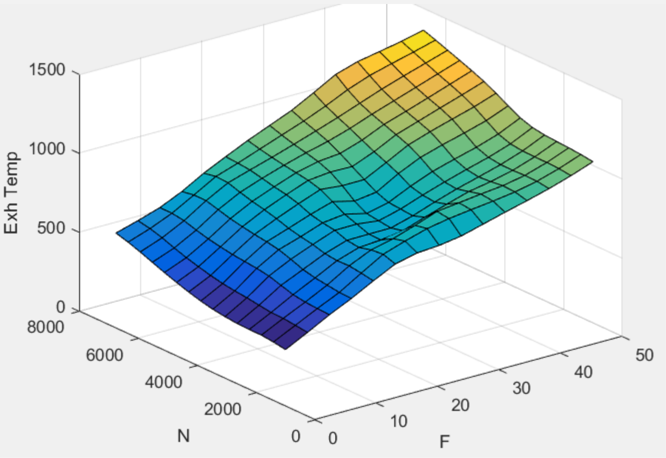
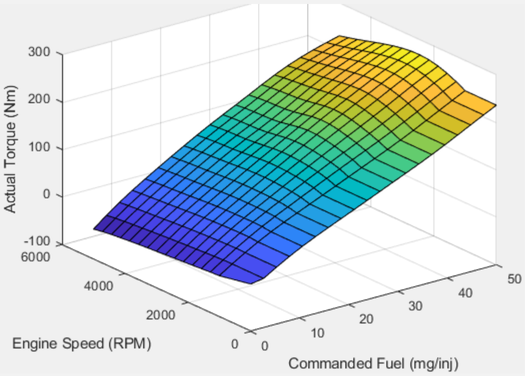
Map	Used For	In	Description
Engine brake torque	"CI Engine Simple Torque Model" on page 2-30	CI Core Engine CI Controller	For the simple torque lookup table model, the CI engine uses a lookup table is a function of engine speed and injected fuel mass, $T_{brake} = f_{Tnf}(F, N)$, where: • $Tq = T_{brake}$ is engine brake torque after accounting for engine mechanical and pumping friction effects, in N·m. • F is injected fuel mass, in mg per injection. • N is engine speed, in rpm. 

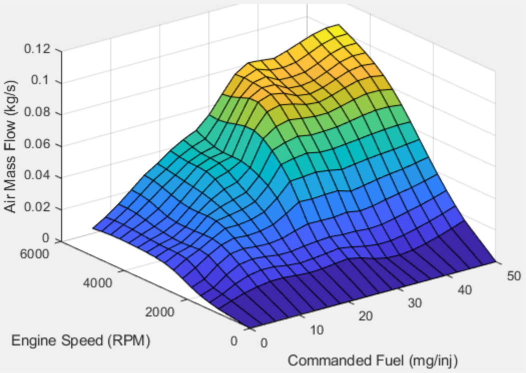
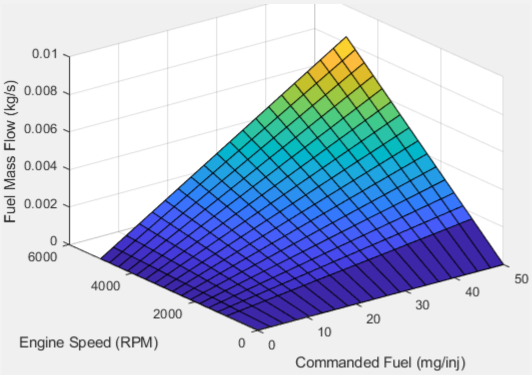
Map	Used For	In	Description
Hydrocarbon (HC) mass fraction	HC emissions	CI Core Engine	The CI Core Engine HC emission mass fraction lookup table is a function of engine torque and engine speed, *HC Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *HC Mass Fraction* is the HC emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

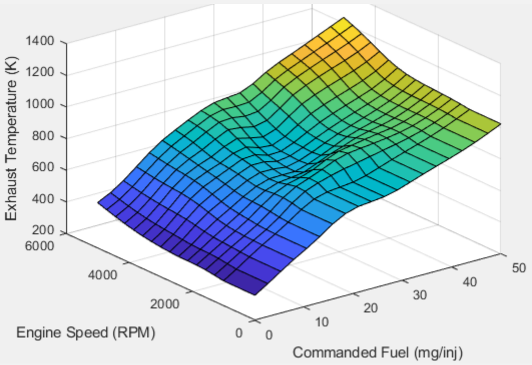
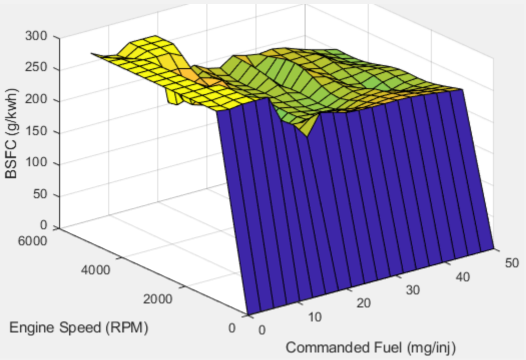
Map	Used For	In	Description
Carbon monoxide (CO) mass fraction	CO emissions	CI Core Engine	The CI Core Engine CO emission mass fraction lookup table is a function of engine torque and engine speed, *CO Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *CO Mass Fraction* is the CO emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

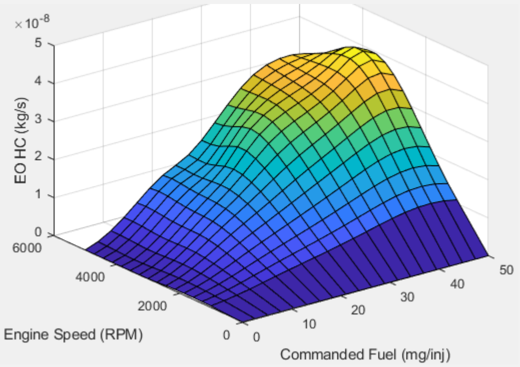
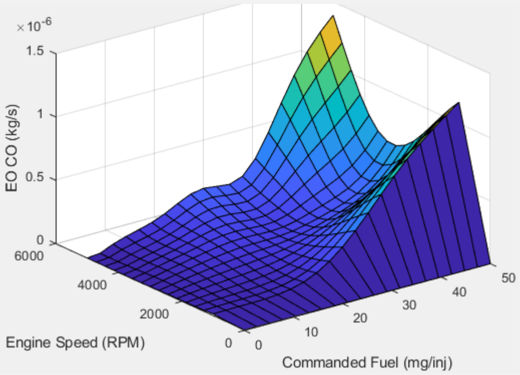
Map	Used For	In	Description
Nitric oxide and nitrogen dioxide (NOx) mass fraction	NOx emissions	CI Core Engine	The CI Core Engine NOx emission mass fraction lookup table is a function of engine torque and engine speed, *NOx Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *NOx Mass Fraction* is the NOx emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

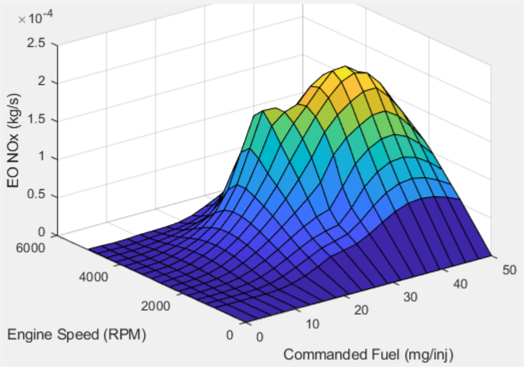
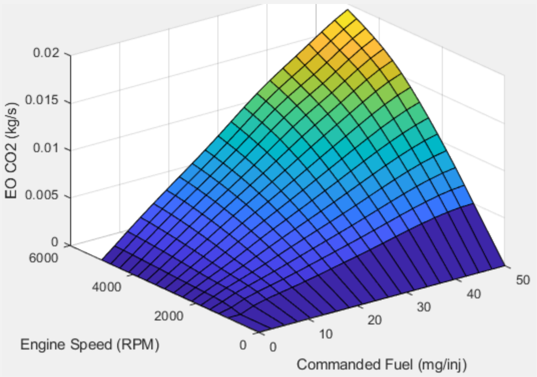
Map	Used For	In	Description
Carbon dioxide (CO_2) mass fraction	CO_2 emissions	CI Core Engine	The CI Core Engine CO_2 emission mass fraction lookup table is a function of engine torque and engine speed, *CO2 Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *CO2 Mass Fraction* is the CO_2 emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

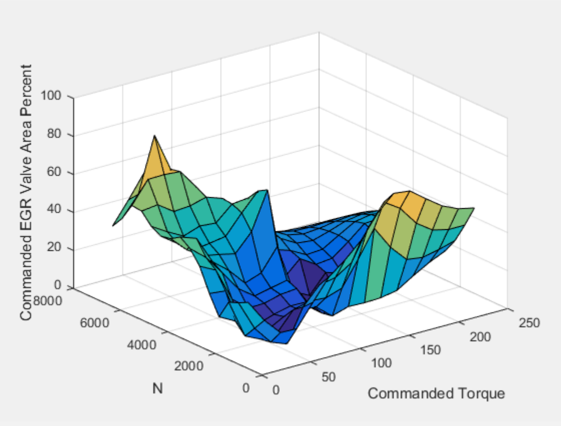
Map	Used For	In	Description
Exhaust temperature	Engine exhaust temperature as a function of injected fuel mass and engine speed	CI Core Engine CI Controller	The lookup table for the exhaust temperature is a function of injected fuel mass and engine speed $T_{exh} = f_{Texh}(F, N)$ where: - T_{exh} is exhaust temperature, in K. - F is injected fuel mass, in mg per injection. - N is engine speed, in rpm. 
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. 

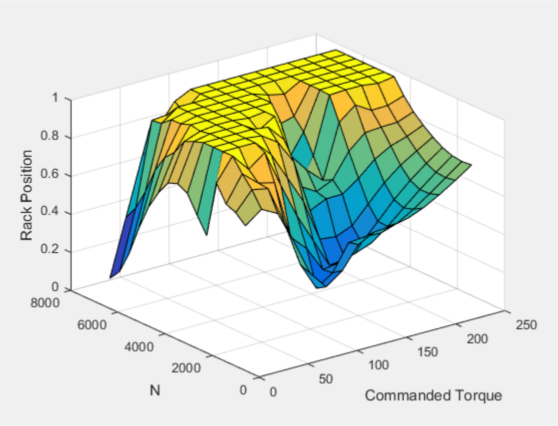
Map	Used For	In	Description
Engine air mass flow	Engine air mass flow as a function of commanded fuel mass and engine speed	Mapped CI Engine	The air mass flow lookup table is a function of commanded fuel mass and engine speed, $\dot{m}_{intk} = f(F_{max}, N)$, where: • $\dot{m}_{intk}$ is engine air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

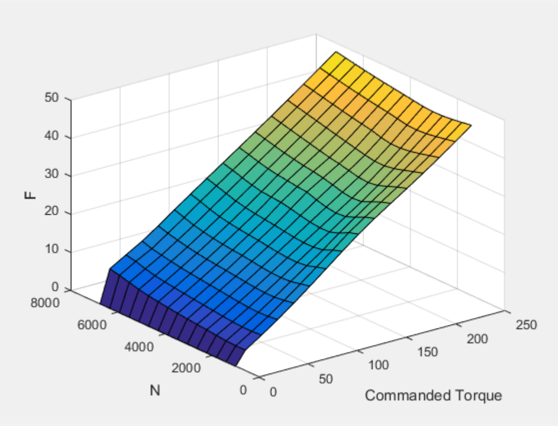
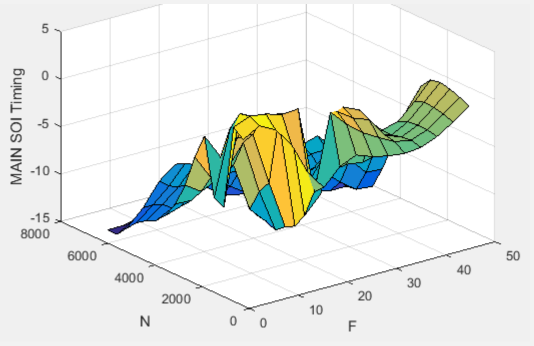
Map	Used For	In	Description
Engine exhaust temperature	Engine exhaust temperature as a function of commanded fuel mass and engine speed	Mapped CI Engine	The engine exhaust temperature table is a function of commanded fuel mass and engine speed, $T_{exh}= ƒ(F, N)$, where: • $T_{exh}$is exhaust temperature, in K. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm. 
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, $BSFC= ƒ(F, N)$, where: • $BSFC$ is BSFC, in g/kWh. • F is commanded fuel mass, in mg per injection. • N is engine speed, in rpm. 

Map	Used For	In	Description
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. 
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*= *f*(*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. 

Map	Used For	In	Description
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. 
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*= ƒ(*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. 

Map	Used For	In	Description
Commanded exhaust gas recirculation (EGR) valve area percent	Commanded exhaust gas recirculation (EGR) valve area percent as a function of commanded torque and engine speed	CI Controller	The commanded exhaust gas recirculation (EGR) valve area percent lookup table is a function of commanded torque and engine speed $EGR_{cmd} = f_{EGRcmd}(Trq_{cmd}, N)$ where: - EGR_{cmd} is commanded EGR valve area percent, in percent. - Trq_{cmd} is commanded engine torque, in N·m. - N is engine speed, in rpm. 

Map	Used For	In	Description
Variable geometry turbocharger (VGT) rack position	Variable geometry turbocharger (VGT) rack position as a function of commanded torque and engine speed	CI Controller	The variable geometry turbocharger (VGT) rack position lookup table is a function of commanded torque and engine speed $RP_{cmd} = f_{RPcmd}(Trq_{cmd}, N)$ where: • RP_{cmd} is VGT rack position command, in percent. • Trq_{cmd} is commanded engine torque, in N·m. • N is engine speed, in rpm. 

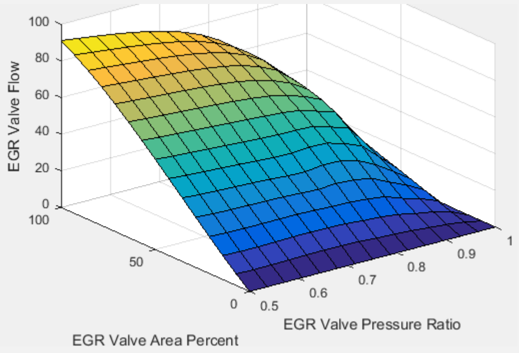
Map	Used For	In	Description
Commanded total fuel mass per injection	Commanded total fuel mass per injection as a function of torque command and engine speed	CI Controller	The commanded total fuel mass per injection table is a function of the torque command and engine speed $F_{cmd,\,tot} = f_{Fcmd,\,tot}(Trq_{cmd}, N)$ where: - $F_{cmd,tot} = F$ is commanded total fuel mass per injection, in mg per cylinder. - Trq_{cmd} is commanded engine torque, in N·m. - N is engine speed, in rpm. 
Main start-of-injection (SOI) timing	SOI timing as a function of commanded fuel mass and engine speed	CI Controller	The main start-of-injection (SOI) timing lookup table is a function of commanded fuel mass and engine speed $MAINSOI = f(F_{cmd,\,tot}, N)$ where: - $MAINSOI$ is the main start-of-injection timing, in degrees crank angle after top dead center (degATDC). - $F_{cmd,tot} = F$ is commanded fuel mass, in mg per injection. - N is engine speed, in rpm. 

<table>
<tr><th>Map</th><th>Used For</th><th>In</th><th>Description</th></tr>
<tr>
<td>Standard exhaust gas recirculation (EGR) mass flow</td>
<td>EGR mass flow as a function of the standard flow pressure ratio and EGR valve flow area</td>
<td>CI Controller</td>
<td>
The standard exhaust gas recirculation (EGR) mass flow is a lookup table that is a function of the standard flow pressure ratio and EGR valve flow area

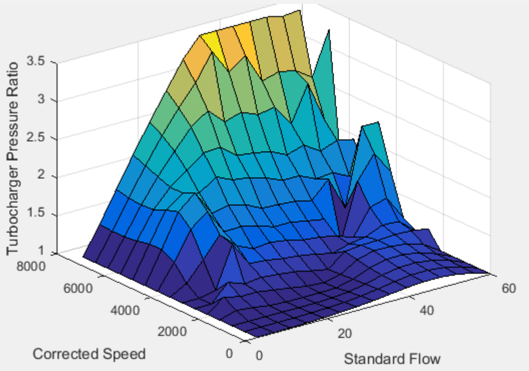
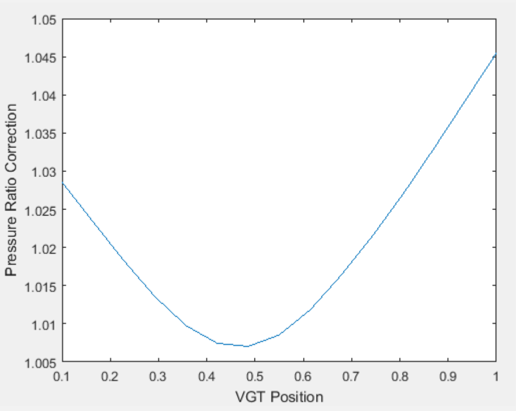
$$\dot{m}_{egr,\,std} = f(\frac{MAP}{P_{exh,\,est}}, EGRap)$$

where:

- $\dot{m}_{egr,\,std}$ is the standard EGR valve mass flow, in g/s.
- $P_{exh,est}$ is the estimated exhaust back-pressure, in Pa.
- MAP is the cycle average intake manifold absolute pressure, in Pa.
- $EGRap$ is the measured EGR valve area, in percent.

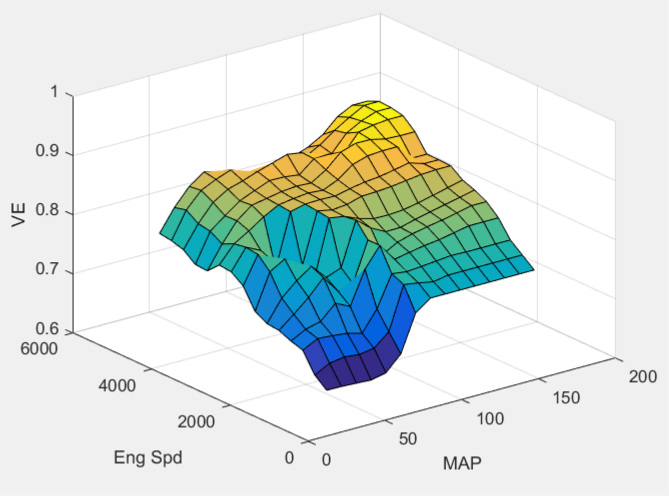


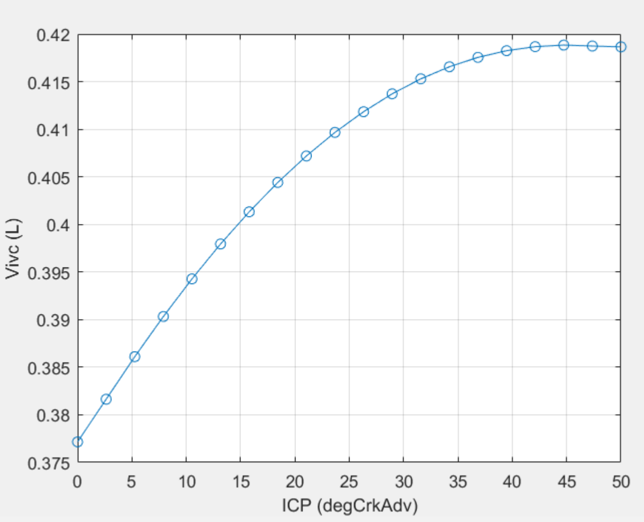
</td>
</tr>
</table>

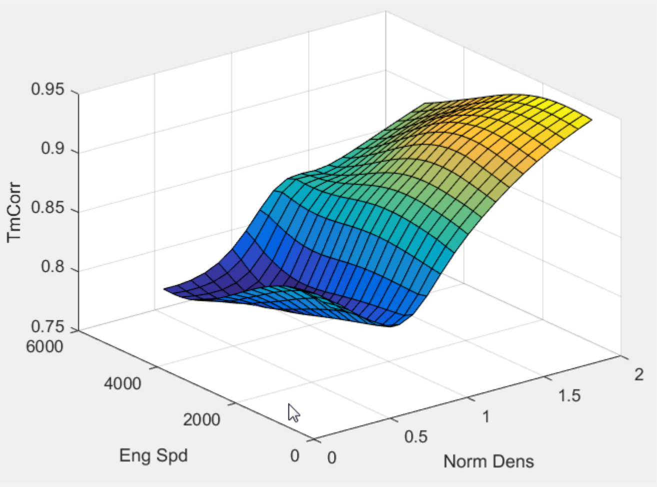
Map	Used For	In	Description
Turbocharger pressure ratio	Turbocharger pressure ratio as a function of the standard air mass flow and corrected turbocharger speed	CI Controller	The turbocharger pressure ratio, corrected for variable geometry turbocharger (VGT) speed, is a lookup table that is a function of the standard air mass flow and corrected turbocharger speed, $Pr_{turbo} = f(\dot{m}_{airstd}, N_{vgtcorr})$, where: - Pr_{turbo} is the turbocharger pressure ratio, corrected for VGT speed. - $\dot{m}_{airstd}$ is the standard air mass flow, in g/s. - $N_{vgtcorr}$ is the corrected turbocharger speed, in rpm/K^(1/2). 
Turbocharger pressure ratio correction	Turbocharger pressure ratio correction as a function of the rack position	CI Controller	The variable geometry turbocharger pressure ratio correction is a function of the rack position, $Pr_{vgtcorr}$= $f(VGT_{pos})$, where: - $Pr_{vgtcorr}$ is the turbocharger pressure ratio correction. - VGT_{pos} is the variable geometry turbocharger (VGT) rack position. 

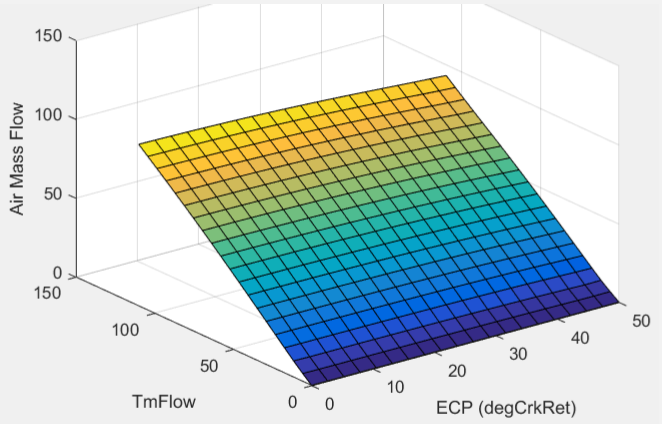
Calibration Maps in Spark-Ignition (SI) Blocks

In the engine models, the Powertrain Blockset blocks implement these calibration maps.

Map	Used for	In	Description
Engine volumetric efficiency	“SI Engine Speed-Density Air Mass Flow Model” on page 2-11	SI Core Engine SI Controller	The engine volumetric efficiency lookup table, f_{η_v}, is a function of intake manifold absolute pressure and engine speed $$\eta_v = f_{\eta_v}(MAP, N)$$ where: - η_v is engine volumetric efficiency, dimensionless. - MAP is intake manifold absolute pressure, in KPa. - N is engine speed, in rpm. 

Map	Used for	In	Description
Cylinder volume at intake valve close table (IVC)	"SI Engine Dual-Independent Cam Phaser Air Mass Flow Model" on page 2-5	SI Core Engine SI Controller	The cylinder volume at intake valve close table (IVC), f_{Vivc} is a function of the intake cam phaser angle $V_{IVC} = f_{Vivc}(\varphi_{ICP})$ where: • V_{IVC} is cylinder volume at IVC, in L. • φ_{ICP} is intake cam phaser angle, in crank advance degrees. 

Map	Used for	In	Description
Trapped mass correction	“SI Engine Dual-Independent Cam Phaser Air Mass Flow Model” on page 2-5	SI Core Engine SI Controller	The trapped mass correction factor table, f_{TMcorr}, is a function of the normalized density and engine speed $$TM_{corr} = f_{TMcorr}(\rho_{norm}, \quad N)$$ where: - TM_{corr}, is trapped mass correction multiplier, dimensionless. - ρ_{norm} is normalized density, dimensionless. - N is engine speed, in rpm. 

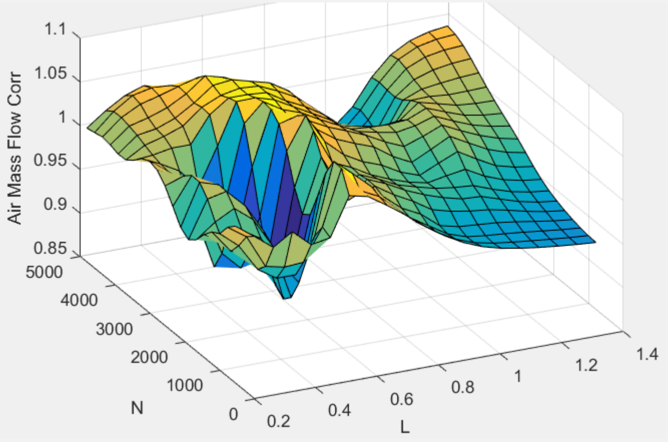
Map	Used for	In	Description
Air mass flow at cam phaser angles	"SI Engine Dual-Independent Cam Phaser Air Mass Flow Model" on page 2-5	SI Core Engine SI Controller	The phaser intake mass flow model lookup table is a function of exhaust cam phaser angles and trapped air mass flow $$\dot{m}_{intkideal} = f_{intkideal}(\varphi_{ECP}, TM_{flow})$$ where: - $\dot{m}_{intkideal}$ is engine intake port mass flow at arbitrary cam phaser angles, in g/s. - φ_{ECP} is exhaust cam phaser angle, in degrees crank retard. - TM_{flow} is flow rate equivalent to corrected trapped mass at the current engine speed, in g/s. 

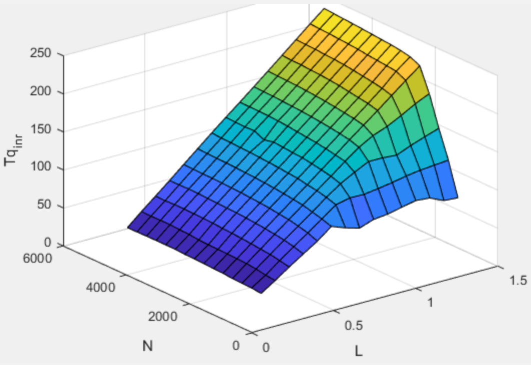
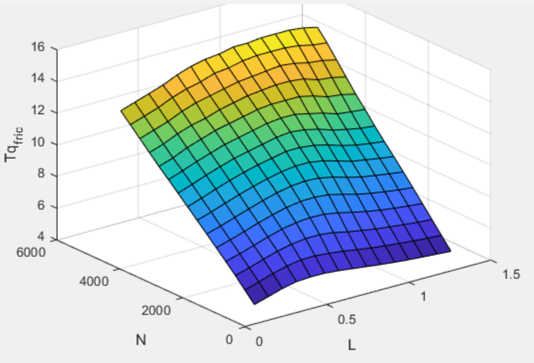
<table>
<tr><th>Map</th><th>Used for</th><th>In</th><th>Description</th></tr>
<tr><td>Air mass flow correction</td><td>“SI Engine Dual-Independent Cam Phaser Air Mass Flow Model” on page 2-5</td><td>SI Core Engine
SI Controller</td><td>The intake air mass flow correction lookup table, $f_{aircorr}$, is a function of ideal load and engine speed

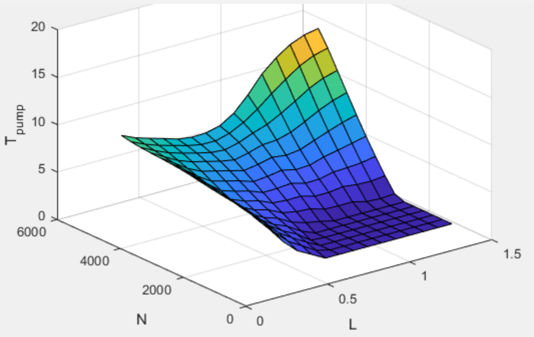
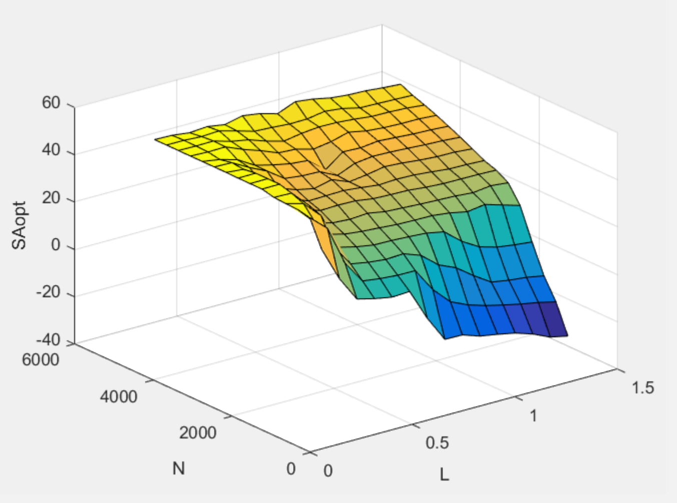
$$\dot{m}_{air} = \dot{m}_{intkideal} f_{aircorr}(L_{ideal}, N)$$

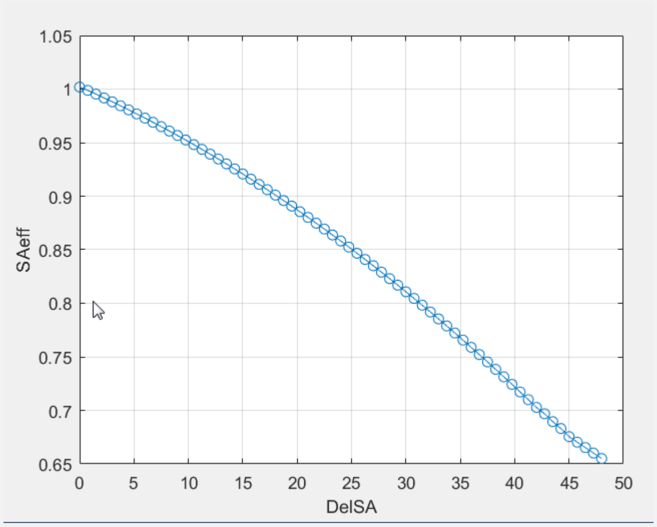
where:

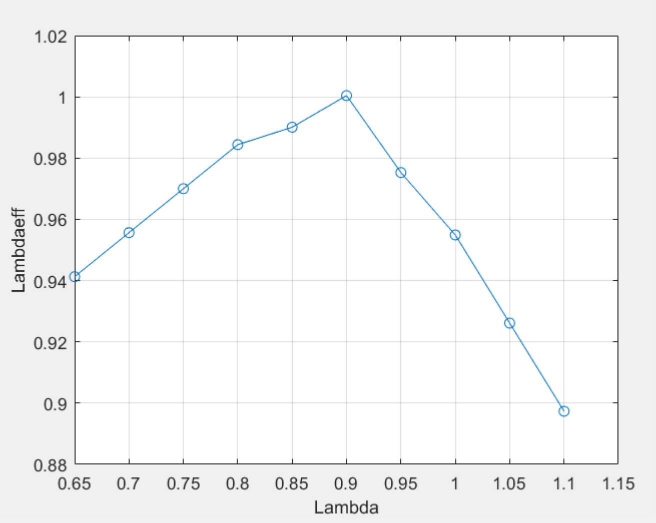
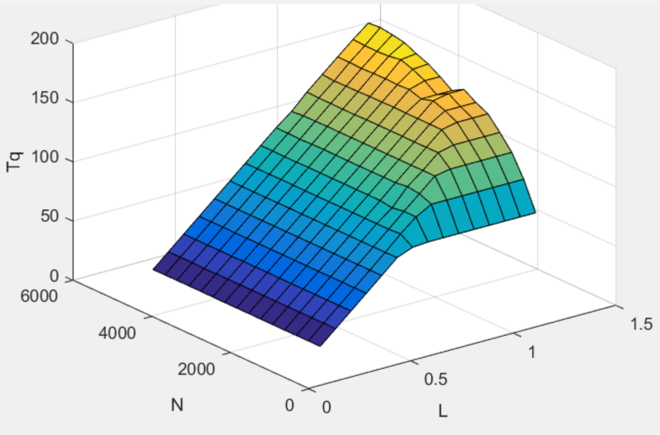
- L_{ideal} is engine load (normalized cylinder air mass) at arbitrary cam phaser angles, uncorrected for final steady-state cam phaser angles, dimensionless.
- N is engine speed, in rpm.
- $\dot{m}_{air}$ is engine intake air mass flow final correction at steady-state cam phaser angles, in g/s.
- $\dot{m}_{intkideal}$ is engine intake port mass flow at arbitrary cam phaser angles, in g/s.

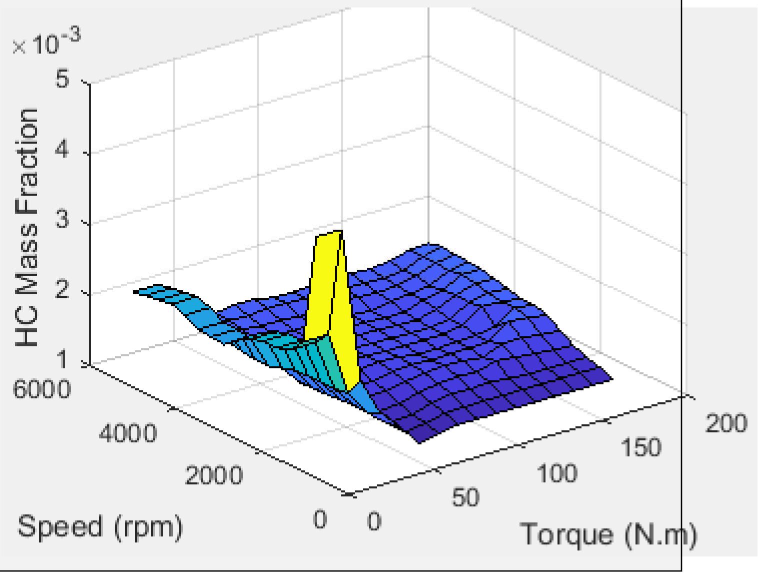

</td></tr>
</table>

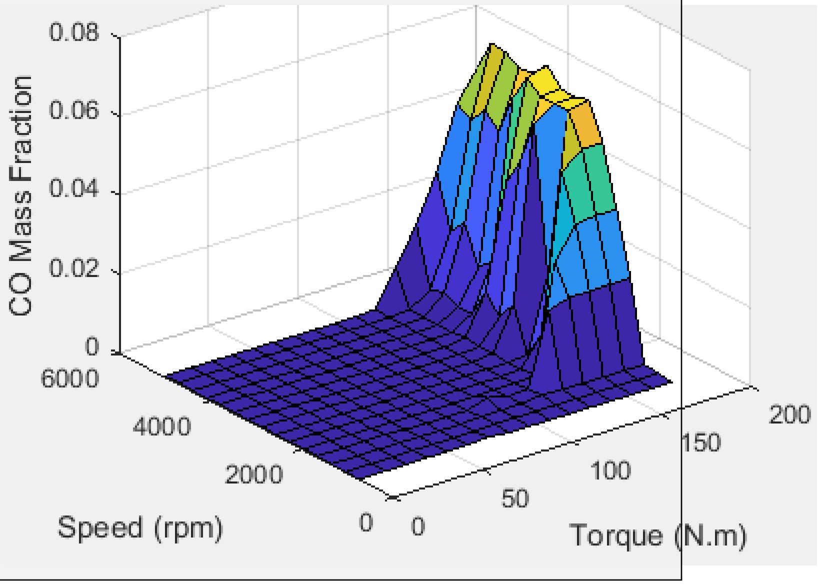
Map	Used for	In	Description
Inner torque	"SI Engine Torque Structure Model" on page 2-14	SI Core Engine SI Controller	The inner torque lookup table, f_{Tqinr}, is a function of engine speed and engine load, $Tq_{inr} = f_{Tqinr}(L, N)$, where: • Tq_{inr} is inner torque based on gross indicated mean effective pressure, in N·m. • L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. • N is engine speed, in rpm. 
Friction torque	"SI Engine Torque Structure Model" on page 2-14	SI Core Engine SI Controller	The friction torque lookup table, f_{Tfric}, is a function of engine speed and engine load, $T_{fric} = f_{Tfric}(L, N)$, where: • T_{fric} is friction torque offset to inner torque, in N·m. • L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. • N is engine speed, in rpm. 

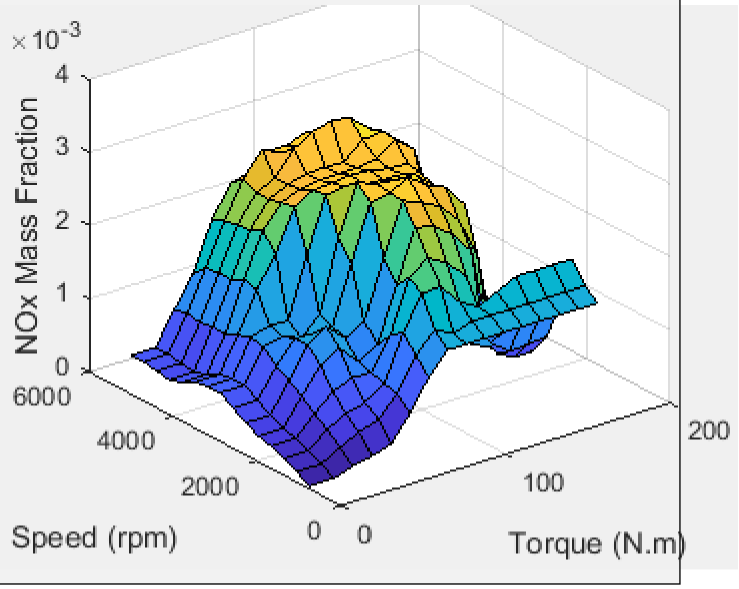
Map	Used for	In	Description
Pumping torque	"SI Engine Torque Structure Model" on page 2-14	SI Core Engine SI Controller	The pumping work lookup table, f_{Tpump}, is a function of engine load and engine speed, T_{pump}=`f`$_{\texttt{Tpump}}$`(L,N)`, where: - T_{pump} is pumping work, in N·m. - L is engine load, as a normalized cylinder air mass, dimensionless. - N is engine speed, in rpm. 
Optimal spark advance	"SI Engine Torque Structure Model" on page 2-14	SI Core Engine SI Controller	The optimal spark lookup table, f_{SAopt}, is a function of engine speed and engine load, $SA_{opt} = f_{SAopt}(L, N)$, where: - SA_{opt} is optimal spark advance timing for maximum inner torque at stoichiometric air-fuel ratio (AFR), in deg. - L is engine load at arbitrary cam phaser angles, corrected for final steady-state cam phaser angles, dimensionless. - N is engine speed, in rpm. 

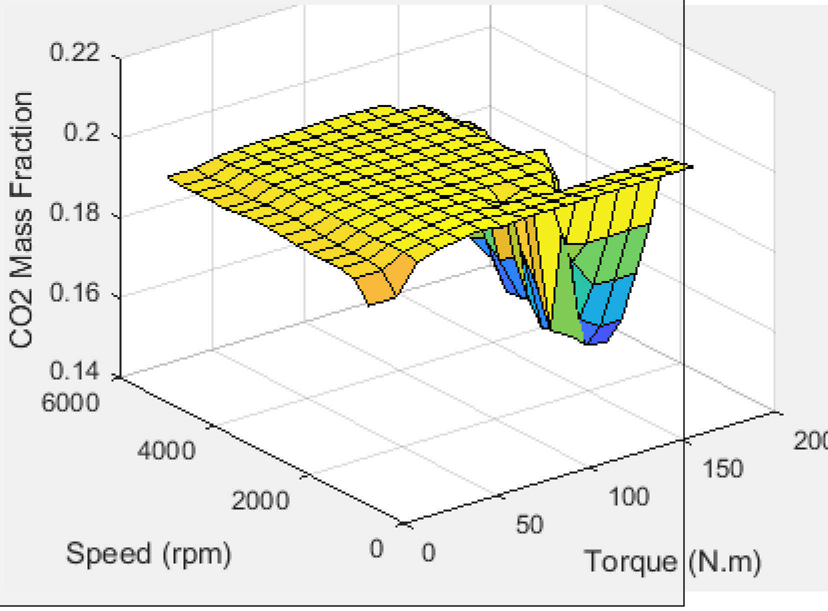
Map	Used for	In	Description
Spark efficiency	“SI Engine Torque Structure Model” on page 2-14	SI Core Engine SI Controller	The spark efficiency lookup table, f_{Msa}, is a function of the spark retard from optimal $M_{sa} = f_{Msa}(\Delta SA)$ $\Delta SA = SA_{opt} - SA$ where: • M_{sa} is the spark retard efficiency multiplier, dimensionless. • $\Delta SA$is the spark retard timing distance from optimal spark advance, in deg. 

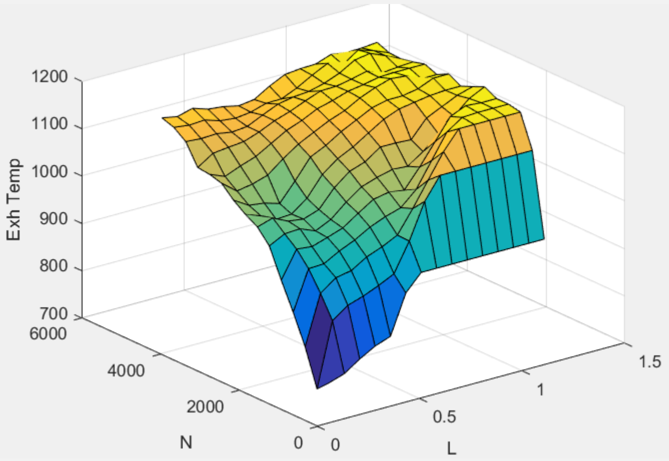
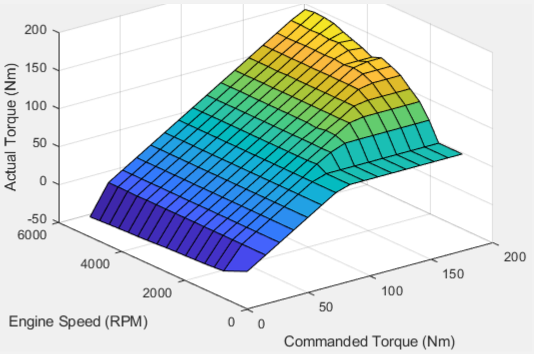
Map	Used for	In	Description
Lambda efficiency	"SI Engine Torque Structure Model" on page 2-14	SI Core Engine SI Controller	The lambda efficiency lookup table, $f_{M\lambda}$, is a function of lambda, $M_\lambda = f_{M\lambda}(\lambda)$, where: • M_λ is the lambda multiplier on inner torque to account for the air-fuel ratio (AFR) effect, dimensionless. • λ is lambda, AFR normalized to stoichiometric fuel AFR, dimensionless. 
Simple torque	"SI Engine Simple Torque Model" on page 2-20	SI Core Engine SI Controller	For the simple torque lookup table model, the SI engine uses a lookup table map that is a function of engine speed and load, $T_{brake} = f_{TnL}(L, N)$, where: • T_{brake} is engine brake torque after accounting for spark advance, AFR, and friction effects, in N·m. • L is engine load, as a normalized cylinder air mass, dimensionless. • N is engine speed, in rpm. 

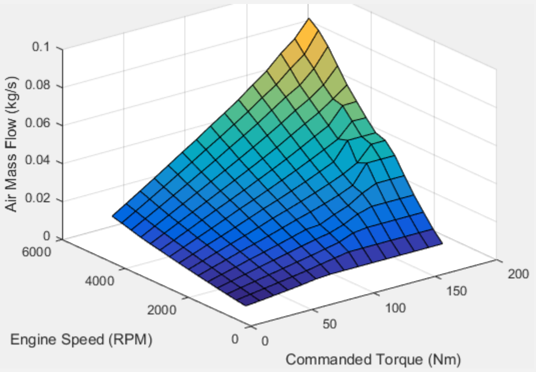
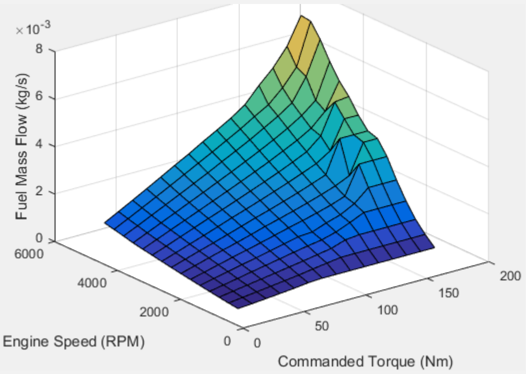
Map	Used for	In	Description
Hydrocarbon (HC) mass fraction	HC emissions	SI Core Engine	The SI Core Engine HC emission mass fraction lookup table is a function of engine torque and engine speed, *HC Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *HC Mass Fraction* is the HC emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

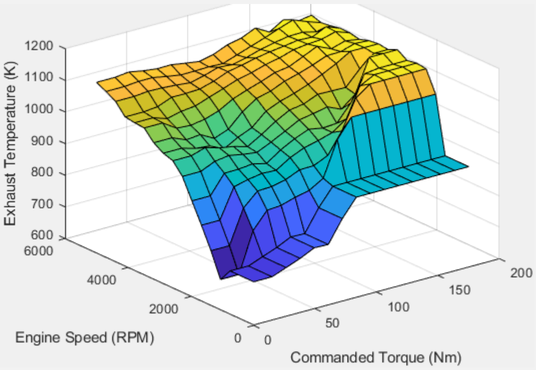
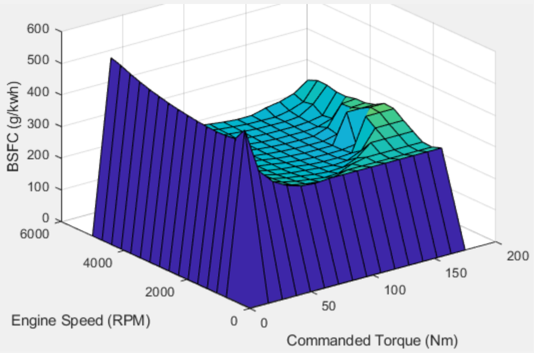
Map	Used for	In	Description
Carbon monoxide (CO) mass fraction	CO emissions	SI Core Engine	The SI Core Engine CO emission mass fraction lookup table is a function of engine torque and engine speed, *CO Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *CO Mass Fraction* is the CO emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

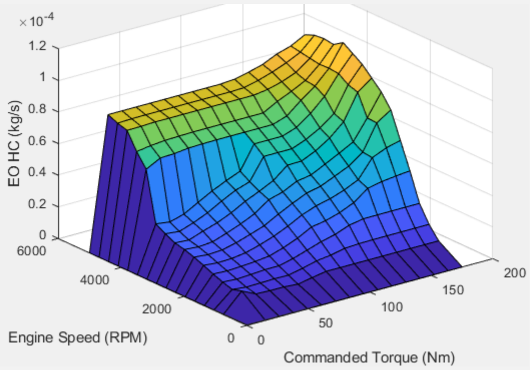
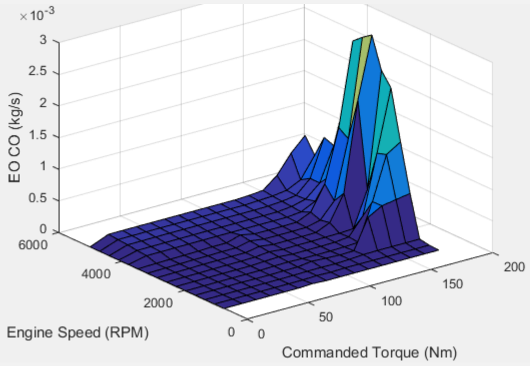
Map	Used for	In	Description
Nitric oxide and nitrogen dioxide (NOx) mass fraction	NOx emissions	SI Core Engine	The SI Core Engine NOx emission mass fraction lookup table is a function of engine torque and engine speed, *NOx Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *NOx Mass Fraction* is the NOx emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

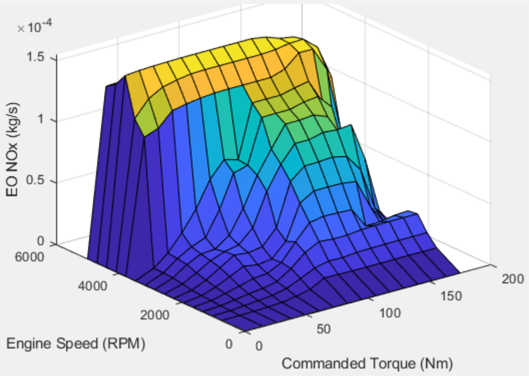
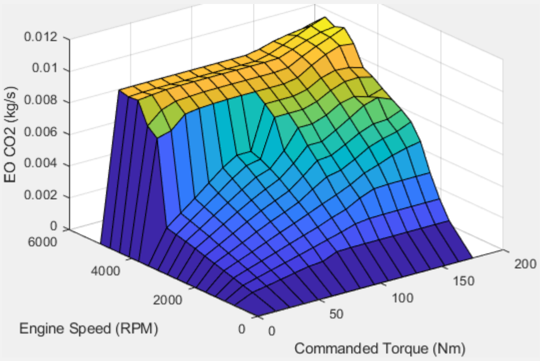
Map	Used for	In	Description
Carbon dioxide (CO_2) mass fraction	CO_2 emissions	SI Core Engine	The SI Core Engine CO_2 emission mass fraction lookup table is a function of engine torque and engine speed, *CO2 Mass Fraction* = ƒ(*Speed*, *Torque*), where: • *CO2 Mass Fraction* is the CO_2 emission mass fraction, dimensionless. • *Speed* is engine speed, in rpm. • *Torque* is engine torque, in N·m. 

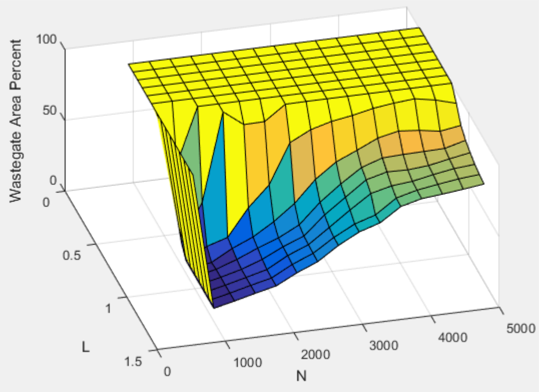
Map	Used for	In	Description
Exhaust temperature	Engine exhaust calculation as a function of engine speed and load	SI Core Engine SI Controller	The exhaust temperature lookup table, f_{Texh}, is a function of engine load and engine speed $$T_{exh} = f_{Texh}(L, N)$$ where: - T_{exh} is engine exhaust temperature, in K. - L is normalized cylinder air mass or engine load, dimensionless. - N is engine speed, in rpm. 
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. 

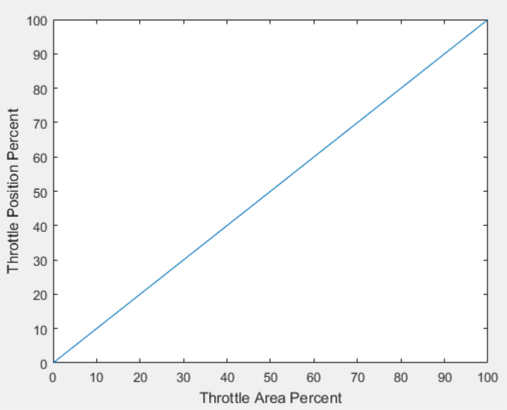
Map	Used for	In	Description
Engine air mass flow	Engine air mass flow as a function of commanded torque and engine speed	Mapped SI Engine	The engine air mass flow lookup table is a function of commanded engine torque and engine speed, $\dot{m}_{intk} = f(T_{cmd}, N)$, where: • $\dot{m}_{intk}$ is engine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

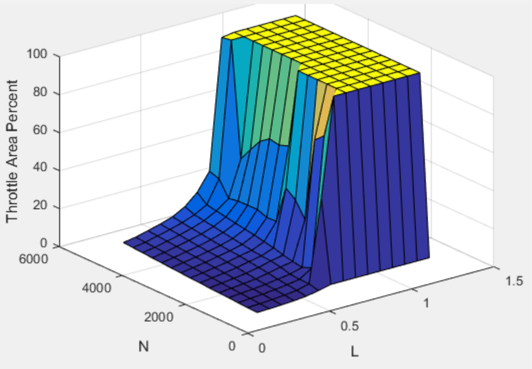
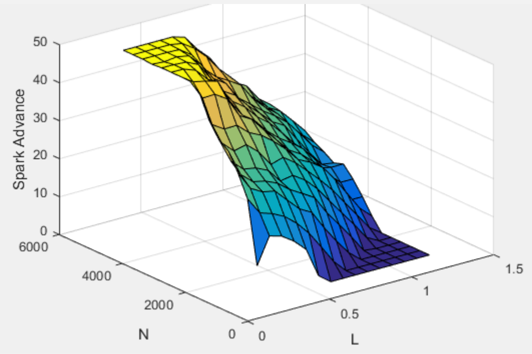
Map	Used for	In	Description
Engine exhaust temperature	Engine exhaust temperature as a function of commanded torque and engine speed	Mapped SI Engine	The engine exhaust temperature lookup table is a function of commanded engine torque and engine speed, T_{exh} = ƒ(T_{cmd}, N), where: • T_{exh} is exhaust 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, $BSFC$ = ƒ(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. 

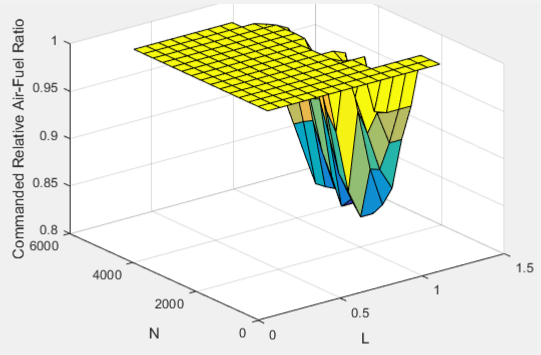
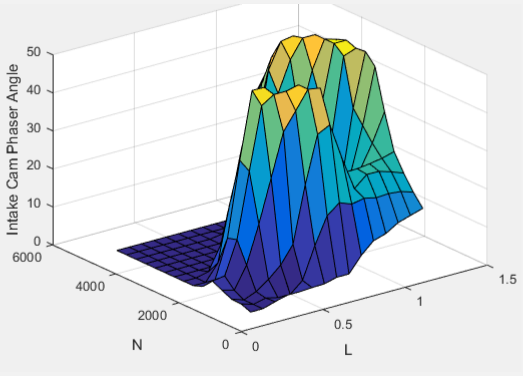
Map	Used for	In	Description
Engine-out (EO) hydrocarbon emissions	EO hydrocarbon emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out hydrocarbon emissions are a function of commanded engine torque and engine speed, $EO\ HC = f(T_{cmd}, N)$, where: • $EO\ HC$ is 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, $EO\ CO = f(T_{cmd}, N)$, 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. 

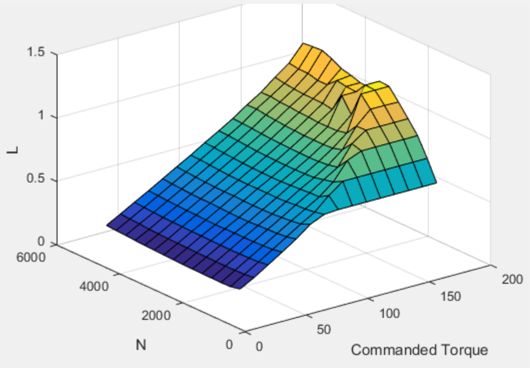
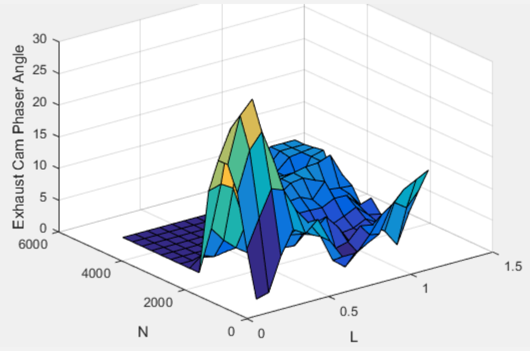
Map	Used for	In	Description
Engine-out (EO) nitric oxide and nitrogen dioxide emissions	EO nitric oxide and nitrogen dioxide emissions as a function of commanded torque and engine speed	Mapped SI Engine	The engine-out nitric oxide and nitrogen dioxide emissions are a function of commanded engine torque and engine speed, *EO NOx* = ƒ(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* = ƒ(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. 

Map	Used for	In	Description
Wastegate area percent command	Wastegate area percent command as a function of the commanded engine load and engine speed	SI Controller	The wastegate area percent command lookup table, f_{WAPcmd}, is a function of the commanded engine load and engine speed $$WAP_{cmd} = f_{WAPcmd}(L_{cmd}, N)$$ where: - WAP_{cmd} is wastegate area percentage command, in percent. - L_{cmd}=L is commanded engine load, dimensionless. - N is engine speed, in rpm. 

Map	Used for	In	Description
Throttle position percent command	Throttle position percent command as a function of the throttle area percentage command	SI Controller	The throttle position percent command lookup table, f_{TPPcmd}, is a function of the throttle area percentage command $$TPP_{cmd} = f_{TPPcmd}(TAP_{cmd})$$ where: • TPP_{cmd} is throttle position percentage command, in percent. • TAP_{cmd} is throttle area percentage command, in percent. 

Map	Used for	In	Description
Throttle area percent command	Throttle area percent command as a function of commanded load and engine speed	SI Controller	The throttle area percent command lookup table, f_{TAPcmd}, is a function of commanded load and engine speed $TAP_{cmd} = f_{TAPcmd}(L_{cmd}, N)$ where: - TAP_{cmd} is throttle area percentage command, in percent. - L_{cmd}=L is commanded engine load, dimensionless. - N is engine speed, in rpm. 
Spark advance	Spark advance as a function of estimated load and engine speed	SI Controller	The spark advance lookup table is a function of estimated load and engine speed. $SA = f_{SA}(L_{est}, N)$ where: - SA is spark advance, in crank advance degrees. - L_{est}=L is estimated engine load, dimensionless. - N is engine speed, in rpm. 

Map	Used for	In	Description
Commanded lambda	Commanded lambda as a function of estimated engine load and measured engine speed	SI Controller	The commanded lambda, λ_{cmd}, lookup table is a function of estimated engine load and measured engine speed $\lambda_{cmd} = f_{\lambda cmd}(L_{est}, N)$ where: • λ_{cmd} is commanded relative AFR, dimensionless. • L_{est}=L is estimated engine load, dimensionless. • N is engine speed, in rpm. 
Intake cam phaser angle command	Intake cam phaser angle command as a function of the engine load and engine speed	SI Controller	The intake cam phaser angle command lookup table, f_{ICPCMD}, is a function of the engine load and engine speed $\varphi_{ICPCMD} = f_{ICPCMD}(L_{est}, N)$ where: • φ_{ICPCMD} is commanded intake cam phaser angle, in degrees crank advance. • L_{est}=L is estimated engine load, dimensionless. • N is engine speed, in rpm. 

Map	Used for	In	Description
Commanded engine load	Commanded engine load as a function of the commanded torque and engine speed	SI Controller	The commanded engine load lookup table, f_{Lcmd}, is a function of the commanded torque and engine speed $L_{cmd} = f_{Lcmd}(T_{cmd}, N)$ where: • L_{cmd}=L is commanded engine load, dimensionless. • T_{cmd} is commanded torque, in N·m. • N is engine speed, in rpm. 
Exhaust cam phaser angle	Exhaust cam phaser angle as a function of the engine load and engine speed	SI Controller	The exhaust cam phaser angle command lookup table, f_{ECPCMD}, is a function of the engine load and engine speed $\varphi_{ECPCMD} = f_{ECPCMD}(L_{est}, N)$ where: • φ_{ECPCMD} is commanded exhaust cam phaser angle, in degrees crank retard. • L_{est}=L is estimated engine load, dimensionless. • N is engine speed, in rpm. 

See Also

SI Core Engine | CI Core Engine | Mapped SI Engine | Mapped CI Engine | SI Controller | CI Controller

External Websites

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

3

Reference Applications

Internal Combustion Engine Reference Application Projects

Use these reference applications as a starting point for your own internal combustion engine vehicle models.

Conventional Vehicle

Start building your own conventional vehicle with the “Conventional Vehicle Reference Application” on page 7-2. You can use it for vehicle analysis, including:

- Design tradeoff analysis and component sizing
- Control parameter optimization
- Hardware-in-the-loop (HIL) testing

The reference application includes a full conventional vehicle with spark-ignition (SI) or combustion-ignition (CI) engine. For more information, see “Build a Conventional Vehicle Model” on page 3-5.

Engine Dynamometers

Start building your own engine dynamometer test harnesses with these reference applications. You can use them for engine and controller calibration, validation, and optimization before integration with the vehicle model.

Reference Application	More Information
“CI Engine Dynamometer Reference Application” on page 7-13	“Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness” on page 3-11
“SI Engine Dynamometer Reference Application” on page 7-15	“Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15

See Also

Related Examples

- “Resize the CI Engine” on page 3-94
- “Resize the SI Engine” on page 3-101

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128

Hybrid and Electric Vehicle Reference Application Projects

Start building your own hybrid electric vehicle (HEV) and electric vehicle (EV) models with these reference applications. You can use them for vehicle analysis, including:

- Design tradeoff analysis and component sizing
- Control parameter optimization
- Hardware-in-the-loop (HIL) testing

Hybrid Electric Vehicle Reference Applications

Use these reference applications to simulate full HEV models with internal combustion engines, transmissions, batteries, motors, generators, and associated powertrain control algorithms.

Reference Application	More Information
“HEV Multimode Reference Application” on page 7-3	“Build Hybrid Electric Vehicle Multimode Model” on page 3-19
“HEV Input Power-Split Reference Application” on page 7-4	“Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
“HEV P0 Reference Application” on page 7-5	“Build Hybrid Electric Vehicle P0 Model” on page 3-51
“HEV P1 Reference Application” on page 7-6	“Build Hybrid Electric Vehicle P1 Model” on page 3-58
“HEV P2 Reference Application” on page 7-7	“Build Hybrid Electric Vehicle P2 Model” on page 3-65
“HEV P3 Reference Application” on page 7-8	“Build Hybrid Electric Vehicle P3 Model” on page 3-75
“HEV P4 Reference Application” on page 7-9	“Build Hybrid Electric Vehicle P4 Model” on page 3-82

Motors and Electric Vehicles

Use these reference applications to simulate full EV models with motor-generators, batteries, direct-drive transmissions, and associated powertrain control algorithms.

Reference Application	More Information
“EV Reference Application” on page 7-10	“Build Full Electric Vehicle Model” on page 3-26
“FCEV Reference Application” on page 7-12	“Build Fuel Cell Electric Vehicle” on page 3-37
“Motor Dynamometer Reference Application” on page 7-17	“Develop, Resize, and Calibrate Motors with Dynamometer Test Harness” on page 3-89

See Also

More About

- “Analyze Power and Energy” on page 3-129
- “Internal Combustion Engine Reference Application Projects” on page 3-2

Build a Conventional Vehicle Model

The conventional vehicle reference application represents a full vehicle model with an internal combustion engine, transmission, and associated powertrain control algorithms. Use the reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing. To create and open a working copy of the conventional vehicle reference application project, enter

```
autoblkConVehStart
```

By default, the conventional vehicle reference application is configured with these powertrain subsystem variants:

- 1.5–L spark-ignition (SI) dynamic engine
- Performance mode transmission controller

This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and ambient temperature and pressure.	

Reference Application Element	Description	Variants
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a transmission control module (TCM) and engine control module (ECM).	✓
`Passenger Car` subsystem	Implements a passenger car that contains transmission drivetrain and engine plant model subsystems.	✓
`Visualization` subsystem	Displays vehicle-level performance, fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Optimize Transmission Shift Maps

You can use the conventional vehicle reference application to optimize the transmission control module (TCM) shift schedules. Use the optimized shift schedules to:

- Design control algorithms.
- Assess the impact of powertrain changes, such as an engine or gear ratio, on performance, fuel economy, and emissions.

TCM shift schedule optimization requires Simulink Design Optimization, the Global Optimization Toolbox, and Stateflow. To increase the performance of the optimization, consider also using the Parallel Computing Toolbox.

To run the TCM shift schedule optimization, open a version of the conventional vehicle reference application that includes the option to optimize transmission shift maps by using this command:

```
autoblkConVehShftOptStart
```

Click **Optimize Transmission Shift Maps**. Optimizing the shift schedules can take time to run.

For more information, see “Optimize Transmission Control Module Shift Schedules” on page 7-18.

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant and drivetrain efficiencies, including an engine plant histogram of time spent at the different engine efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

For more information about the live script, see “Analyze Power and Energy” on page 3-129.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.

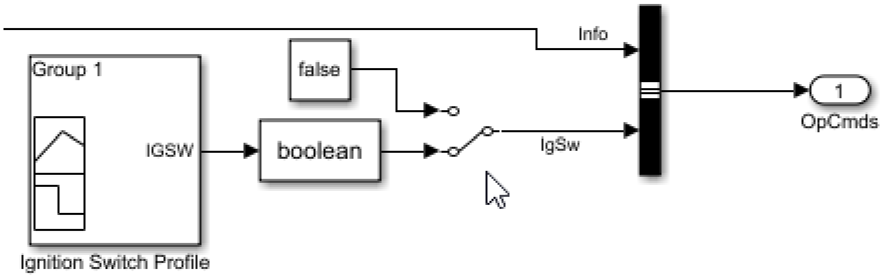
Block Variants			Description
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

To implement a powertrain control module (PCM), the `Controller` subsystem has a transmission control module (TCM) and an engine control module (ECM). The reference application has these variants.

Controller	Variant	Description
Engine controller — ECM	`SiEngineController` (default)	SI engine controller
	`CiEngineController`	CI engine controller
Transmission controller — TCM	`PowertrainMaxPowerController` (default)	Performance mode transmission controller

Controller	Variant	Description
	`PowertrainBestFuelController`	Fuel economy mode transmission controller

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain and engine plant model subsystems. To create your own internal combustion engine variants for the reference application, use the CI and SI engine project templates. The reference application has these variants.

Drivetrain Subsystem	Variant	Description
Dual clutch transmission (DCT)	`DCT Block` (default)	Configure drivetrain with DCT block or DCT system. For the DCT system, you can configure the type of filter.
	`DCT System`	
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Vehicle	`Vehicle Body 3 DOF Longitudinal`	Vehicle configured for 3 degrees of freedom.
Wheels and Brakes	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine
	`SiEngineCoreV`	Dynamic SI V Twin-Turbo Single-Intake Engine
	`SiEngineCoreVNA`	Dynamic SI V Engine
	`SiEngineCoreVThr2`	Dynamic SI V Twin-Turbo Twin-Intake Engine
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiDLEngine`	Deep learning SI engine
	`CiEngine`	Dynamic CI Core Engine with turbocharger
	`CiMappedEngine`	Mapped CI Engine with implicit turbocharger

See Also

Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Conventional Vehicle Reference Application” on page 7-2
- “Conventional Vehicle Spark-Ignition Engine Fuel Economy and Emissions” on page 1-10
- “Conventional Vehicle Powertrain Efficiency” on page 1-15
- “Optimize Transmission Control Module Shift Schedules” on page 7-18
- “Track Drive Cycle Errors” on page 5-3

More About

- “Analyze Power and Energy” on page 3-129
- “Internal Combustion Engine Reference Application Projects” on page 3-2
- Simulation Data Inspector
- “Variant Systems”

Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness

The compression-ignition (CI) engine dynamometer reference application represents a CI engine plant and controller connected to an AC dynamometer with a tailpipe emission analyzer. Using the reference application, you can calibrate, validate, and optimize the engine controller and plant model parameters before integrating the engine with the vehicle model. To create and open a working copy of the CI engine dynamometer reference application project, enter

```
autoblkCIDynamometerStart
```

By default, the reference application is configured with a 1.5–L CI dynamic engine.

You can configure the reference application project for different dynamometer control modes. To implement the operating modes, the reference application uses variant subsystems.

This table summarizes the dynamometer tests.

Test	Objective	Method	CI Engine Variant	
			Mapped	Dynamic
Execute Engine Mapping Experiment	Assess engine torque, fuel flow, and emission performance results using an existing engine controller calibration.	Dynamometer controller commands a series of engine speeds and torques to the engine controller. At each quasi-steady-state operating point, the experiment records the engine plant model output and the controller commands for the current calibration parameters.	✓	✓
Execute Model Predictive Control Plant Model Experiment	Generate transient engine datasets for linear plant models useful for model predictive controllers.	Dynamometer controller commands engine speed and torque dynamically as a function of time using a pseudo random binary sequence. Experiment records the transient engine torque, temperature, airflow, and emission responses determined from linear dynamic plant model fitting via system identification.	✓	✓
Recalibrate Controller	Match measured engine torque to commanded engine torque across engine operating range.	Dynamometer controller generates a feedforward fuel command table by matching the measured engine torque to the commanded engine torque across the engine operating range.		✓

Test	Objective	Method	CI Engine Variant	
			Mapped	Dynamic
Resize Engine and Recalibrate Controller	Match engine torque to desired engine power and number of cylinders.	Dynamometer resizes the dynamic engine and engine calibration parameters. Also, the dynamometer recalibrates the controller and mapped engine model to match the resized dynamic engine.	✓	✓
Generate Mapped Engine from Spreadsheet	Generate a mapped engine calibration from a data spreadsheet. Update the mapped engine with the calibrated data.	Dynamometer uses the Model-Based Calibration Toolbox to fit data from a spreadsheet, generate calibrated tables, and update the mapped engine parameters.	✓	

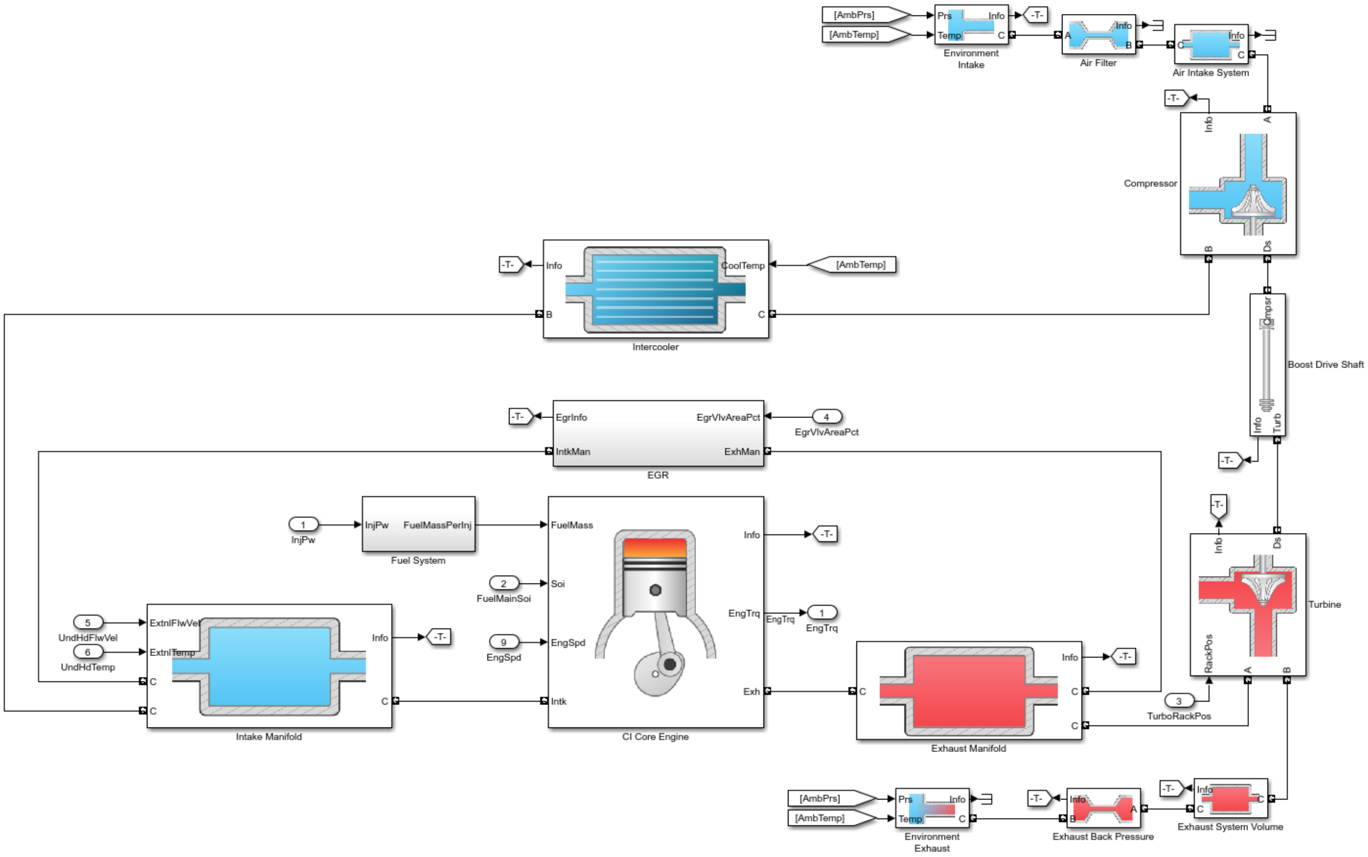
Engine System

The reference application includes variant subsystems for mapped (steady-state) and dynamic 1.5–L CI engine systems with a variable geometry turbocharger (VGT). Using the CI engine project template, you can create your own CI engine variants.

Objective	Engine Variant
Dynamic analysis, including manifold and turbocharger dynamics	Dynamic
Faster execution	Mapped

Dynamic

`CiEngineCore.slx` contains the engine intake system, exhaust system, exhaust gas recirculation (EGR), fuel system, core engine, and turbocharger subsystems.



Mapped

`CiMappedEngine.slx` uses the Mapped CI Engine block to look up power, air mass flow, fuel flow, exhaust temperature, efficiency, and emission performance as functions of engine speed and injected fuel mass.

Performance Monitor

The reference application contains a Performance Monitor block that you can use to plot steady-state and dynamic results. You can plot:

- Steady-state results as a function of one or two variables.
- Dynamic results using the Simulation Data Inspector.

See Also

Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “CI Engine Dynamometer Reference Application” on page 7-13
- “Generate Mapped CI Engine from a Spreadsheet” on page 3-108
- “Resize the CI Engine” on page 3-94

More About

- “CI Engine Project Template” on page 4-2
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128
- “Variant Systems”

Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness

The spark-ignition (SI) engine dynamometer reference application represents a SI engine plant and controller connected to an AC dynamometer with a tailpipe emission analyzer. Using the reference application, you can calibrate, validate, and optimize the engine controller and plant model parameters before integrating the engine with the vehicle model. To create and open a working copy of the SI engine dynamometer reference application project, enter

```
autoblkSIDynamometerStart
```

By default, the reference application is configured with a 1.5–L SI dynamic engine.

You can configure the reference application project for different dynamometer control modes. To implement the operating modes, the reference application uses variant subsystems.

This table summarizes the dynamometer tests.

Test	Objective	Method	SI Engine Variant	
			Mapped	Dynamic
Execute Engine Mapping Experiment	Assess engine torque, fuel flow, and emission performance results using an existing engine controller calibration.	Dynamometer controller commands a series of engine speeds and torques to the engine controller. At each quasi-steady-state operating point, the experiment records the engine plant model output and the controller commands for the current calibration parameters.	✓	✓
Execute Model Predictive Control Plant Model Experiment	Generate transient engine datasets for linear plant models useful for model predictive controllers.	Dynamometer controller commands engine speed and torque dynamically as a function of time using a pseudo random binary sequence. Experiment records the transient engine torque, temperature, airflow, and emission responses determined from linear dynamic plant model fitting via system identification.	✓	✓
Recalibrate Controller	Match measured engine torque to commanded engine torque across engine operating range.	Dynamometer controller generates a feedforward throttle table by matching the measured engine torque to the commanded engine torque across the engine operating range.		✓

Test	Objective	Method	SI Engine Variant	
			Mapped	Dynamic
Resize Engine and Recalibrate Controller	Match engine torque to desired engine power and number of cylinders.	Dynamometer resizes the dynamic engine and engine calibration parameters. Also, the dynamometer recalibrates the controller and mapped engine model to match the resized dynamic engine. For an example, see “Resize the SI Engine” on page 3-101.	✓	✓
Generate Mapped Engine from Spreadsheet	Generate a mapped engine calibration from a data spreadsheet. Update the mapped engine with the calibrated data.	Dynamometer uses the Model-Based Calibration Toolbox to fit data from a spreadsheet, generate calibrated tables, and update the mapped engine parameters. For an example, see “Generate Mapped SI Engine from a Spreadsheet” on page 3-113.	✓	
Generate Deep Learning Engine Model	Train a deep learning model of dynamic engine behavior from measured laboratory data or a high-fidelity engine model.	Dynamometer uses the Deep Learning Toolbox™ and Statistics and Machine Learning Toolbox™ to generate a dynamic deep learning engine model and update the mapped engine parameters. For an example, see “Generate Deep Learning SI Engine Model” on page 3-117.	✓	

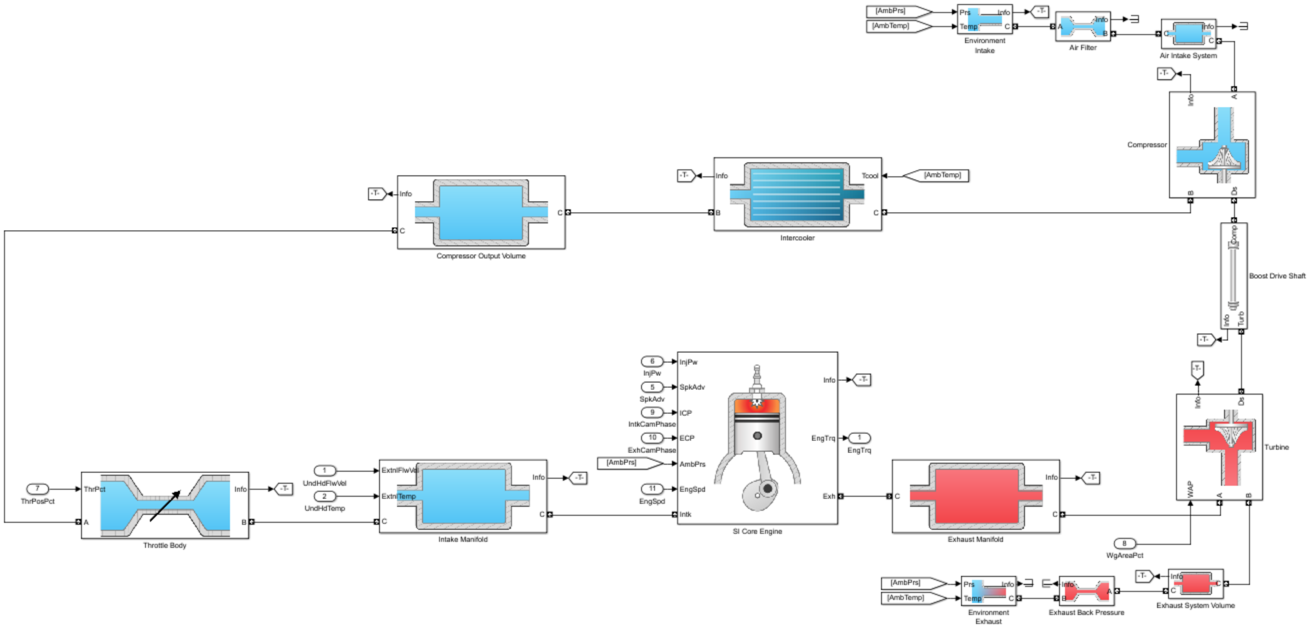
Engine System

The reference application includes variant subsystems for mapped (steady-state) and dynamic turbocharged 1.5–L SI engine. Using the SI engine project template, you can create your own SI engine variants.

Objective	Engine Variant
Dynamic analysis, including manifold and turbocharger dynamics	Dynamic
Faster execution	Mapped

Dynamic

`SiEngineCore.slx` contains the engine intake system, exhaust system, core engine, and turbocharger subsystems.



Mapped

SiMappedEngine.slx uses the Mapped SI Engine block to look up power, air mass flow, fuel flow, exhaust temperature, efficiency, and emission performance as functions of engine speed and commanded torque.

Performance Monitor

The reference application contains a Performance Monitor block that you can use to plot steady-state and dynamic results. You can plot:

- Steady-state results as a function of one or two variables.
- Dynamic results using the Simulation Data Inspector.

See Also

SI Core Engine | Mapped SI Engine | SI Controller

Related Examples

- “SI Engine Dynamometer Reference Application” on page 7-15
- “Generate Mapped SI Engine from a Spreadsheet” on page 3-113
- “Generate Deep Learning SI Engine Model” on page 3-117
- “Resize the SI Engine” on page 3-101

More About

- “SI Engine Project Template” on page 4-4
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128

- “Variant Systems”

Build Hybrid Electric Vehicle Multimode Model

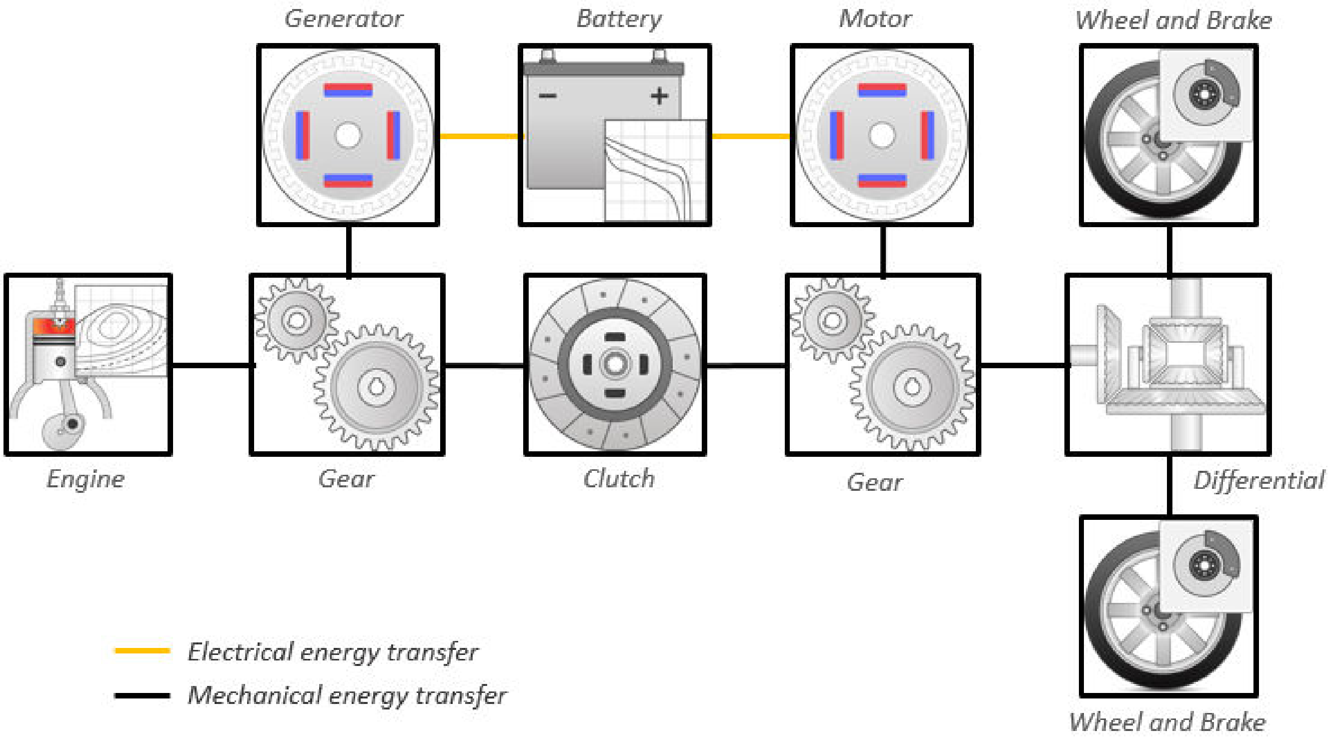
The hybrid electric vehicle reference application represents a full multimode hybrid electric vehicle (HEV) model with an internal combustion engine, transmission, battery, motor, generator, and associated powertrain control algorithms. Use the reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing. To create and open a working copy of the hybrid electric vehicle reference application project, enter

```
autoblkHevStart
```

By default, the HEV multimode reference application is configured with:

- Mapped motor and generator
- 1.5–L spark-ignition (SI) dynamic engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see "Analyze Power and Energy" on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a hybrid control module (HCM) and an engine control module (ECM).	✓
`Passenger Car` subsystem	Implements a hybrid passenger car that contains engine, electric plant, and drivetrain subsystems.	✓

Reference Application Element	Description	Variants
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.

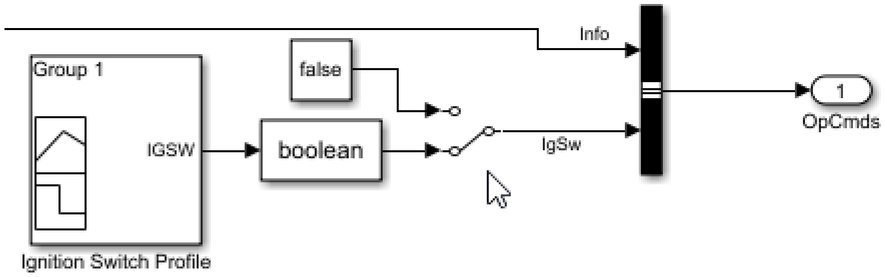
Block Variants			Description
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM with an HCM and an ECM.

ECM

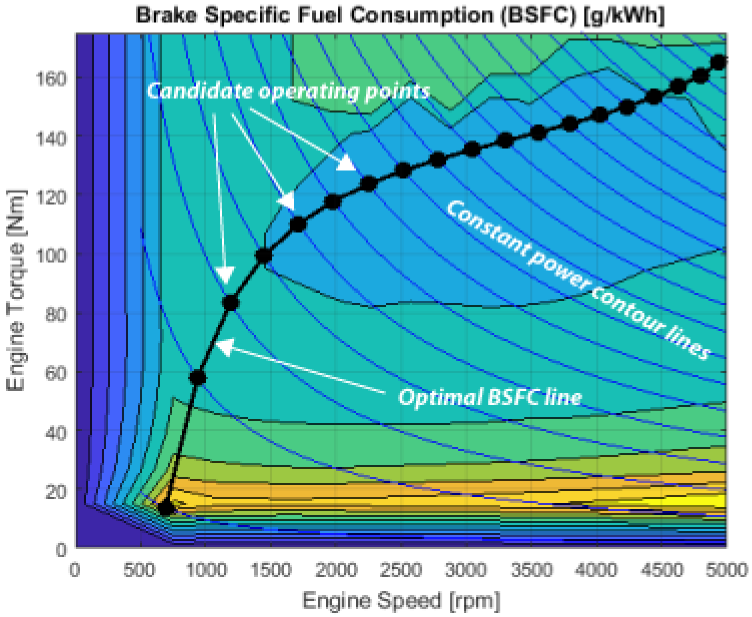
The reference application has these variants for the ECM.

Controller	Variant	Description
ECM	`SiEngineController` (default)	SI engine controller
	`CiEngineController`	CI engine controller

HCM

The HCM implements a dynamic embedded controller that directly determines the engine operating point that minimizes brake-specific fuel consumption (BSFC) while meeting or exceeding power required by the battery charging and vehicle propulsion subsystems.

To calculate the optimal engine operating point in speed and torque, the controller starts with a candidate set of discrete engine power levels. For each power level candidate, the block has a parameterized vector of torque and speed operating points that minimize BSFC.



The optimizer then removes power level candidates that are unacceptable for either of these reasons:

- Too much power sent through the generator to the battery.
- Too little power to meet charging and propulsion subsystem requirements.

Of the remaining power level candidates, the controller selects the one with the lowest BSFC. The controller then sends the associated torque / speed operating point command to the engine.

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these subsystem variants.

Drivetrain

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Vehicle	`Vehicle Body 3 DOF Longitudinal`	Configured for 3 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevMm` (default)	Configured with electric battery
Generator	`GenMapped` (default)	Mapped generator
	`GenDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller
Motor	`MotMapped` (default)	Mapped motor with implicit controller
	`MotDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine
	`SiEngineCoreV`	Dynamic SI V Twin-Turbo Single-Intake Engine
	`SiEngineCoreVNA`	Dynamic SI V Engine
	`SiEngineCoreVThr2`	Dynamic SI V Twin-Turbo Twin-Intake Engine
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiDLEngine`	Deep learning SI engine
	`CiEngine`	Dynamic CI Core Engine with turbocharger
	`CiMappedEngine`	Mapped CI Engine with implicit turbocharger

References

[1] Higuchi, N., Shimada, H., Sunaga, Y., and Tanaka, M., *Development of a New Two-Motor Plug-In Hybrid System*. SAE Technical Paper 2013-01-1476. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2013.

See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “HEV Multimode Reference Application” on page 7-3

More About

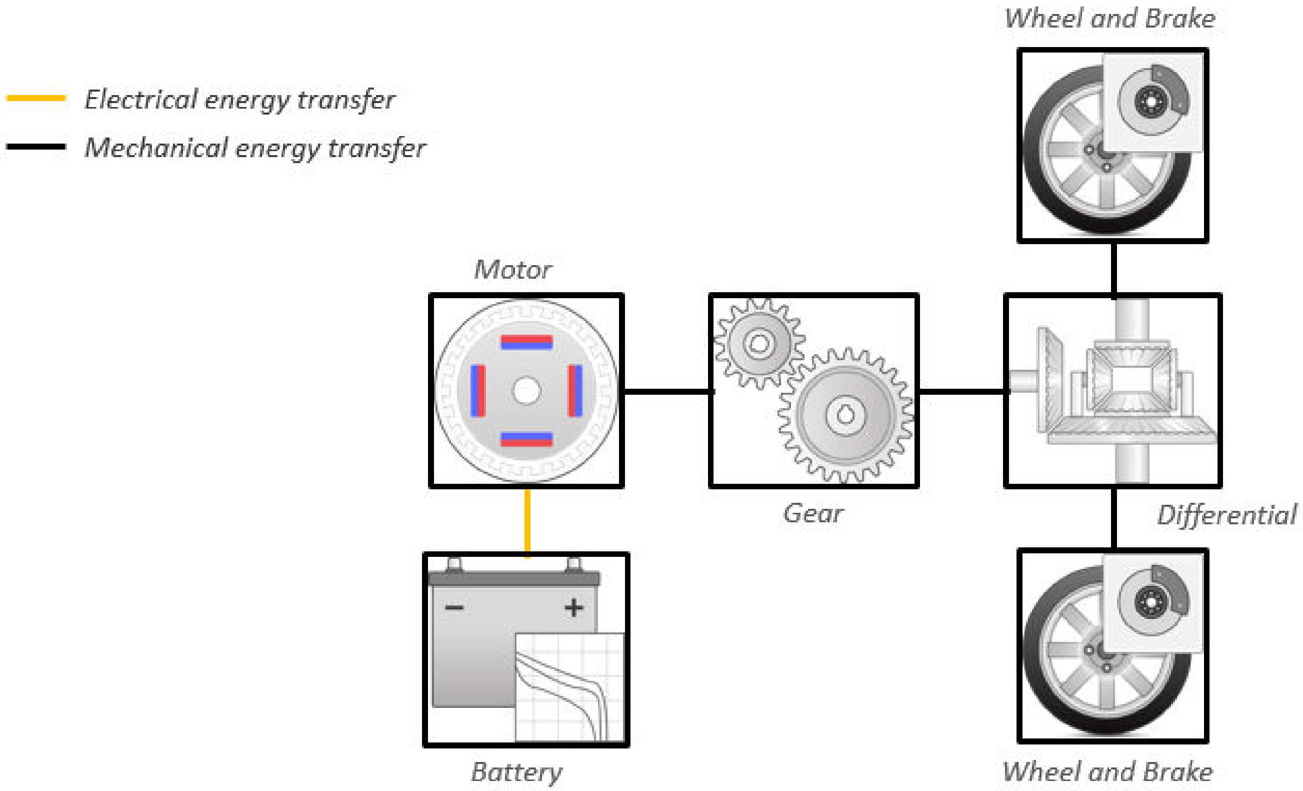
- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Full Electric Vehicle Model

The electric vehicle (EV) reference application represents a full electric vehicle model with a motor-generator, battery, direct-drive transmission, and associated powertrain control algorithms. Use the electric vehicle reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing. To create and open a working copy of the conventional vehicle reference application project, enter

```
autoblkEvStart
```

The electric vehicle reference application is configured with a mapped motor and battery. This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) with regenerative braking, motor torque arbitration and power management.	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a passenger car that contains an electric plant and drivetrain subsystems. To model the electric plant, use the **Toggle To Simscape Electric Plant** button to switch between Simscape and Powertrain Blockset variants of the plant subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), and equivalent fuel economy results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Electric plant and drivetrain plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

For more information about the live script, see “Analyze Power and Energy” on page 3-129.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The Longitudinal Driver subsystem generates normalized acceleration and braking commands. The reference application has these variants.

Block Variants			Description
Longitudinal Driver (default)	Control	Mapped	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		Predictive	Optimal single-point preview (look ahead) control.
		Scalar (default)	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	LPF	Use an LPF on target velocity error for smoother driving.
		pass	Do not use a filter on velocity error.
	Shift	Basic	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		External	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		None (default)	No transmission.
		Scheduled	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

Controllers

To determine the motor torque and brake pressure commands, the reference application implements a supervisory controller. Specifically, the controller subsystem includes a powertrain control module (PCM) with:

- Regenerative braking control
- Motor torque arbitration and power management
  - Converts the driver accelerator pedal signal to a torque request.
  - Converts the driver brake pedal signal to a brake pressure request. The algorithm multiplies the brake pedal signal by a maximum brake pressure.
  - Implements a regenerative braking algorithm for the traction motor to recover the maximum amount of kinetic energy from the vehicle.
  - Implements a virtual battery management system. The algorithm outputs the dynamic discharge and charge power limits as functions of battery state of charge (SOC).
  - Implements a power management algorithm that ensures the battery dynamic discharge and charge power limits are not exceeded.

Regen Braking Control has these variants.

Controller	Variant	Description
Regen Braking Control	Series Regen Brake (default)	Friction braking provides the torque not supplied by regenerative motor braking.
	Parallel Regen Braking	Friction braking and regenerative motor braking independently provide the torque.

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains a drivetrain and electric plant subsystem. The reference application has these variants.

Drivetrain

Drivetrain Subsystem	Variant	Description
Differential and Compliance	All Wheel Drive	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	Front Wheel Drive (default)	
	Rear Wheel Drive	
Vehicle	Vehicle Body 3 DOF Longitudinal	Configured for 3 degrees of freedom
Wheels and Brakes	Longitudinal Wheel - Front 1	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	Longitudinal Wheel - Rear 1	

Electric Plant

To model the electric plant, use the **Toggle To Simscape Electric Plant** button to switch between Simscape and Powertrain Blockset variants of the plant subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Electric Plant Subsystem	Variant	Description
Battery	`BattEv` (default)	Configured with electric battery
Motor	`MotGenEvMapped` (default)	Mapped motor with implicit controller
	`MotGenEvDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller

See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | Mapped Motor

Related Examples

- “EV Reference Application” on page 7-10

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

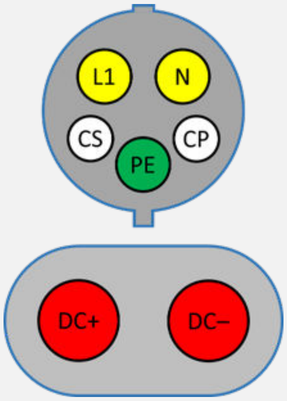
Charge an Electric Vehicle

You can use the electric vehicle (EV) charging reference application to model DC charging of an EV battery from an electric vehicle supply equipment (EVSE) unit, including digital communication between them. The application uses SAE J1772 protocol for basic charging and ISO15118 protocol for high level communication (HLC). The application incorporates a mapped battery block without thermal management.

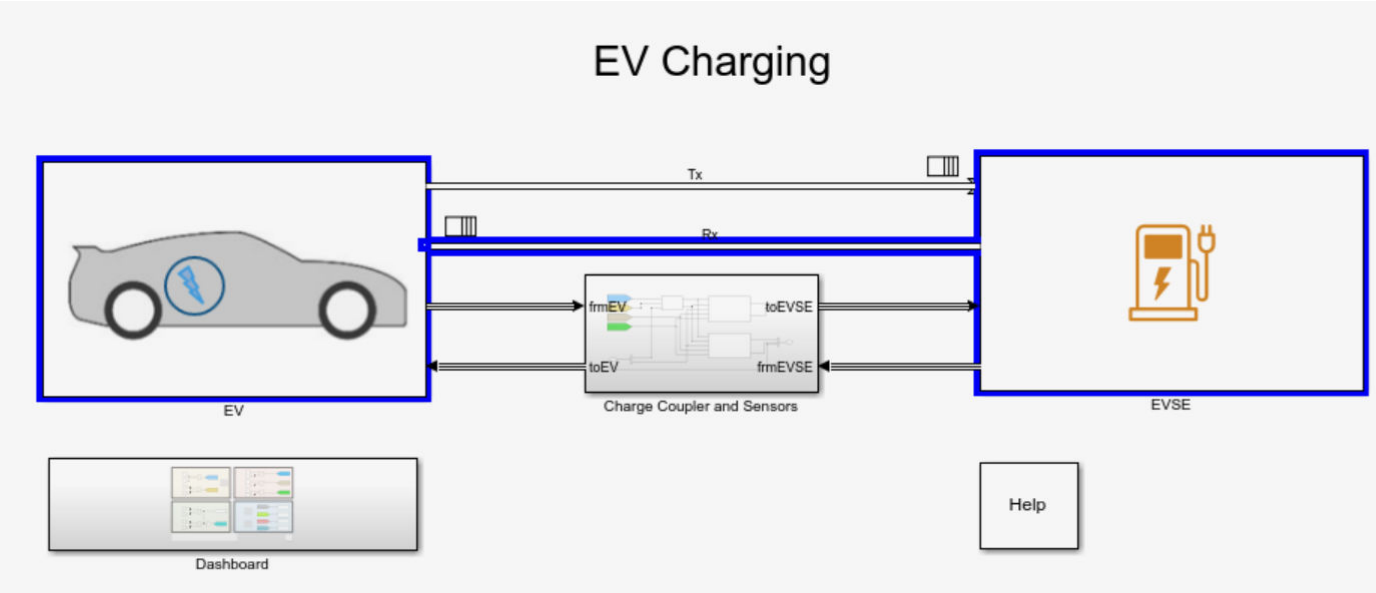
Use the EV charging reference application for control and diagnostic algorithm design and to test digital communication protocols. To create and open a working copy of the reference application project, use this command.

```
autoblkEvChargingStart
```

The application uses the SAE J1772/CCS Combo Coupler, which is the combined charging system (CCS) version of the SAE J1772 coupler. The table shows the coupler and describes the pins used for communication and charging.

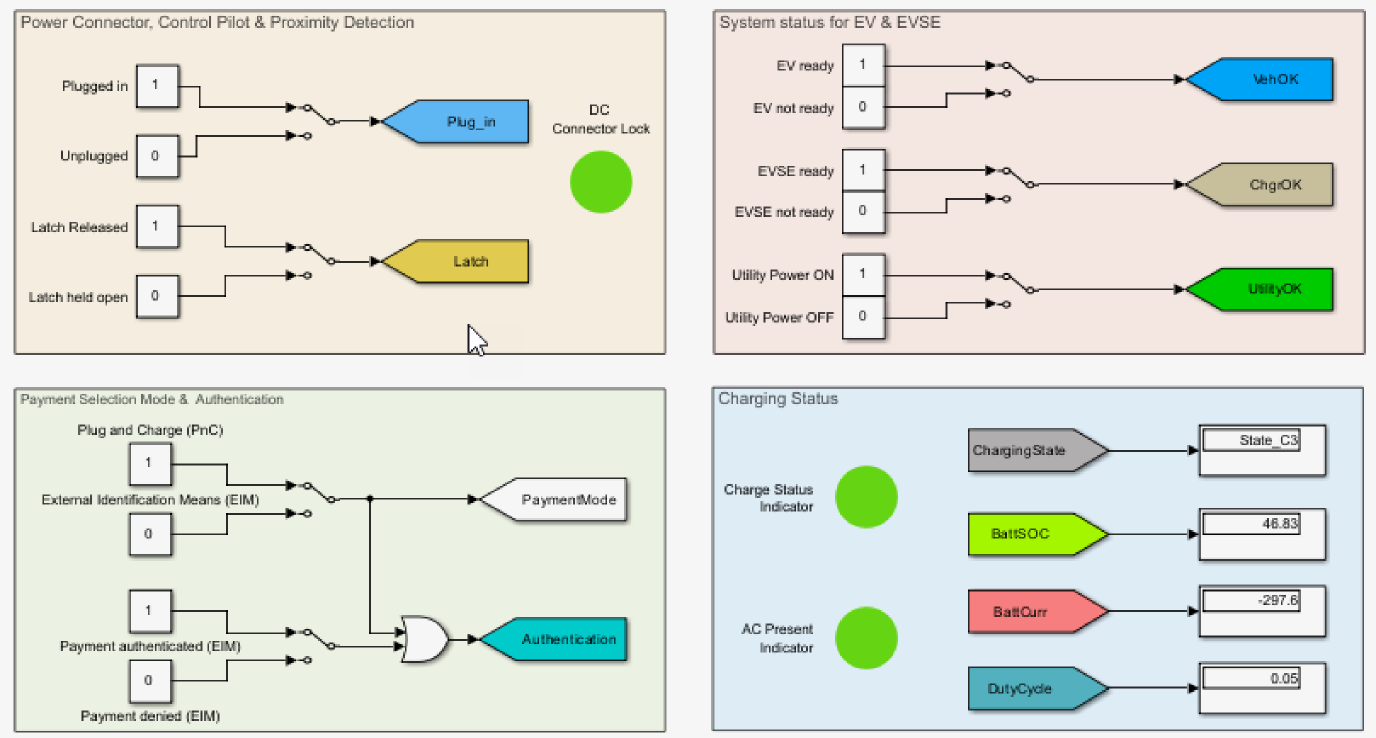
Pin Label	Name	Description	J1772/CCS Combo Coupler
L1	Line (AC1)	Single-phase AC power	
N	Neutral (AC2)	Neutral single-phase AC power	
PE	Ground (Protective Earth)	Common ground	
CP	Control Pilot	Detects contact between connector and plug and carries the two way communication on a 1 kHz PWM signal	
CS	Proximity Detection (Connection Signal)	Detects proximity of connector to plug, even if not fully inserted, enabling warning of damage to connection hardware if vehicle or EVSE moves	
DC+	DC Power Terminal (+)	Positive DC power connection	
DC-	DC Power Terminal (-)	Negative DC power connection	

To start charging, click **Run**. You can choose one of two views of the process. This first view shows the digital communications between the EV and the EVSE.



The app defaults to the second view, called the dashboard. This view uses less animation than the first, so the app runs faster with the dashboard displayed. The dashboard shows the states of the charging coupler, the readiness of the EV and EVSE to charge, and the authentication and authorization to charge. Numerical displays show the rate and progress of the charge.

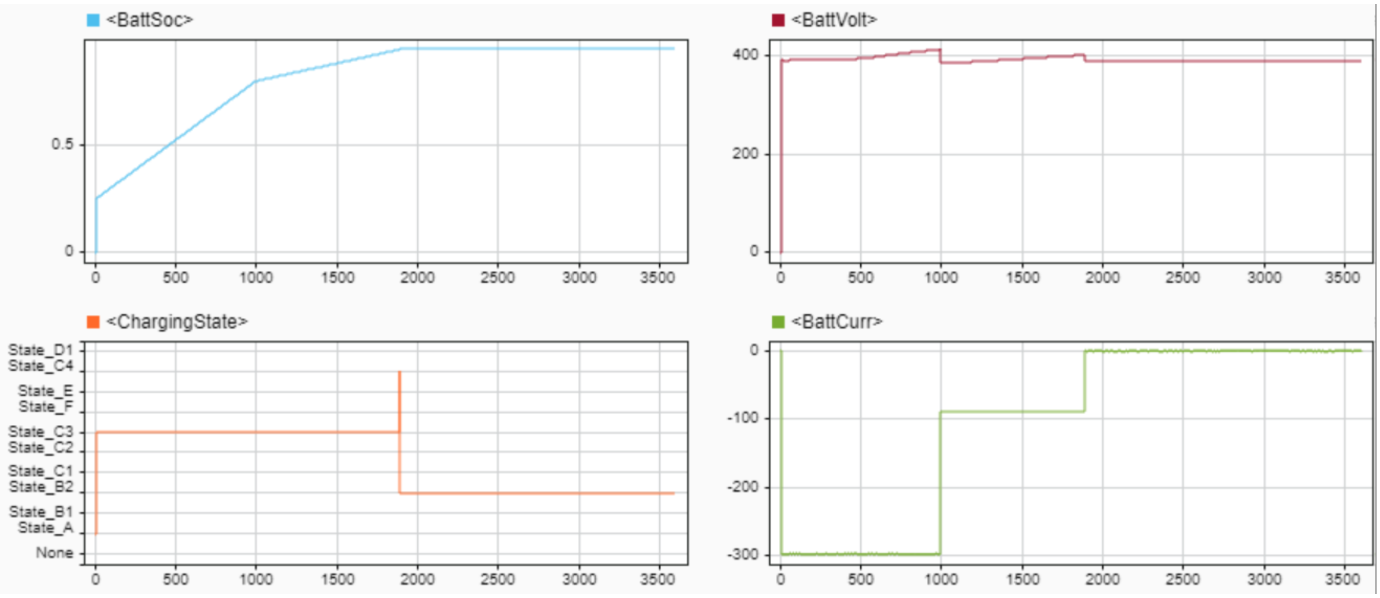
As you develop your model, you can replace the manual switches with timed or triggered ones and verify their actions.



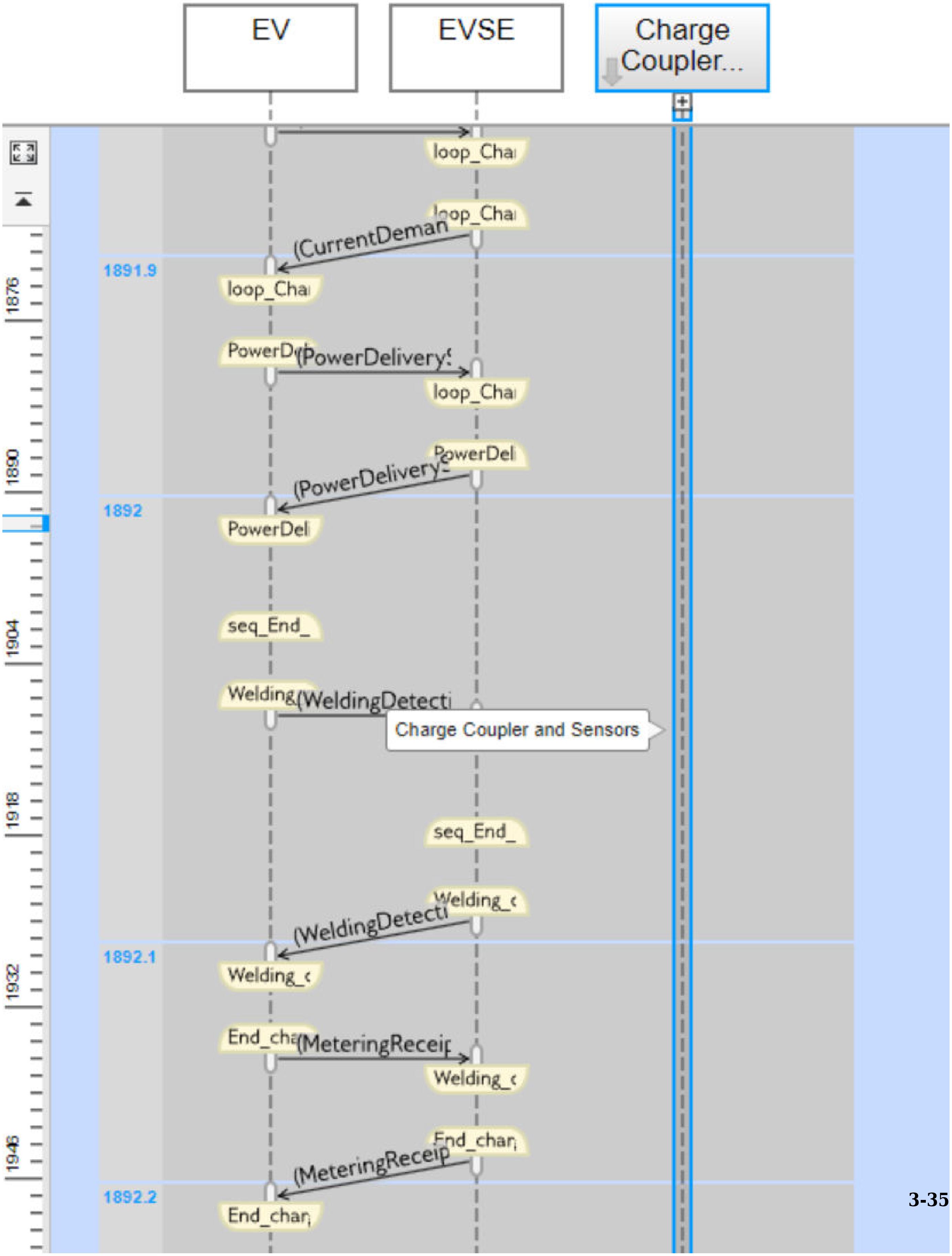
Here is a partial list of the `Charging_State` values and their descriptions.

State Designation	Description
A	Vehicle not connected
B1	Vehicle connected / not ready to accept energy EVSE not ready to supply energy
B2	Vehicle connected / not ready to accept energy EVSE capable to supply energy
C	Vehicle connected / ready to accept energy / indoor charging area ventilation not required EVSE capable to supply energy
D	Vehicle connected / ready to accept energy / indoor charging area ventilation required EVSE capable to supply energy
E	EVSE disconnected from vehicle / EVSE disconnected from utility, EVSE loss of utility power or control pilot short to control pilot reference
F	Other EVSE problem

Once charging is complete, you can display the results in the Simulation Data Inspector versus time. You can select results for battery current, battery state of charge, charging status, and others.



You can also use the Sequence Viewer in the Simulation Data Inspector to see the communications between the EV and the EVSE.


EV
EVSE
Charge Coupler...
loop_Chai
loop_Chai
(CurrentDeman
1891.9
loop_Chai
PowerDe
(PowerDelivery!
loop_Chai
PowerDel
(PowerDelivery
1892
PowerDel
seq_End_
Welding
(WeldingDetecti
Charge Coupler and Sensors
seq_End_
Welding_c
(WeldingDetecti
1892.1
Welding_c
End_cha
(MeteringReceip
Welding_c
End_char
(MeteringReceip
1892.2
End_char
1876
1890
1904
1918
1932
1946

See Also

Interior PMSM | Drive Cycle Source | Mapped Motor

Related Examples

- "EV Charging Reference Application" on page 7-11

More About

- "Variant Systems"

External Websites

- SAE Standards

Build Fuel Cell Electric Vehicle

To design an energy system for a hydrogen-based vehicle, use the fuel cell reference application project with a high-fidelity fuel cell model in Simscape. You can switch between a detailed and a mapped fuel cell. Use these models for design tradeoff analysis and component sizing, control parameter optimization, and hardware-in-the-loop (HIL) testing. To create and open a working copy of the reference application project, enter

```
autoblkFCEvStart
```

This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Generate Mapped Fuel Cell from Spreadsheet	Use the Model-Based Calibration Toolbox to create a mapped fuel cell model from measured fuel cell performance data stored in a spreadsheet. For more information, see “Generate Mapped Fuel Cell from a Spreadsheet” on page 3-133.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	

Reference Application Element	Description	Variants
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) with regenerative braking, motor torque arbitration and power management.	✓
`Passenger Car` subsystem	Implements a passenger car that contains an electric plant and drivetrain subsystems.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), and equivalent fuel economy results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Electric plant and drivetrain plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

For more information about the live script, see “Analyze Power and Energy” on page 3-129.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar` (default)	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None` (default)	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

Controllers

To determine the motor torque and brake pressure commands, the reference application implements a supervisory controller. Specifically, the controller subsystem includes a powertrain control module (PCM) with:

- Regenerative braking control
- Motor torque arbitration and power management
    - Converts the driver accelerator pedal signal to a torque request.

- Converts the driver brake pedal signal to a brake pressure request. The algorithm multiplies the brake pedal signal by a maximum brake pressure.
- Implements a regenerative braking algorithm for the traction motor to recover the maximum amount of kinetic energy from the vehicle.
- Implements a virtual battery management system. The algorithm outputs the dynamic discharge and charge power limits as functions of battery state of charge (SOC).
- Implements a power management algorithm that ensures the battery dynamic discharge and charge power limits are not exceeded.

Regen Braking Control has these variants.

Controller	Variant	Description
Regen Braking Control	`Series Regen Brake` (default)	Friction braking provides the torque not supplied by regenerative motor braking.
	`Parallel Regen Braking`	Friction braking and regenerative motor braking independently provide the torque.

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains a drivetrain and electric plant subsystem. The reference application has these variants.

Drivetrain

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Vehicle	`Vehicle Body 3 DOF Longitudinal`	Configured for 3 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles,

Drivetrain Subsystem	Variant	Description
	Longitudinal Wheel - Rear 1	which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	BattEv (default)	Configured with electric battery
Motor	MotGenEvMapped (default)	Mapped motor with implicit controller
	MotGenEvDynamic	Interior permanent magnet synchronous motor (PMSM) with controller

See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | Mapped Motor

Related Examples

- “FCEV Reference Application” on page 7-12

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle Input Power-Split Model

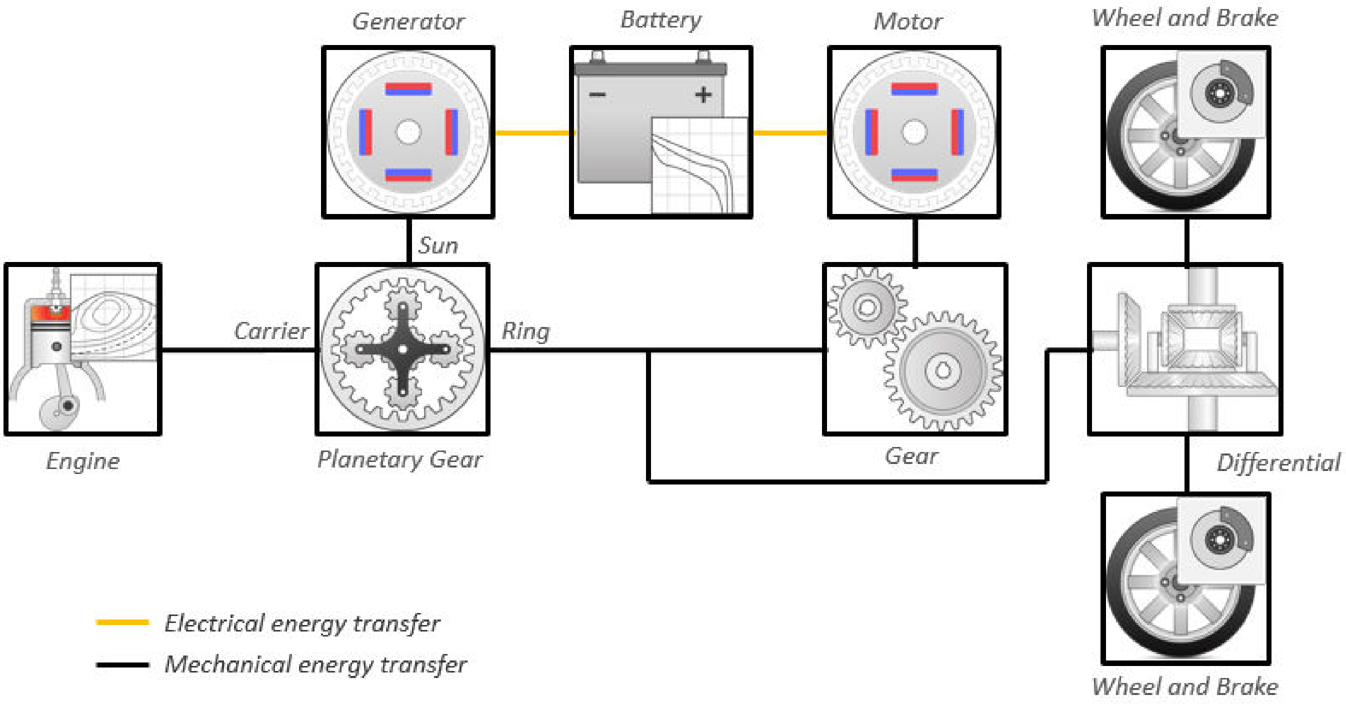
The hybrid electric vehicle (HEV) input power-split reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, generator, and associated powertrain control algorithms. Use the HEV input power-split reference application for HIL testing, tradeoff analysis, and control parameter optimization of a power-split hybrid like the Toyota® Prius®. To create and open a working copy of the HEV input power-split reference application project, enter

```
autoblkHevIpsStart
```

By default, the HEV input power-split reference application is configured with:

- Nickel-metal hydride (NiMH) battery pack
- Mapped electric motors
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see "Analyze Power and Energy" on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing an input power-split hybrid control module (HCM) and an engine control module (ECM).	✓
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems.	✓

Reference Application Element	Description	Variants
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see "Analyze Power and Energy" on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.

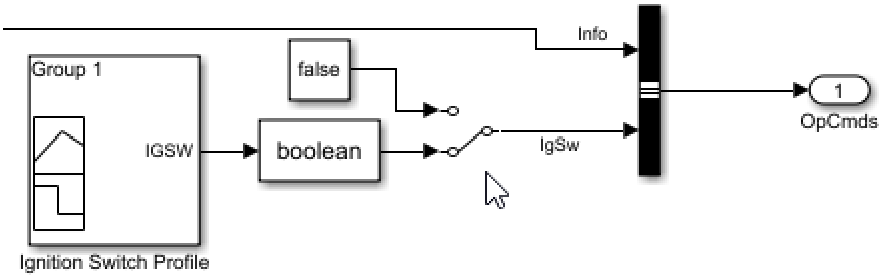
Block Variants			Description
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

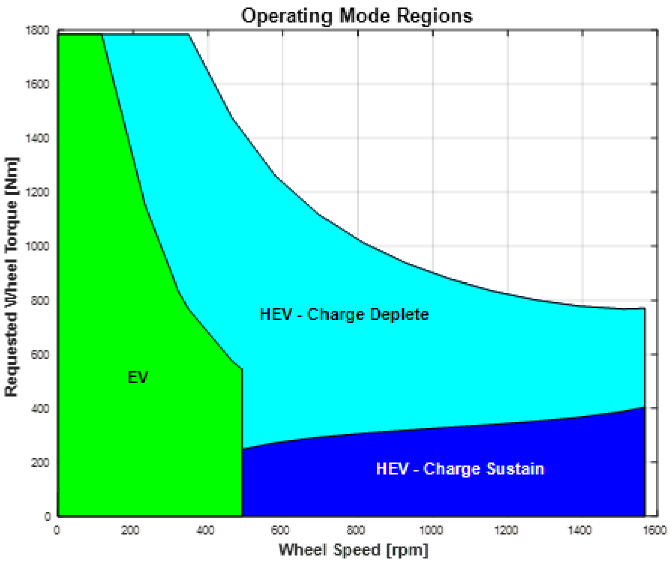
Controllers

The `Controller` subsystem has a PCM containing an input power-split HCM and an ECM. The controller has these variants.

Controller	Variant	Description
ECM	`SiEngineController` (default)	SI engine controller
Input power split HCM	`Series Regen Brake` (default)	Friction braking provides the torque not supplied by regenerative motor braking.
	`Parallel Regen Braking`	Friction braking and regenerative motor braking independently provide the torque.

The input-power split HCM implements a dynamic supervisory controller that determines the engine torque, generator torque, motor torque, and brake pressure commands. Specifically, the input power-split HCM:

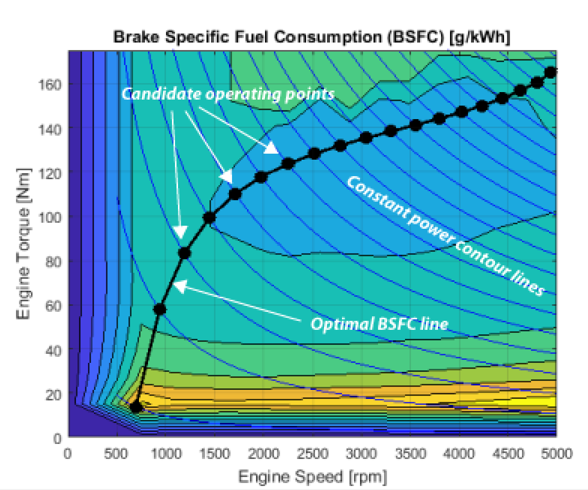
- Converts the driver accelerator pedal signal to a wheel torque request. The algorithm uses the optimal engine torque and maximum motor torque curves to calculate the total powertrain torque at the wheels.
- Converts the driver brake pedal signal to a brake pressure request. The algorithm multiplies the brake pedal signal by a maximum brake pressure.
- Implements a regenerative braking algorithm for the traction motor to recover the maximum amount of kinetic energy from the vehicle.
- Implements a virtual battery management system. The algorithm outputs the dynamic discharge and charge power limits as functions of battery SOC.
- Determines the vehicle operating mode through a set of rules and decision logic implemented in Stateflow. The operating modes are functions of wheel speed and requested wheel torque. The algorithm uses the wheel power request, accelerator pedal, battery SOC, and vehicle speed rules to transition between electric vehicle (EV) and HEV modes.



Mode	Description
EV	Traction motor provides the wheel torque request.

Mode	Description
HEV – Charge Sustaining (Low Power)	• Engine provides the wheel torque request. • Torque blending algorithm transitions the torque production from the EV motor to the HEV engine. The algorithm allows the motor to ramp down the torque while the engine torque ramps up. Once the blending is complete, the motor can start sustaining the charge (negative torque), if needed. • Based on the target battery SOC and available kinetic energy, the HEV mode determines a charge sustain power level. The mode includes the additional charge power in the engine power command. To provide the desired charge power, the traction motor acts as a generator. • Depending on the instantaneous speeds of the engine and motor, the generator may consume energy while regulating the engine speed. In this case, the motor provides the additional charge sustaining power.
HEV – Charge Depleting (High Power)	• Engine provides the wheel power request up to its maximum output. • If the wheel torque request is greater than the engine torque output at the wheels, the traction motor provides the remainder of the wheel torque request.
Stationary	While the vehicle is at rest, the engine and generator can provide optional charging if battery SOC is below a minimum SOC value.

- Controls the motor, generator, and engine through a set of rules and decision logic implemented in Stateflow.

Control	Description
Engine	• Decision logic determines the engine operation modes (off, start, run). • In engine run mode, lookup tables determine the engine torque and engine speed that optimizes the break-specific fuel consumption (BSFC) for a given engine power request. The ECM uses the optimal engine torque command. The generator control uses the optimal engine speed command. 

Control	Description
Generator	- As determined by the HCM, the generator either starts the engine or regulates the engine speed. To regulate the engine speed, the generator uses a PI controller. - A rule-based power management algorithm calculates a generator torque that does not exceed the dynamic power limits.
Motor	A rule-based power management algorithm calculates a motor torque that does not exceed the dynamic power limits.

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these subsystem variants.

Drivetrain

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Gearbox	`Ideal Fixed Gear Transmission`	Configure gearbox efficiency with a constant (default) or 3D lookup table.
Vehicle	`Vehicle Body 3 DOF Longitudinal`	Configured for 3 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: - Brake - Force calculation - Resistance calculation - Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique

Drivetrain Subsystem	Variant	Description
	`Longitudinal Wheel - Rear 1`	blocks to model each wheel increases model complexity and computational cost.

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery and DC-DC Converter	`BattHevIps`	Configured with NiMH battery
Generator	`GenMapped` (default)	Mapped generator with implicit controller
	`GenDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller
Motor	`MotMapped` (default)	Mapped motor with implicit controller
	`MotDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller

Engine

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

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Burress, T. A. et al, *Evaluation of the 2010 Toyota Prius Hybrid Synergy Drive System*. Technical Report ORNL/TM-2010/253. U.S. Department of Energy, Oak Ridge National Laboratory, March 2011.

[3] Rask, E., Duoba, M., Loshse-Busch, H., and Bocci, D., *Model Year 2010 (Gen 3) Toyota Prius Level-1 Testing Report*. Technical Report ANL/ES/RP-67317. U.S. Department of Energy, Argonne National Laboratory, September 2010.

See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- "HEV Input Power-Split Reference Application" on page 7-4

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle P0 Model

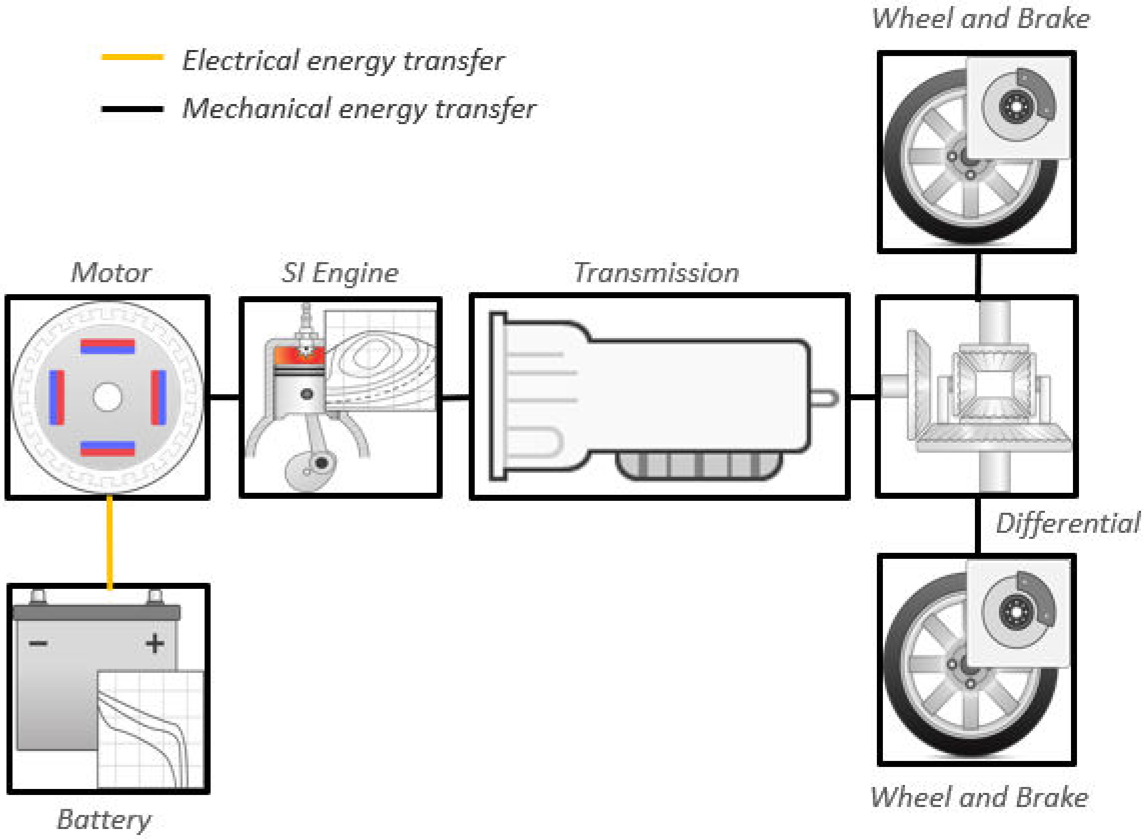
The hybrid electric vehicle (HEV) P0 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P0 hybrid. To create and open a working copy of the reference application project, enter

```
autoblkHevP0Start
```

By default, the HEV P0 reference application is configured with:

- Lithium-ion battery pack
- Mapped electric motor
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a P0 hybrid control module (HCM), an engine control module (ECM), and a transmission control module (TCM).	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems. To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

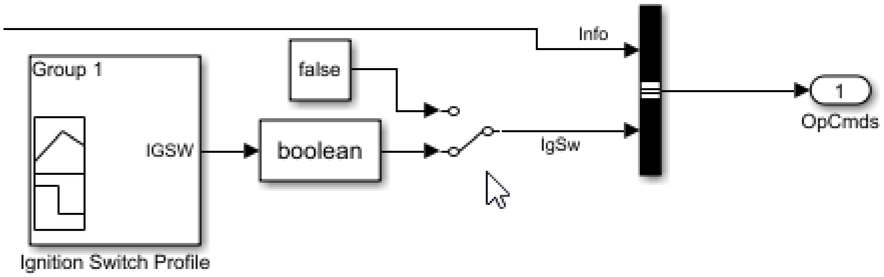
Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM containing an ECM, HCM, and TCM. The controller has these variants.

Controller	Variant	Description
ECM	`SiEngineController` (default)	Implements the SI Controller
	`CiEngineController`	Implements the CI Controller
HCM	`ECMS`	Implements the Equivalent Consumption Minimization Strategy
TCM	`TransmissionController`	Implements the transmission controller

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these subsystem variants.

Drivetrain

To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Tip The reference application sets the appropriate solvers to optimize performance for each engine and drivetrain combination. Select the engine variant first, then select the drivetrain using the toggle button. If you select the drivetrain before changing the engine, you may encounter a solver error.

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Torque Converter Automatic Transmission	`Ideal Fixed Gear Transmission`	Configure locked and unlocked transmission efficiency with either a 1D or 4D (default) lookup table.
	`Torque Converter`	Configure for external, internal (default), or no lockup.
Vehicle	`Vehicle Body 1 DOF Longitudinal`	Configured for 1 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevP0`	Configured with Lithium Ion battery
Electric Machine	`MotMapped`	Mapped Motor with implicit controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine

Limitations

MathWorks used the SI Core Engine and SI Controller to calibrate the hybrid control module (HCM). If you use the CI Core Engine and CI Controller variants, the simulation may error because the HCM does not use calibrated results.

Acknowledgment

MathWorks would like to acknowledge the contribution of Dr. Simona Onori to the ECMS optimal control algorithm implemented in this reference application. Dr. Onori is a Professor of Energy Resources Engineering at Stanford University. Her research interests include electrochemical modeling, estimation and optimization of energy storage devices for automotive and grid-level applications, hybrid and electric vehicles modeling and control, PDE modeling, and model-order reduction and estimation of emission mitigation systems. She is a senior member of IEEE®.

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Onori, S., Serrao, L., and Rizzoni, G., *Hybrid Electric Vehicles Energy Management Systems*. New York: Springer, 2016.

See Also

Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Core Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller | Mapped Motor

Related Examples

- “HEV P0 Reference Application” on page 7-5

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle P1 Model

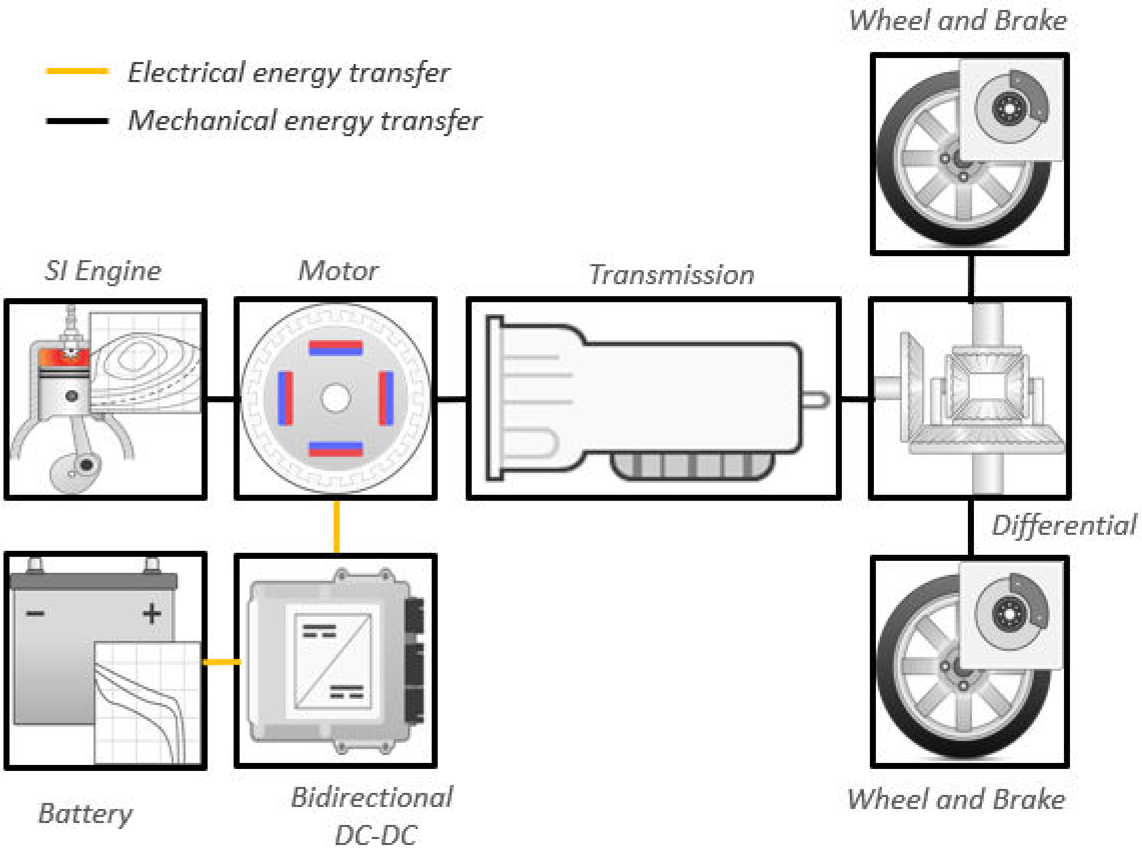
The hybrid electric vehicle (HEV) P1 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P1 hybrid. To create and open a working copy of the reference application project, enter

```
autoblkHevP1Start
```

By default, the HEV P1 reference application is configured with:

- Lithium-ion battery pack
- Mapped electric motor
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	**Description**	**Variants**
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a P1 hybrid control module (HCM), an engine control module (ECM), and a transmission control module (TCM).	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems. To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

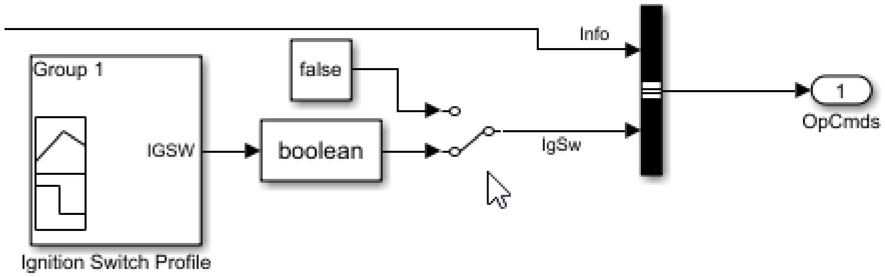
Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM containing an ECM, HCM, and TCM. The controller has these variants.

Controller	Variant	Description
ECM	`SiEngineController` (default)	Implements the SI Controller
	`CiEngineController`	Implements the CI Controller
HCM	`ECMS`	Implements the Equivalent Consumption Minimization Strategy
TCM	`TransmissionController`	Implements the transmission controller

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these subsystem variants.

Drivetrain

To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Tip The reference application sets the appropriate solvers to optimize performance for each engine and drivetrain combination. Select the engine variant first, then select the drivetrain using the toggle button. If you select the drivetrain before changing the engine, you may encounter a solver error.

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Torque Converter Automatic Transmission	`Ideal Fixed Gear Transmission`	Configure locked and unlocked transmission efficiency with either a 1D or 4D (default) lookup table.
	`Torque Converter`	Configure for external, internal (default), or no lockup.
Vehicle	`Vehicle Body 1 DOF Longitudinal`	Configured for 1 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevP1`	Configured with Lithium Ion battery
Electric Machine	`MotMapped`	Mapped Motor with implicit controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine

Limitations

MathWorks used the SI Core Engine and SI Controller to calibrate the hybrid control module (HCM). If you use the CI Core Engine and CI Controller variants, the simulation may error because the HCM does not use calibrated results.

Acknowledgment

MathWorks would like to acknowledge the contribution of Dr. Simona Onori to the ECMS optimal control algorithm implemented in this reference application. Dr. Onori is a Professor of Energy Resources Engineering at Stanford University. Her research interests include electrochemical modeling, estimation and optimization of energy storage devices for automotive and grid-level applications, hybrid and electric vehicles modeling and control, PDE modeling, and model-order reduction and estimation of emission mitigation systems. She is a senior member of IEEE.

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Onori, S., Serrao, L., and Rizzoni, G., *Hybrid Electric Vehicles Energy Management Systems*. New York: Springer, 2016.

See Also

Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Core Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller | Mapped Motor

Related Examples

- “HEV P1 Reference Application” on page 7-6

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle P2 Model

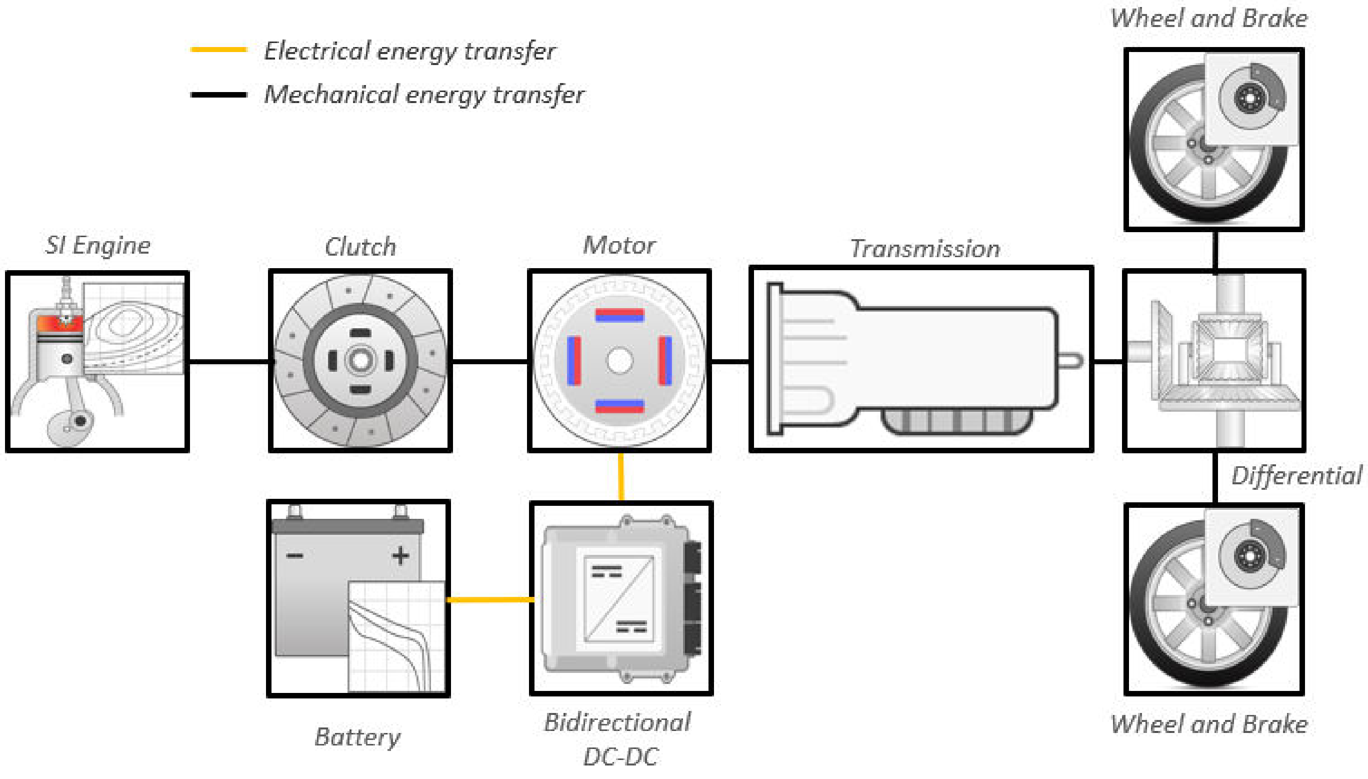
The hybrid electric vehicle (HEV) P2 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P2 hybrid. To create and open a working copy of the reference application project, enter

```
autoblkHevP2Start
```

By default, the HEV P2 reference application is configured with:

- Lithium-ion battery pack
- Mapped electric motor
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a P2 hybrid control module (HCM), an engine control module (ECM), and a transmission control module (TCM).	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems. To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

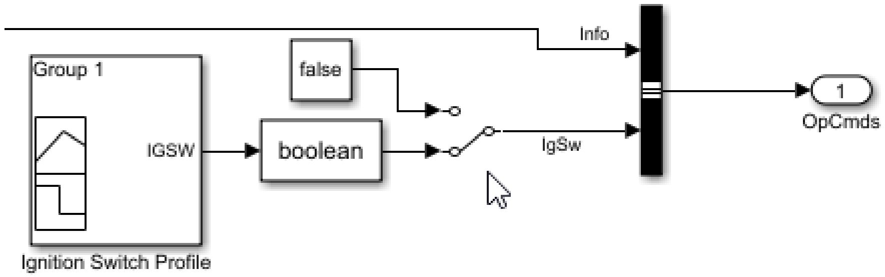
Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM containing an ECM, HCM, and TCM. The controller has these variants.

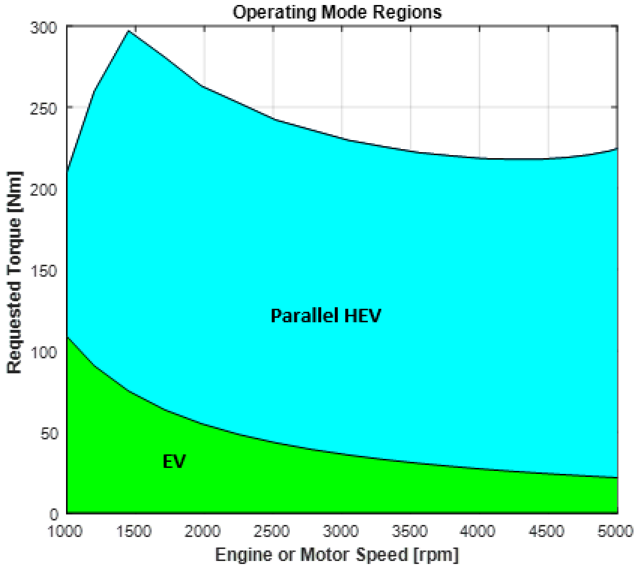
Controller	Variant		Description
ECM	`SiEngineController` (default)		Implements the SI Controller
	`CiEngineController`		Implements the CI Controller
TCM	`TransmissionController`		Implements the transmission controller
HCM	`Optimal Control` (default)	`Energy Management System`	Implements the Equivalent Consumption Minimization Strategy
	`Rule-Based Control`	`P2 Supervisory Control`	Implements a dynamic supervisory controller that determines the engine torque, motor torque, starter, clutch, and brake pressure commands.
		`Regen Braking Control`	Implements a parallel or series regenerative braking controller during rule-based control.

Rule-Based Control

The HCM implements a dynamic supervisory controller that determines the engine torque, motor torque, starter, clutch, and brake pressure commands. Specifically, the HCM:

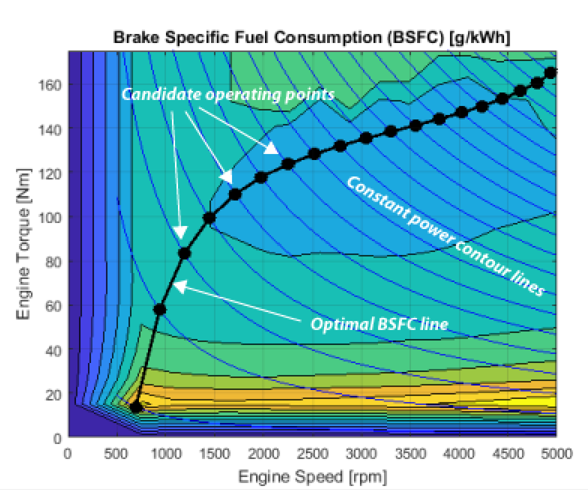
- Converts the driver accelerator pedal signal to a torque request. The algorithm uses the optimal engine torque and maximum motor torque curves to calculate the total powertrain torque.
- Converts the driver brake pedal signal to a brake pressure request. The algorithm multiplies the brake pedal signal by a maximum brake pressure.
- Implements a regenerative braking algorithm for the traction motor to recover the maximum amount of kinetic energy from the vehicle.
- Implements a virtual battery management system. The algorithm outputs the dynamic discharge and charge power limits as functions of battery SOC.

The HCM determines the vehicle operating mode through a set of rules and decision logic implemented in Stateflow. The operating modes are functions of motor speed and requested torque. The algorithm uses the calculated power request, accelerator pedal, battery SOC, and vehicle speed rules to transition between electric vehicle (EV) and parallel HEV modes.



Mode	Description
EV	Traction motor provides the torque request.
Parallel HEV	The engine and the motor split the power request. Based on the target battery SOC and available kinetic energy, the HEV mode determines a charge sustain power level. The parallel HEV mode adds the charge sustain power to the engine power command. To provide the desired charge sustain power, the traction motor acts as a generator if charging is needed, and as a motor if discharging is needed. If the power request is greater than the engine power, the traction motor provides the remainder of the power request.
Stationary	While the vehicle is at rest, the engine and generator can provide optional charging if battery SOC is below a minimum SOC value.

The HCM controls the motor, and engine through a set of rules and decision logic implemented in Stateflow.

<table>
<tr><th>Control</th><th>Description</th></tr>
<tr><td>Engine</td><td>• Decision logic determines the engine operation modes (off, start, on).
• To start the engine in engine start (stationary) mode, the motor closes clutch 1 and puts the transmission in neutral. If the high-voltage battery SOC is low, the mode uses the low-voltage starter motor.
• To start the engine in engine start (driving) mode, the mode uses the low-voltage starter motor with clutch 1 open. To connect the driveline, the engine controller matches the engine and motor speeds and closes clutch 1.
• In engine on (stationary) mode, lookup tables determine the engine torque and engine speed that optimizes the brake-specific fuel consumption (BSFC) for a given engine power request. The ECM uses the optimal engine torque command. The motor control uses the optimal engine speed command.

• In engine on (parallel HEV) mode, a lookup table determines the engine torque for a given engine power. However, because the drivetrain couples the engine and wheel speeds, engine on mode might not operate at speeds that minimize BSFC.</td></tr>
<tr><td>Motor</td><td>A rule-based power management algorithm calculates a motor torque that does not exceed the dynamic power limits.</td></tr>
</table>

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these variants.

Drivetrain

To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Tip The reference application sets the appropriate solvers to optimize performance for each engine and drivetrain combination. Select the engine variant first, then select the drivetrain using the toggle button. If you select the drivetrain before changing the engine, you may encounter a solver error.

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Torque Converter Automatic Transmission	`Ideal Fixed Gear Transmission`	Configure locked and unlocked transmission efficiency with either a 1D or 4D (default) lookup table.
	`Torque Converter`	Configure for external, internal (default), or no lockup.
Vehicle	`Vehicle Body 1 DOF Longitudinal`	Configured for 1 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevP2`	Configured with Lithium Ion battery and DC-DC converter

Electric Plant Subsystem	Variant	Description
Low Voltage Starting System	`StarterSystemP2`	Configured with a low voltage starting system
Motor	`MotMapped` (default)	Mapped Motor with implicit controller
	`MotDynamic`	Interior permanent magnet synchronous motor (PMSM) with controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine

Limitations

MathWorks used the SI Core Engine and SI Controller to calibrate the hybrid control module (HCM). If you use the CI Core Engine and CI Controller variants, the simulation may error because the HCM does not use calibrated results.

Acknowledgment

MathWorks would like to acknowledge the contribution of Dr. Simona Onori to the ECMS optimal control algorithm implemented in this reference application. Dr. Onori is a Professor of Energy Resources Engineering at Stanford University. Her research interests include electrochemical modeling, estimation and optimization of energy storage devices for automotive and grid-level applications, hybrid and electric vehicles modeling and control, PDE modeling, and model-order reduction and estimation of emission mitigation systems. She is a senior member of IEEE.

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Onori, S., Serrao, L., and Rizzoni, G., *Hybrid Electric Vehicles Energy Management Systems*. New York: Springer, 2016.

See Also

Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Core Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller | Mapped Motor

Related Examples

- “HEV P2 Reference Application” on page 7-7

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle P3 Model

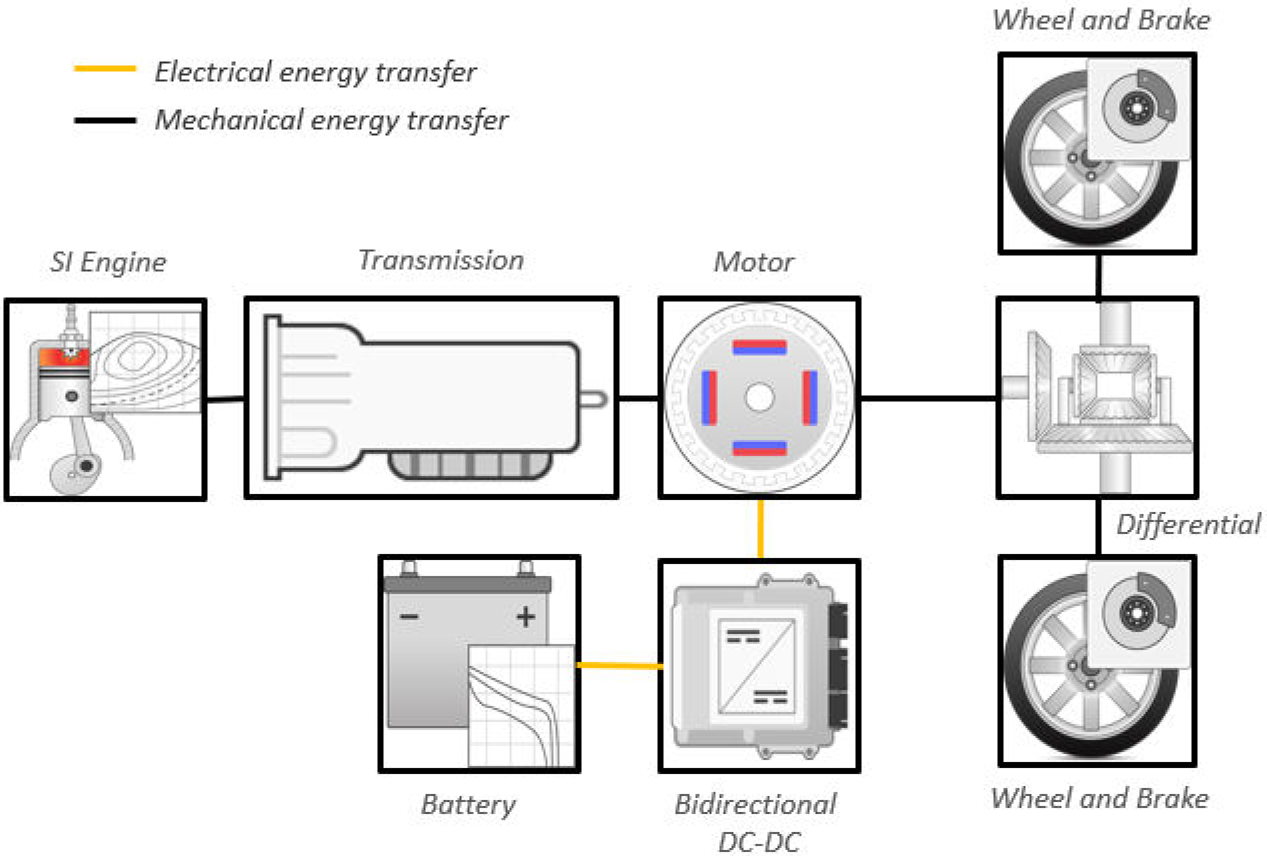
The hybrid electric vehicle (HEV) P3 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P3 hybrid. To create and open a working copy of the reference application project, enter

```
autoblkHevP3Start
```

By default, the HEV P3 reference application is configured with:

- Lithium-ion battery pack
- Mapped electric motor
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a P3 hybrid control module (HCM), an engine control module (ECM), and a transmission control module (TCM).	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems. To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

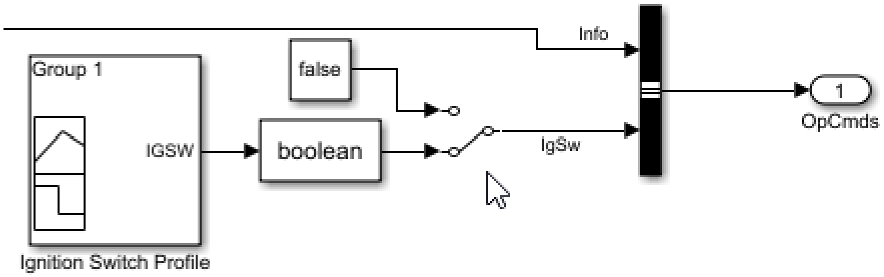
Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM containing an ECM, HCM, and TCM. The controller has these variants.

Controller	Variant	Description
ECM	`SiEngineController` (default)	Implements the SI Controller
	`CiEngineController`	Implements the CI Controller
HCM	`ECMS`	Implements the Equivalent Consumption Minimization Strategy
TCM	`TransmissionController`	Implements the transmission controller

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these subsystem variants.

Drivetrain

To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Tip The reference application sets the appropriate solvers to optimize performance for each engine and drivetrain combination. Select the engine variant first, then select the drivetrain using the toggle button. If you select the drivetrain before changing the engine, you may encounter a solver error.

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`All Wheel Drive`	Configure drivetrain for all wheel, front wheel, or rear wheel drive. For the all wheel drive variant, you can configure the type of coupling torque.
	`Front Wheel Drive` (default)	
	`Rear Wheel Drive`	
Torque Converter Automatic Transmission	`Ideal Fixed Gear Transmission`	Configure locked and unlocked transmission efficiency with either a 1D or 4D (default) lookup table.
	`Torque Converter`	Configure for external, internal (default), or no lockup.
Vehicle	`Vehicle Body 1 DOF Longitudinal`	Configured for 1 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevP3`	Configured with Lithium Ion battery
Electric Machine	`MotMapped`	Mapped Motor with implicit controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine

Limitations

MathWorks used the SI Core Engine and SI Controller to calibrate the hybrid control module (HCM). If you use the CI Core Engine and CI Controller variants, the simulation may error because the HCM does not use calibrated results.

Acknowledgment

MathWorks would like to acknowledge the contribution of Dr. Simona Onori to the ECMS optimal control algorithm implemented in this reference application. Dr. Onori is a Professor of Energy Resources Engineering at Stanford University. Her research interests include electrochemical modeling, estimation and optimization of energy storage devices for automotive and grid-level applications, hybrid and electric vehicles modeling and control, PDE modeling, and model-order reduction and estimation of emission mitigation systems. She is a senior member of IEEE.

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Onori, S., Serrao, L., and Rizzoni, G., *Hybrid Electric Vehicles Energy Management Systems*. New York: Springer, 2016.

See Also

Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Core Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller | Mapped Motor

Related Examples

- “HEV P3 Reference Application” on page 7-8

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Build Hybrid Electric Vehicle P4 Model

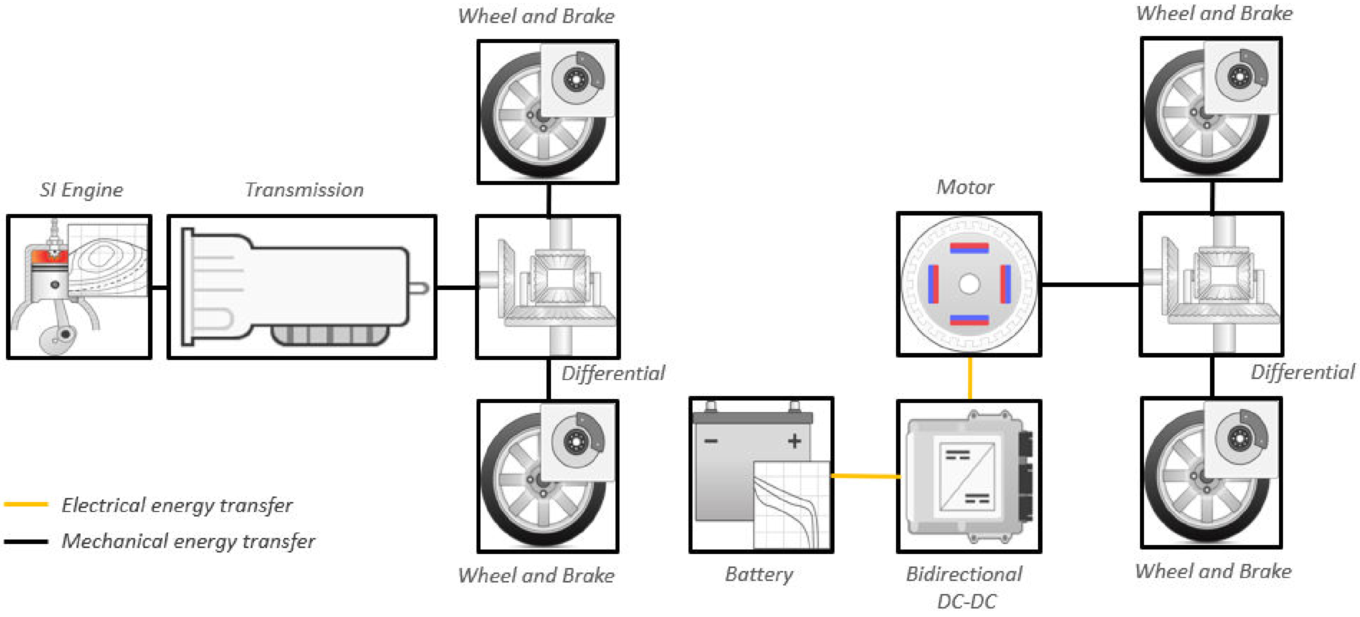
The hybrid electric vehicle (HEV) P4 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P4 hybrid. To create and open a working copy of the reference application project, enter

```
autoblkHevP4Start
```

By default, the HEV P4 reference application is configured with:

- Lithium-ion battery pack
- Mapped electric motor
- Mapped spark-ignition (SI) engine

This diagram shows the powertrain configuration.



This table describes the blocks and subsystems in the reference application, indicating which subsystems contain variants. To implement the model variants, the reference application uses variant subsystems.

Reference Application Element	Description	Variants
Analyze Power and Energy	Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see "Analyze Power and Energy" on page 3-129.	*NA*
Drive Cycle Source block — FTP75 (2474 seconds)	Generates a standard or user-specified drive cycle velocity versus time profile. Block output is the selected or specified vehicle longitudinal speed.	✓
`Environment` subsystem	Creates environment variables, including road grade, wind velocity, and atmospheric temperature and pressure.	
`Longitudinal Driver` subsystem	Uses the Longitudinal Driver or Open Loop variant to generate normalized acceleration and braking commands. • Longitudinal Driver variant implements a driver model that uses vehicle target and reference velocities. • Open Loop variant allows you to configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.	✓
`Controllers` subsystem	Implements a powertrain control module (PCM) containing a P4 hybrid control module (HCM), an engine control module (ECM), and a transmission control module (TCM).	✓

Reference Application Element	Description	Variants
`Passenger Car` subsystem	Implements a hybrid passenger car that contains drivetrain, electric plant, and engine subsystems. To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.	✓
`Visualization` subsystem	Displays vehicle-level performance, battery state of charge (SOC), fuel economy, and emission results that are useful for powertrain matching and component selection analysis.	

Evaluate and Report Power and Energy

Double-click **Analyze Power and Energy** to open a live script. Run the script to evaluate and report power and energy consumption at the component- and system-level. For more information about the live script, see “Analyze Power and Energy” on page 3-129.

The script provides:

- An overall energy summary that you can export to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain plant efficiencies, including an engine histogram of time spent at the different engine plant efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency and energy transfer signals.

Drive Cycle Source

The `Drive Cycle Source` block generates a target vehicle velocity for a selected or specified drive cycle. The reference application has these options.

Timing	Variant	Description
Output sample time	`Continuous` (default)	Continuous operator commands
	`Discrete`	Discrete operator commands

Longitudinal Driver

The `Longitudinal Driver` subsystem generates normalized acceleration and braking commands. The reference application has these variants.

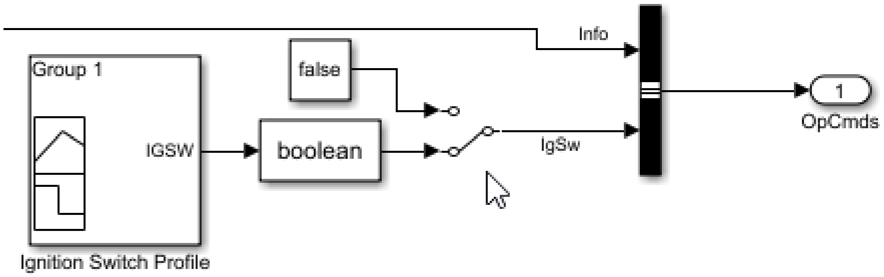
Block Variants			Description
Longitudinal Driver (default)	Control	`Mapped`	PI control with tracking windup and feed-forward gains that are a function of vehicle velocity.
		`Predictive`	Optimal single-point preview (look ahead) control.
		`Scalar`	Proportional-integral (PI) control with tracking windup and feed-forward gains.
	Low-pass filter (LPF)	`LPF`	Use an LPF on target velocity error for smoother driving.
		`pass`	Do not use a filter on velocity error.
	Shift	`Basic`	Stateflow chart models reverse, neutral, and drive gear shift scheduling.
		`External`	Input gear, vehicle state, and velocity feedback generates acceleration and braking commands to track forward and reverse vehicle motion.
		`None`	No transmission.
		`Scheduled`	Stateflow chart models reverse, neutral, park, and N-speed gear shift scheduling.
Open Loop			Open-loop control subsystem. In the subsystem, you can configure the acceleration, deceleration, gear, and clutch commands with constant or signal-based inputs.

To idle the engine at the beginning of a drive cycle and simulate catalyst light-off before moving the vehicle with a pedal command, use the Longitudinal Driver variant. The Longitudinal Driver subsystem includes an ignition switch signal profile, `IgSw`. The engine controller uses the ignition switch signal to start both the engine and a catalyst light-off timer.

The catalyst light-off timer overrides the engine stop-start (ESS) stop function control while the catalyst light-off timer is counting up. During the simulation, after the `IgSw` down-edge time reaches the catalyst light-off time `CatLightOffTime`, normal ESS operation resumes. If there is no torque command before the simulation reaches the `EngStopTime`, the ESS shuts down the engine.

To control ESS and catalyst light-off:

- In the Longitudinal Driver Model subsystem, set the ignition switch profile `IgSw` to `'on'`.



- In the engine controller model workspace, set these calibration parameters:
  - `EngStopStartEnable` — Enables ESS. To disable ESS, set the value to false.
  - `CatLightOffTime` — Engine idle time from engine start to catalyst light-off.
  - `EngStopTime` — ESS engine run time after driver model torque request cut-off.

Controllers

The `Controller` subsystem has a PCM containing an ECM, HCM, and TCM. The controller has these variants.

Controller	Variant	Description
ECM	`SiEngineController` (default)	Implements the SI Controller
	`CiEngineController`	Implements the CI Controller
HCM	`ECMS`	Implements the Equivalent Consumption Minimization Strategy
TCM	`TransmissionController`	Implements the transmission controller

Passenger Car

To implement a passenger car, the `Passenger Car` subsystem contains drivetrain, electric plant, and engine subsystems. To create your own engine variants for the reference application, use the CI and SI engine project templates. The reference application has these variants.

Drivetrain

To model the drivetrain, use the **Toggle To Simscape Drivetrain** button to switch between Simscape and Powertrain Blockset variants of the drivetrain subsystem. By default, the reference application uses the Powertrain Blockset variant. The Simscape variant incorporates physical connections to provide a flexible way to assemble components.

Tip The reference application sets the appropriate solvers to optimize performance for each engine and drivetrain combination. Select the engine variant first, then select the drivetrain using the toggle button. If you select the drivetrain before changing the engine, you may encounter a solver error.

Drivetrain Subsystem	Variant	Description
Differential and Compliance	`Limited Slip Differential`	You can vary the type of coupling torque and efficiency. By default, the differential is configured with an ideal wet clutch and constant efficiency.
	`Open Differential`	You can vary the type of differential efficiency. By default, the open differential is configured with a constant efficiency
Torque Converter Automatic Transmission	`Ideal Fixed Gear Transmission`	Configure locked and unlocked transmission efficiency with either a 1D or 4D (default) lookup table.
	`Torque Converter`	Configure for external, internal (default), or no lockup.
Vehicle	`Vehicle Body 1 DOF Longitudinal`	Configured for 1 degrees of freedom
Wheels and Brakes	`Longitudinal Wheel - Front 1`	For the wheels, you can configure the type of: • Brake • Force calculation • Resistance calculation • Vertical motion For performance and clarity, to determine the longitudinal force of each wheel, the variants implement the Longitudinal Wheel block. To determine the *total* longitudinal force of all wheels acting on the axle, the variants use a scale factor to multiply the force of one wheel by the number of wheels on the axle. By using this approach to calculate the total force, the variants assume equal tire slip and loading at the front and rear axles, which is common for longitudinal powertrain studies. If this is not the case, for example when friction or loads differ on the left and right sides of the axles, use unique Longitudinal Wheel blocks to calculate independent forces. However, using unique blocks to model each wheel increases model complexity and computational cost.
	`Longitudinal Wheel - Rear 1`	

Electric Plant

Electric Plant Subsystem	Variant	Description
Battery	`BattHevP4`	Configured with Lithium Ion battery
Electric Machine	`MotMapped`	Mapped Motor with implicit controller

Engine

Engine Subsystem	Variant	Description
Engine	`SiEngineCore`	Dynamic SI Core Engine with turbocharger
	`SiMappedEngine` (default)	Mapped SI Engine with implicit turbocharger
	`SiEngineCoreNA`	Dynamic naturally aspirated SI Core Engine

Limitations

MathWorks used the SI Core Engine and SI Controller to calibrate the hybrid control module (HCM). If you use the CI Core Engine and CI Controller variants, the simulation may error because the HCM does not use calibrated results.

Acknowledgment

MathWorks would like to acknowledge the contribution of Dr. Simona Onori to the ECMS optimal control algorithm implemented in this reference application. Dr. Onori is a Professor of Energy Resources Engineering at Stanford University. Her research interests include electrochemical modeling, estimation and optimization of energy storage devices for automotive and grid-level applications, hybrid and electric vehicles modeling and control, PDE modeling, and model-order reduction and estimation of emission mitigation systems. She is a senior member of IEEE.

References

[1] Balazs, A., Morra, E., and Pischinger, S., *Optimization of Electrified Powertrains for City Cars*. SAE Technical Paper 2011-01-2451. Warrendale, PA: SAE International Journal of Alternative Powertrains, 2012.

[2] Onori, S., Serrao, L., and Rizzoni, G., *Hybrid Electric Vehicles Energy Management Systems*. New York: Springer, 2016.

See Also

Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Core Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller | Mapped Motor

Related Examples

- “HEV P4 Reference Application” on page 7-9

More About

- “Analyze Power and Energy” on page 3-129
- “Hybrid and Electric Vehicle Reference Application Projects” on page 3-3
- “Variant Systems”

Develop, Resize, and Calibrate Motors with Dynamometer Test Harness

The motor dynamometer reference application represents a motor plant and controller connected to an AC dynamometer. Using the reference application, you can design and test traction e-motors and controllers for your electrified powertrain vehicle. To create and open a working copy of the motor dynamometer reference application project, use this command.

```
autoblkMotDynamometerStart
```

By default, the reference application is configured with an 80 kW Flux-Based PMSM motor.

You can configure the reference application project for different dynamometer control modes. To implement the operating modes, the reference application uses variant subsystems.

This table summarizes the dynamometer tests.

Test	Objective	Method	Motor Variant		
			Mapped	Dynamic PMSM	Flux-Based PMSM
Resize Mapped Motor Model	Match motor torque to desired maximum power, torque, DC link voltage, and constant power speed ratio (CPSR).	Dynamometer resizes the mapped motor and motor calibration parameters. For more details, see “Resize Motors” on page 3-92.	✓		
Resize Dynamic PMSM and Recalibrate Controller	Match motor torque to desired maximum power level and DC link voltage.	Dynamometer resizes the dynamic PMSM motor and motor calibration parameters. The dynamometer also recalibrates the controller. For more details, see “Resize Motors” on page 3-92.		✓	
Resize Flux-based PMSM and Recalibrate Controller	Match motor torque to desired maximum power level and DC link voltage.	Dynamometer resizes the flux-based PMSM motor and motor calibration parameters. The dynamometer also recalibrates the controller. For more details, see “Resize Motors” on page 3-92.			✓

Test	Objective	Method	Motor Variant		
			Mapped	Dynamic PMSM	Flux-Based PMSM
Run Performance Tests	Run dynamic and steady-state performance tests at different operating points of torque and speed.	Dynamometer controller commands a series of motor speeds and torques to the motor controller. At each quasi-steady-state operating point, the experiment records the motor plant model output and the controller commands for the current calibration parameters.	✓	✓	✓

Motor System

The reference application includes variant subsystems for Mapped (steady-state), Dynamic PMSM, and Flux-Based PMSM motors.

Objective	Motor Variant
Simulate and validate motor system response based on high-level specifications.	Mapped
Simulate and validate linear dynamic models based on existing motor details.	Dynamic PMSM
Simulate and validate nonlinear dynamic models based on existing motor details.	Flux-Based PMSM

Performance Monitor

The reference application contains a Performance Monitor block that you can use to plot steady-state and dynamic results. You can plot:

- Steady-state results as a function of one or two variables.
- Dynamic results using the Simulation Data Inspector.

See Also

Mapped Motor | Interior PM Controller | Interior PMSM | Flux-Based PM Controller | Flux-Based PMSM

Related Examples

- “Motor Dynamometer Reference Application” on page 7-17
- “Resize Motors” on page 3-92

More About

- “Variant Systems”

Resize Motors

By default, the motor dynamometer reference application motor is configured with an 80 kW Flux-Based PMSM motor. Based on desired specifications, you can resize the motor variant for different vehicle applications.

These motor variants are available.

Motor Subsystem	Variant	Description
Motor & Inverter Plant	Mapped	Simulate and validate motor system response based on high-level specifications.
	Dynamic PMSM	Simulate and validate linear dynamic models based on existing motor details.
	Flux-Based PMSM (default)	Simulate and validate nonlinear dynamic models based on existing motor details.

You can use the **Virtual Vehicle Composer** app to resize your virtual vehicle motor based on high-level specifications. For more information, see “Resize Engines and Mapped Motors” on page 8-18.

To set the variant and resize the motor, use the dynamometer reference application. After you open the reference application, click one of the Resize buttons:

- For a mapped motor, click **Resize Mapped Motor Model**. Then, set the desired maximum power, torque, DC link voltage and constant power speed ratio (CPSR).
- For a dynamic motor, click **Resize Dynamic PMSM and Recalibrate Controller**. Then, set the desired maximum power level, DC link voltage, and axial resize factor.
- For a flux-based motor, click **Resize Flux-based PMSM and Recalibrate Controller**. Then, set the desired maximum power level, DC link voltage, and axial resize factor.

Click **Apply** and **Resize Motor** to resize the motor with your desired specifications. The reference application:

- Resizes the motor and motor calibration parameters
- Recalibrates the controller to match the resized motor
- Saves the motor and controller parameters to the model

Click **Plot Characteristics** to view characteristics of the new motor.

You can use the variants in other applications, for example, in vehicle projects that require a motor model.

See Also

Mapped Motor | Interior PM Controller | Interior PMSM | Flux-Based PM Controller | Flux-Based PMSM

More About

- “Develop, Resize, and Calibrate Motors with Dynamometer Test Harness” on page 3-89

- “Resize Engines and Mapped Motors” on page 8-18

Resize the CI Engine

By default, the compression-ignition (CI) engine dynamometer reference application engine is configured with a dynamic 1.5-L turbocharged diesel engine. Based on a desired number of cylinders and maximum engine power or engine displacement, you can resize the dynamic engine (`CiEngine`) for different vehicle applications.

You can use the **Virtual Vehicle Composer** app to resize your virtual vehicle engine based on desired maximum engine power or engine displacement. For more information, see “Resize Engines and Mapped Motors” on page 8-18.

To resize the engine, use the dynamometer reference application. After you open the reference application, click **Resize Engine and Recalibrate Controller**. In the dialog box, set **Power or displacement** to either:

- `Power` – Enter a **Desired maximum power** value
- `Displacement` – Enter a **Desired displacement** value

For either power or displacement, enter a **Desired number of cylinders** value.

After you apply the changes, the reference application:

- Resizes the dynamic engine and engine calibration parameters. The **Resize Engine and Recalibrate Controller** block mask provides the updated engine performance characteristics based on the resized calibration parameters.
- Recalibrates the controller and mapped engine model to match the resized dynamic engine.

You can use the variants in other applications, for example, in vehicle projects that require a larger engine model.

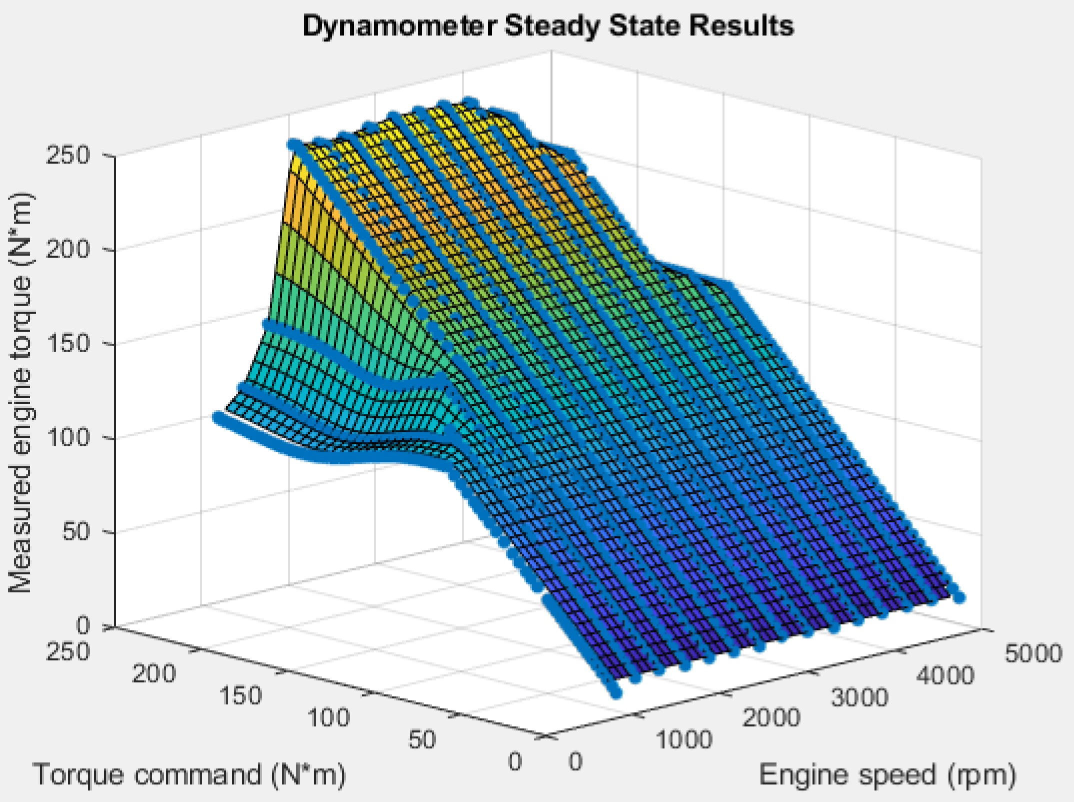
Create CI Engine Models with Twice the Power

1 If it is not already open, open a copy of the CI engine reference application project by entering

```
autoblkCIDynamometerStart
```

2 In the `CiDynoReferenceApplication` model window, click **Recalibrate Controller**.

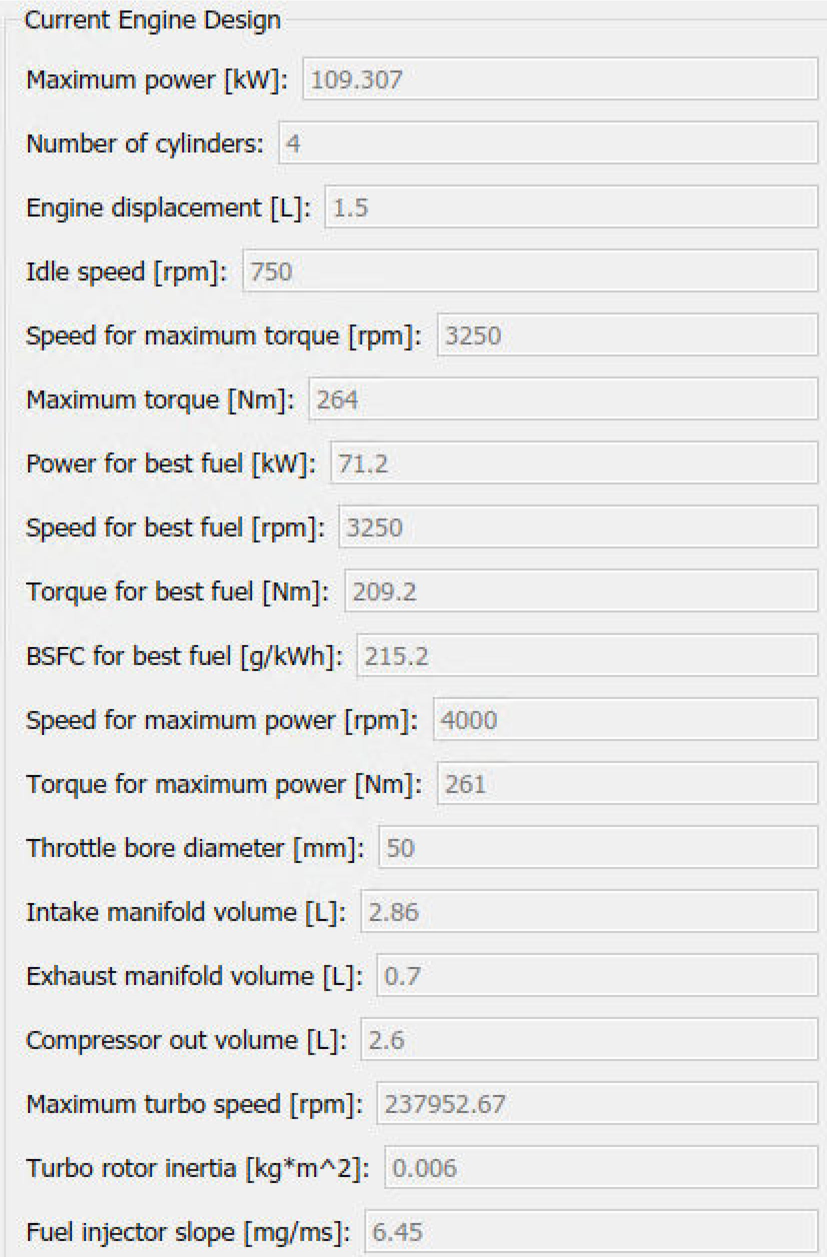
The reference application performs a dynamometer test to calibrate the engine controller for the default 1.5-L turbocharged diesel engine. For engine speeds 2000–5000 rpm, the measured engine torque approaches 240 N·m. The steady-state results for measured engine torque as a function of torque command and engine speed are similar to this plot.



3 In the `CiDynoReferenceApplication` model window, click **Resize Engine and Recalibrate Controller**.

The dialog box opens with default values for **Desired maximum power** and **Desired number of cylinders**. These values represent the calibration parameters for the default 1.5-L dynamic engine.

The dialog box provides the calibration parameters for the current engine design. The parameters are similar to these.



Current Engine Design

Maximum power [kW]: 109.307

Number of cylinders: 4

Engine displacement [L]: 1.5

Idle speed [rpm]: 750

Speed for maximum torque [rpm]: 3250

Maximum torque [Nm]: 264

Power for best fuel [kW]: 71.2

Speed for best fuel [rpm]: 3250

Torque for best fuel [Nm]: 209.2

BSFC for best fuel [g/kWh]: 215.2

Speed for maximum power [rpm]: 4000

Torque for maximum power [Nm]: 261

Throttle bore diameter [mm]: 50

Intake manifold volume [L]: 2.86

Exhaust manifold volume [L]: 0.7

Compressor out volume [L]: 2.6

Maximum turbo speed [rpm]: 237952.67

Turbo rotor inertia [kg*m^2]: 0.006

Fuel injector slope [mg/ms]: 6.45

4 In the **Resize Engine and Recalibrate Controller** dialog box, enter values that represent approximately twice the maximum power and number of cylinders. For example, set:

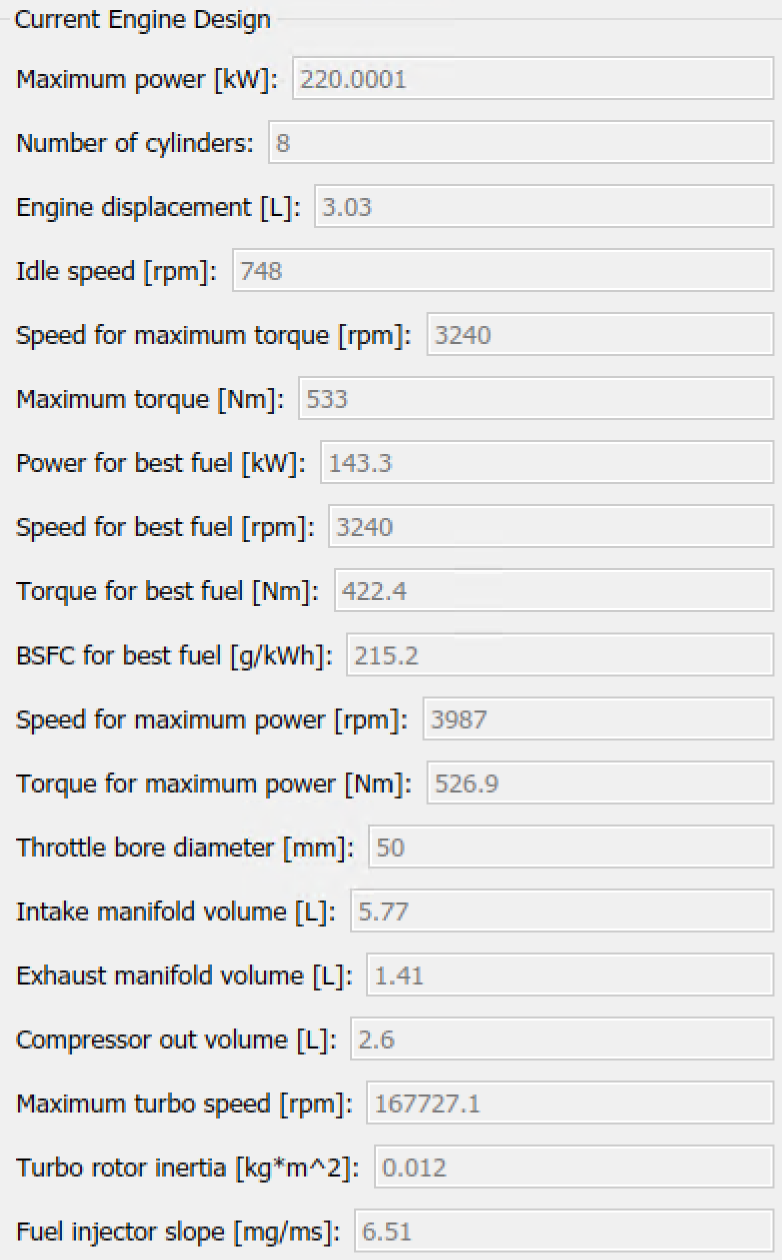
- **Desired maximum power** to `220`.
- **Desired number of cylinders** to `8`.

Click **Resize Engine**. The reference application:

- Resizes the dynamic engine (`CiEngineCore`) and engine calibration parameters. The dialog box provides the updated engine performance characteristics based on the resized calibration parameters.

- Recalibrates the controller (`CiEngineController`) and mapped engine model (`CiMappedEngine`) to match the resized dynamic engine (`CiEngineCore`).

After resizing and recalibration, the dialog box provides the calibration parameters for the resized engine. The parameters are similar to these.



Current Engine Design

Maximum power [kW]: 220.0001

Number of cylinders: 8

Engine displacement [L]: 3.03

Idle speed [rpm]: 748

Speed for maximum torque [rpm]: 3240

Maximum torque [Nm]: 533

Power for best fuel [kW]: 143.3

Speed for best fuel [rpm]: 3240

Torque for best fuel [Nm]: 422.4

BSFC for best fuel [g/kWh]: 215.2

Speed for maximum power [rpm]: 3987

Torque for maximum power [Nm]: 526.9

Throttle bore diameter [mm]: 50

Intake manifold volume [L]: 5.77

Exhaust manifold volume [L]: 1.41

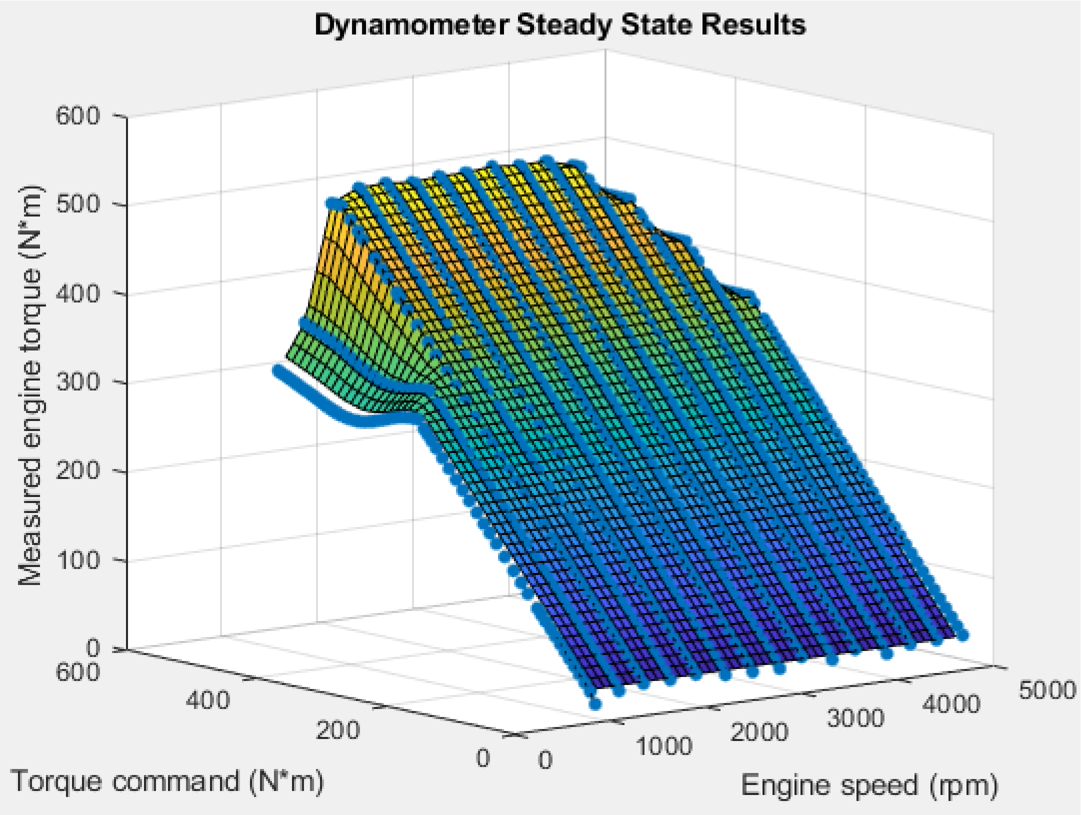
Compressor out volume [L]: 2.6

Maximum turbo speed [rpm]: 167727.1

Turbo rotor inertia [kg*m^2]: 0.012

Fuel injector slope [mg/ms]: 6.51

5 Examine the dynamometer steady-state results. For engine speeds 2000–5000 rpm, the measured engine torque approaches 500 N·m. This result is approximately twice the power of the default dynamic engine. The steady-state results for measured engine torque as a function of torque command and engine speed are similar to this plot.



6 To save the engine controller, resized engine mapped variant, and resized dynamic engine variant, in the `CiDynoReferenceApplication` model window, save the reference application.

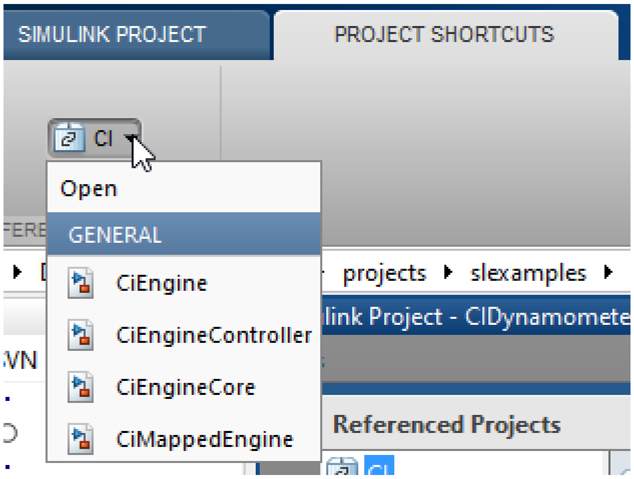
By default, this process creates:

- An updated CI engine controller
- Two engine variants — mapped and dynamic

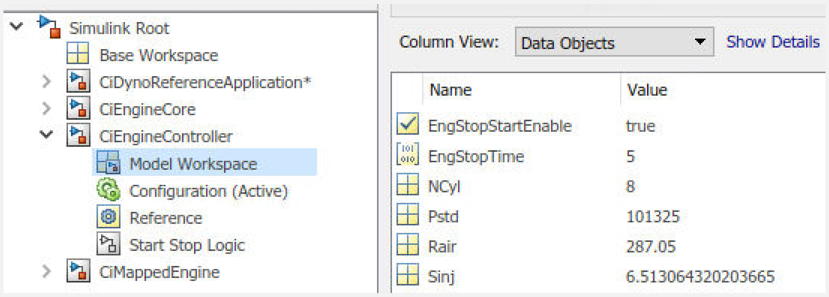
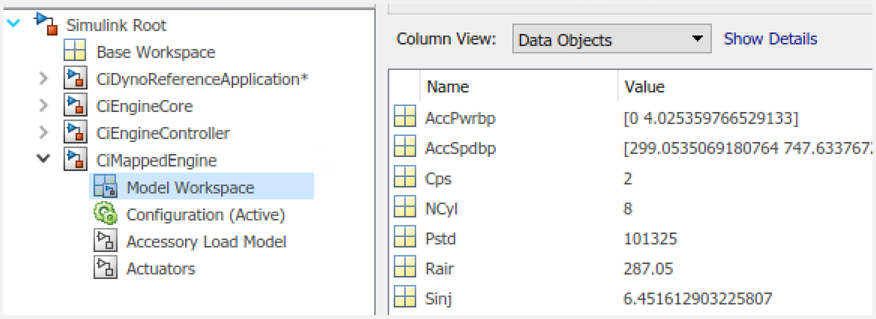
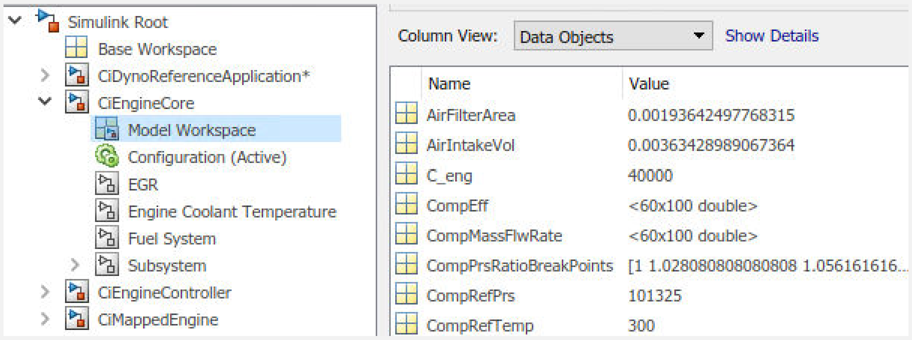
To see the parameters associated with the controller and engine variants:

1 In MATLAB, use the **Project Shortcuts** tab to open these models:

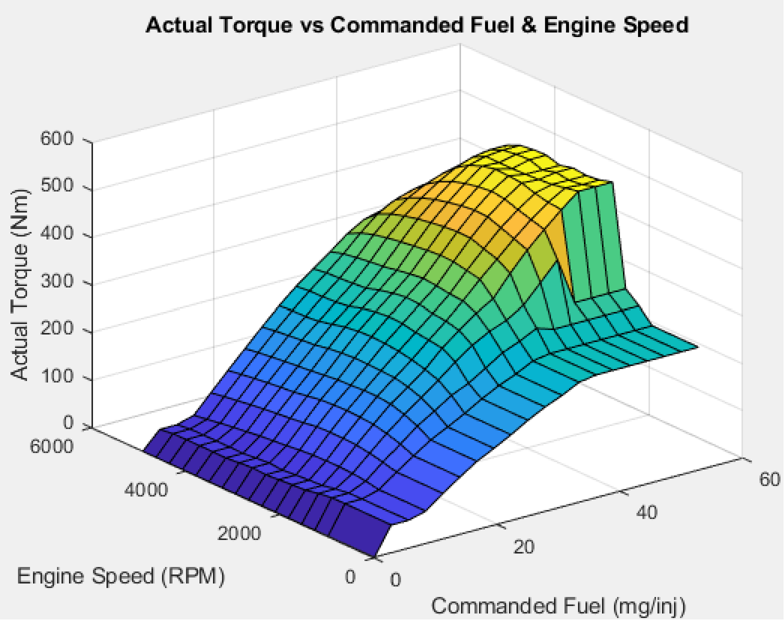
   - `CiEngineController`
   - `CiEngineCore`
   - `CiMappedEngine`



2 Use the Model Explorer to view the resized parameters:

Engine Model	Model Explorer
Controller — `CiEngineController`	 Simulink Root Base Workspace CiDynoReferenceApplication* CiEngineCore CiEngineController Model Workspace Configuration (Active) Reference Start Stop Logic CiMappedEngine Column View: Show Details Column View: Data Objects Name — Value ☑ EngStopStartEnable — true EngStopTime — 5 NCyl — 8 Pstd — 101325 Rair — 287.05 Sinj — 6.513064320203665
Mapped — `CiMappedEngine`	 Simulink Root Base Workspace CiDynoReferenceApplication* CiEngineCore CiEngineController CiMappedEngine Model Workspace Configuration (Active) Accessory Load Model Actuators Column View: Show Details Column View: Data Objects Name — Value AccPwrbp — [0 4.025359766529133] AccSpdbp — [299.0535069180764 747.633767 Cps — 2 NCyl — 8 Pstd — 101325 Rair — 287.05 Sinj — 6.451612903225807
Dynamic — `CiEngineCore`	 Simulink Root Base Workspace CiDynoReferenceApplication* CiEngineCore Model Workspace Configuration (Active) EGR Engine Coolant Temperature Fuel System Subsystem CiEngineController CiMappedEngine Column View: Show Details Column View: Data Objects Name — Value AirFilterArea — 0.00193642497768315 AirIntakeVol — 0.00363428989067364 C_eng — 40000 CompEff — <60x100 double> CompMassFlwRate — <60x100 double> CompPrsRatioBreakPoints — [1 1.028080808080808 1.056161616... CompRefPrs — 101325 CompRefTemp — 300

3 In the `CiDynoReferencApplication` > `Engine System` > `Engine Plant` > `Engine` > `CIMappedEngine` subsystem, open the Mapped CI Engine block. On the Power tab, plot the actual torque as a function of engine speed and commanded fuel.



See Also

CI Core Engine | Mapped CI Engine | **Virtual Vehicle Composer**

More About

- “Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness” on page 3-11

Resize the SI Engine

By default, the spark-ignition (SI) engine dynamometer reference application engine is configured with a turbocharged 1.5-L dynamic gasoline engine. Based on a desired number of cylinders and maximum engine power or engine displacement, you can resize the dynamic engine variant for different vehicle applications.

You can use the **Virtual Vehicle Composer** app to resize your virtual vehicle engine based on desired maximum engine power or engine displacement. For more information, see “Resize Engines and Mapped Motors” on page 8-18.

To resize the engine, use the dynamometer reference application. After you open the reference application, click **Resize Engine and Recalibrate Controller**. In the dialog box, set **Resize option** to either:

- `Power` – Enter a **Desired maximum power** value.
- `Displacement` – Enter a **Desired displacement** value.

For either power or displacement, enter a **Desired number of cylinders** value.

When in `Displacement` mode, you can define the maximum torque and the engine speed at which maximum torque occurs. Click the checkboxes to enable these entry fields.

You can choose the architecture, air path configuration (turbocharged or naturally aspirated), and presence or absence of cooled exhaust gas re-circulation (EGR) of your engine model. After making your selections, click **Resize Engine** to set the engine variant.

The available engine variants are:

Engine Subsystem	Variant	Description
Engine	`SiEngineCore` (default)	Dynamic Inline Turbo SI Core Engine
	`SiEngineCoreNA`	Dynamic SI Inline Naturally Aspirated Engine
	`SiEngineCoreV`	Dynamic SI V Twin-Turbo Single-Intake Engine
	`SiEngineCoreVNA`	Dynamic SI V Naturally Aspirated Engine
	`SiEngineCoreVThr2`	Dynamic SI V Twin-Turbo Twin-Intake Engine

After you apply the changes, the reference application:

- Resizes the dynamic engine and engine calibration parameters. The **Resize Engine and Recalibrate Controller** block mask provides the updated engine performance characteristics based on the resized calibration parameters.
- Recalibrates the controller and mapped engine model to match the resized dynamic engine.

You can use the variants in other applications, for example, in vehicle projects that require a larger engine model.

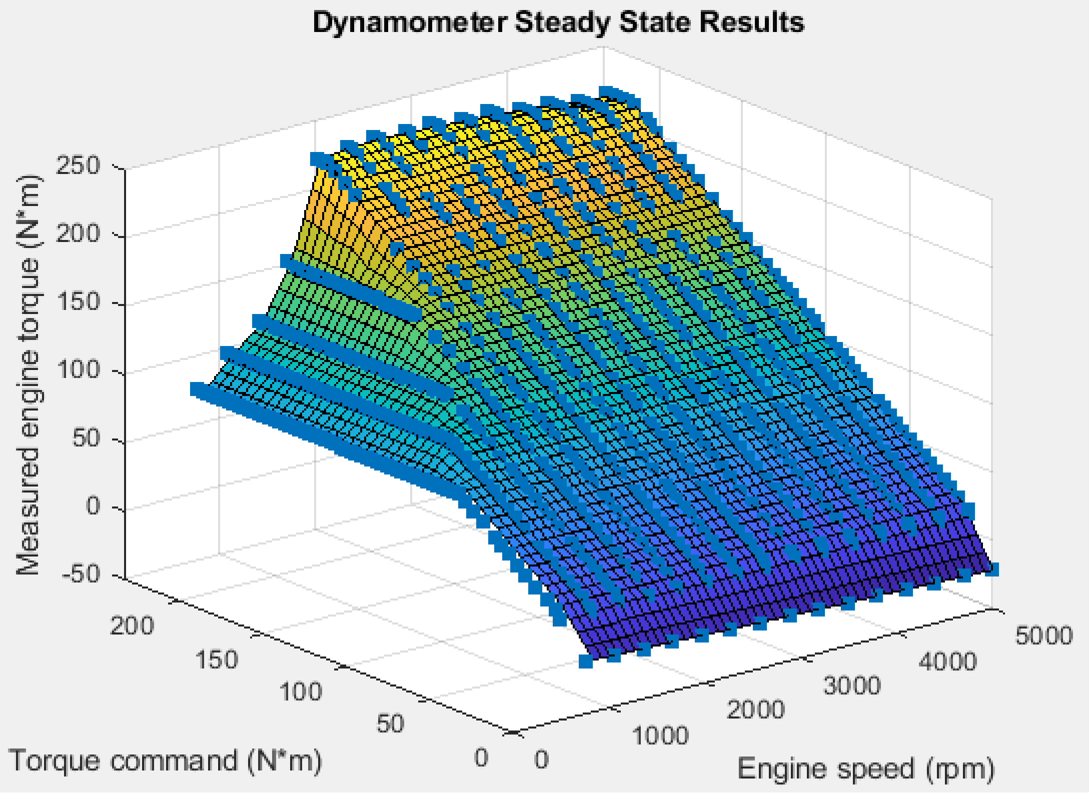
Create SI Engine Models with Twice the Power

1. If it is not already open, open a copy of the SI engine reference application project by entering

```
autoblkSIDynamometerStart
```

2 In the `SiDynoReferenceApplication` model window, click **Recalibrate Controller**.

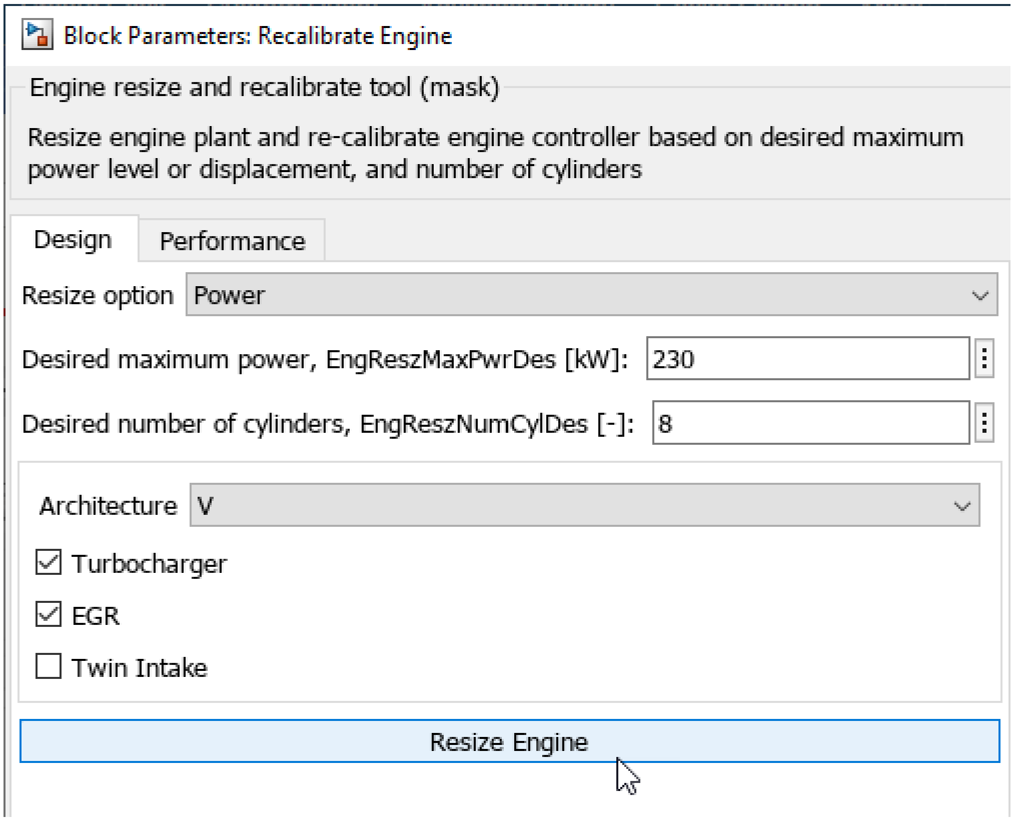
The reference application performs a dynamometer test to calibrate the engine controller for the default 1.5-L dynamic engine. For engine speeds 2000–5000 rpm, the measured engine torque approaches 220 N·m. The steady-state results for measured engine torque as a function of torque command and engine speed are similar to this plot.



3 In the `SiDynoReferenceApplication` model window, click **Resize Engine and Recalibrate Controller**. The dialog box provides the calibration parameters for the current engine design, the default 1.5-L dynamic engine.

4 In the **Resize Engine and Recalibrate Controller** dialog box, on the **Design** tab, set the resize options and engine architecture that represent twice the maximum power and number of cylinders. For example, set:

- **Resize option** to `Power`.
- **Desired maximum power** to `230`.
- **Desired number of cylinders** to `8`.
- **Architecture** to `V`.



Block Parameters: Recalibrate Engine

Engine resize and recalibrate tool (mask)

Resize engine plant and re-calibrate engine controller based on desired maximum power level or displacement, and number of cylinders

Resize option: Power

Desired maximum power, EngReszMaxPwrDes [kW]: 230

Desired number of cylinders, EngReszNumCylDes [-]: 8

Architecture: V

☑ Turbocharger

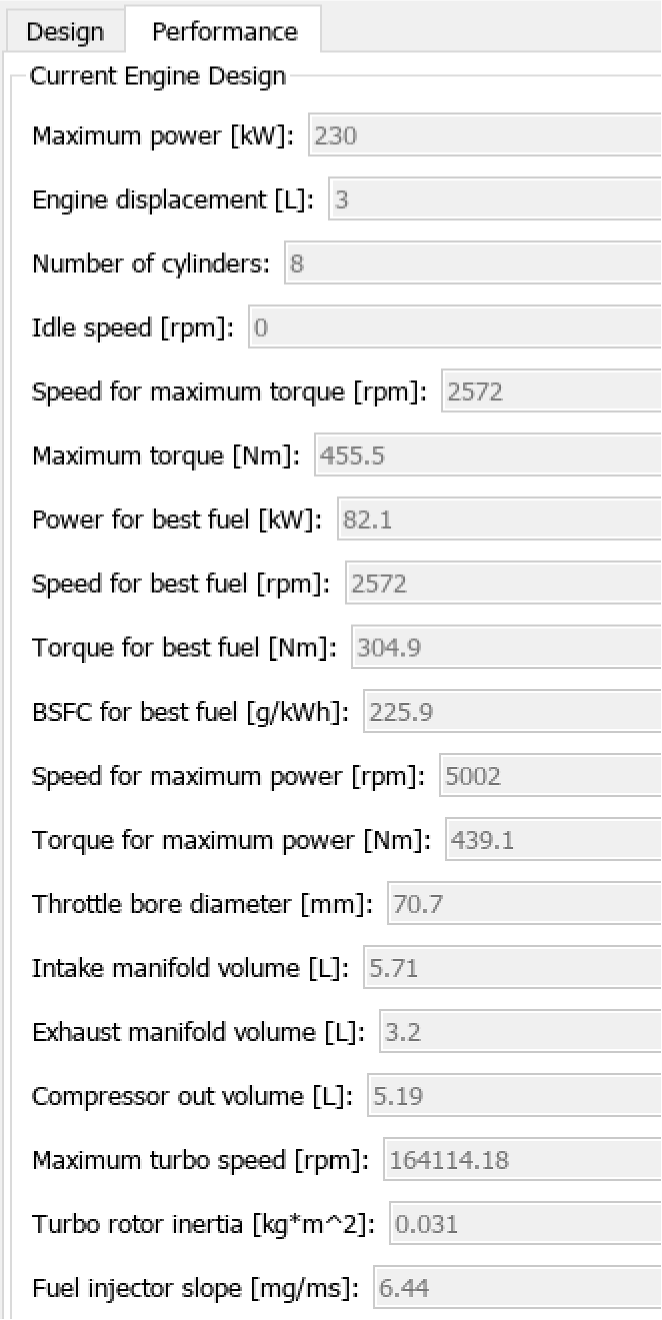
☑ EGR

□ Twin Intake

5 Click **Resize Engine**. The reference application:

- Resizes the dynamic engine (`SiEngineCore`) and engine calibration parameters. The **Recalibrate Engine** dialog box provides the updated engine performance characteristics based on the resized calibration parameters.
- Recalibrates the controller (`SiEngineController`) and mapped engine model (`SiMappedEngine`) to match the resized dynamic engine (`SiEngineCore`).

After resizing and recalibration, the dialog box provides the calibration parameters for the resized engine. The parameters are similar to these.



Current Engine Design

Maximum power [kW]: 230

Engine displacement [L]: 3

Number of cylinders: 8

Idle speed [rpm]: 0

Speed for maximum torque [rpm]: 2572

Maximum torque [Nm]: 455.5

Power for best fuel [kW]: 82.1

Speed for best fuel [rpm]: 2572

Torque for best fuel [Nm]: 304.9

BSFC for best fuel [g/kWh]: 225.9

Speed for maximum power [rpm]: 5002

Torque for maximum power [Nm]: 439.1

Throttle bore diameter [mm]: 70.7

Intake manifold volume [L]: 5.71

Exhaust manifold volume [L]: 3.2

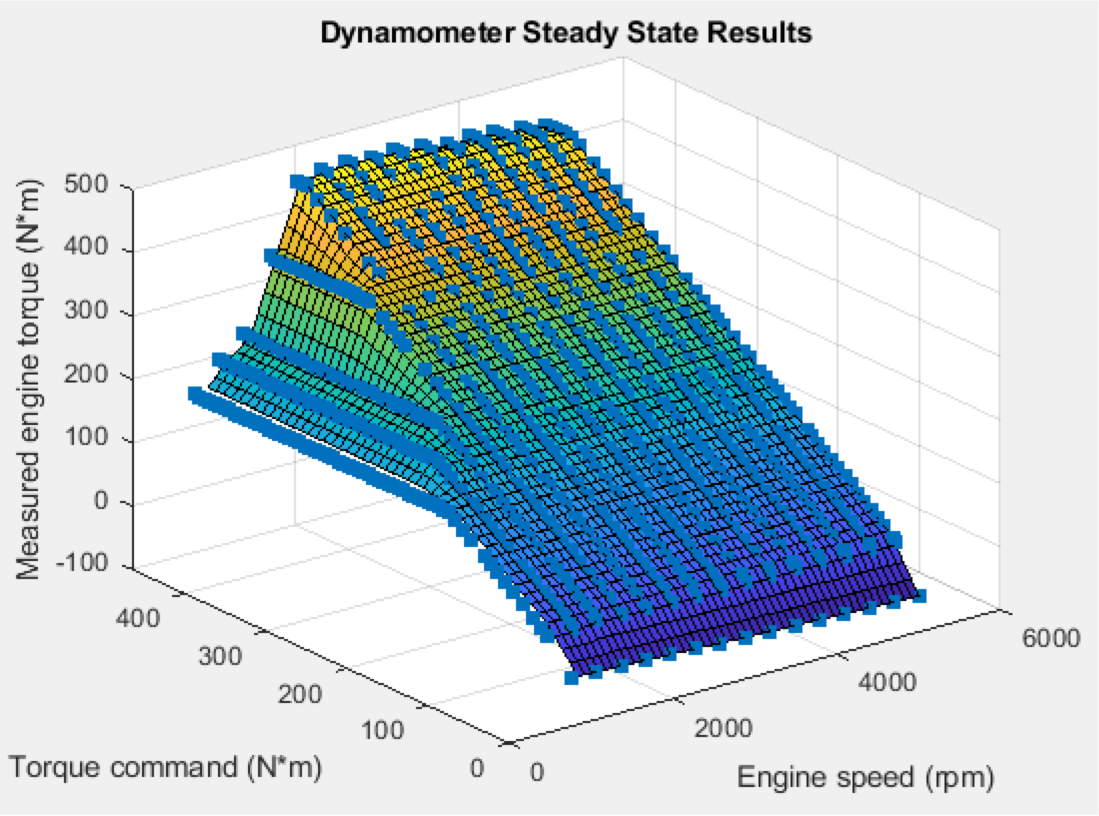
Compressor out volume [L]: 5.19

Maximum turbo speed [rpm]: 164114.18

Turbo rotor inertia [kg*m^2]: 0.031

Fuel injector slope [mg/ms]: 6.44

6 Examine the dynamometer steady-state results. For engine speeds 2000–5000 rpm, the measured engine torque approaches 450 N·m. This result is approximately twice the power of the default dynamic engine. The steady-state results for measured engine torque as a function of torque command and engine speed are similar to this plot.



7 To save the engine controller, resized engine mapped variant, and resized dynamic engine variant, in the `SiDynoReferenceApplication` model window, save the reference application.

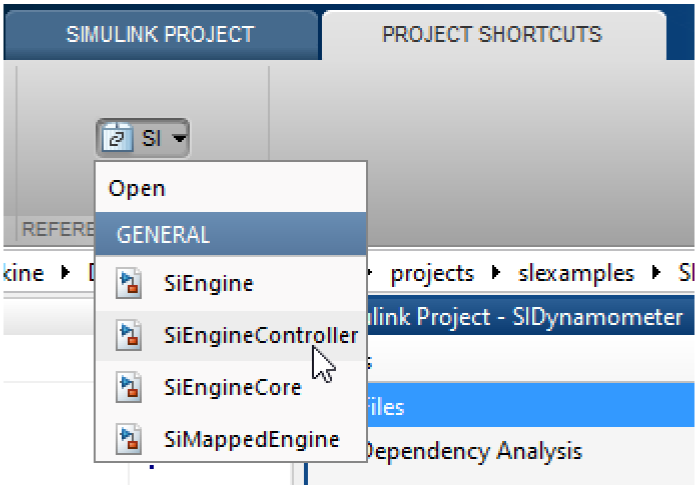
By default, this process creates:

- An updated SI engine controller
- Two engine variants — mapped and dynamic

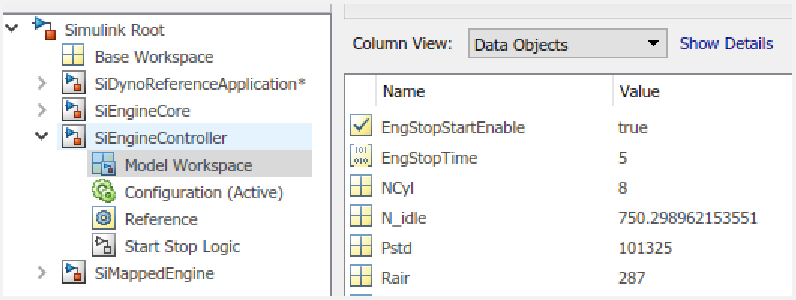
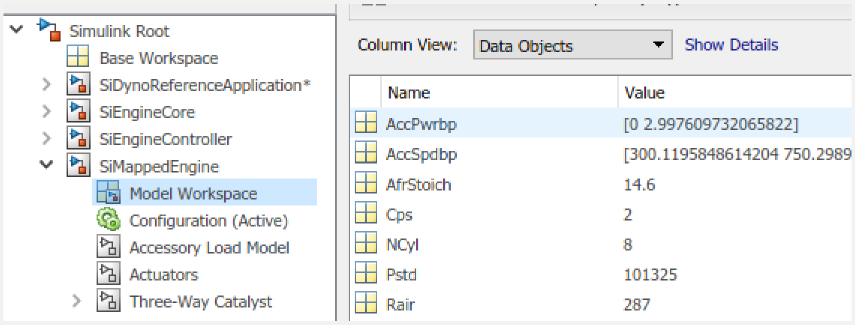
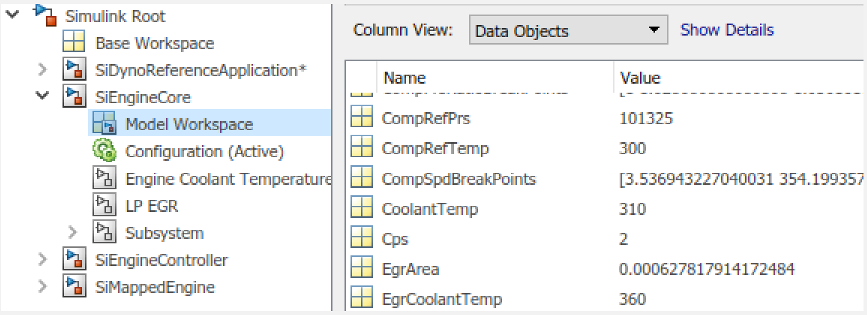
To see the parameters associated with the controller and engine variants:

1 In MATLAB, use the **Project Shortcuts** tab to open these models:

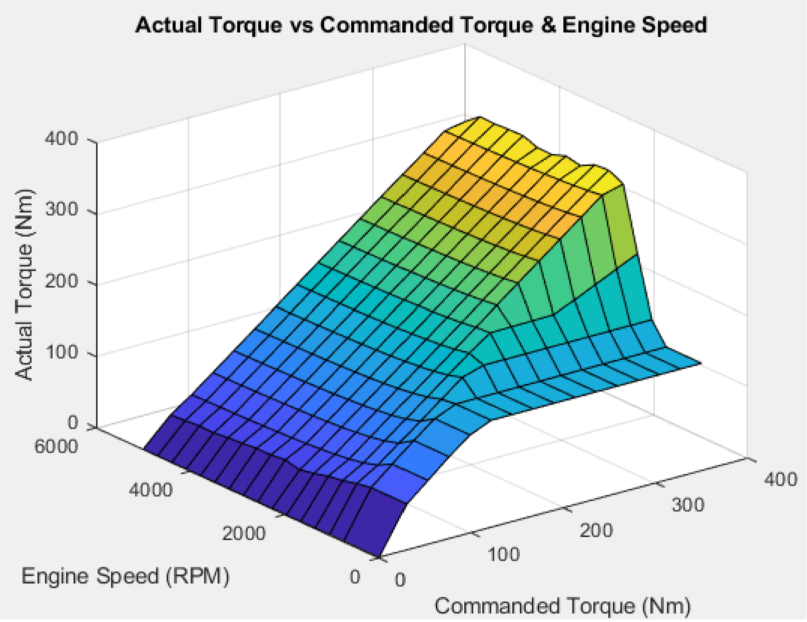
   - `SiEngineController`
   - `SiEngineCore`
   - `SiMappedEngine`



2 Use the Model Explorer to view the resized parameters:

Engine Model	Model Explorer
Controller — `SiEngineController`	 Simulink Root Base Workspace SiDynoReferenceApplication* SiEngineCore SiEngineController Model Workspace Configuration (Active) Reference Start Stop Logic SiMappedEngine Column View: Data Objects Show Details Name: Value EngStopStartEnable: true EngStopTime: 5 NCyl: 8 N_idle: 750.298962153551 Pstd: 101325 Rair: 287
Mapped — `SiMappedEngine`	 Simulink Root Base Workspace SiDynoReferenceApplication* SiEngineCore SiEngineController SiMappedEngine Model Workspace Configuration (Active) Accessory Load Model Actuators Three-Way Catalyst Column View: Data Objects Show Details Name: Value AccPwrbp: [0 2.997609732065822] AccSpdbp: [300.1195848614204 750.2989 AfrStoich: 14.6 Cps: 2 NCyl: 8 Pstd: 101325 Rair: 287
Dynamic — `SiEngineCore`	 Simulink Root Base Workspace SiDynoReferenceApplication* SiEngineCore Model Workspace Configuration (Active) Engine Coolant Temperature LP EGR Subsystem SiEngineController SiMappedEngine Column View: Data Objects Show Details Name: Value CompRefPrs: 101325 CompRefTemp: 300 CompSpdBreakPoints: [3.536943227040031 354.199357 CoolantTemp: 310 Cps: 2 EgrArea: 0.000627817914172484 EgrCoolantTemp: 360

3. In the `SiDynoReferencApplication` > `Engine System` > `Engine Plant` > `Engine` > `SIMappedEngine` subsystem, open the Mapped SI Engine block. On the Power tab, plot the actual torque as a function of engine speed and commanded torque.



See Also

SI Core Engine | Mapped SI Engine | **Virtual Vehicle Composer**

More About

- “Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15

Generate Mapped CI Engine from a Spreadsheet

If you have Model-Based Calibration Toolbox and Stateflow, you can use the engine dynamometer reference application to generate lookup tables for the Mapped CI Engine block. The reference application uses engine data to calibrate the engine and generate the tables.

1 If it is not opened, open the reference application.

```
autoblkCIDynamometerStart
```

2 Click **Generate Mapped Engine from Spreadsheet**.

Step 1: Generate Mapped Engine Calibration

1 Use the **Spreadsheet file** field to provide a data file. By default, the reference application has `CiEngineData.xlsx` containing 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.

Note To specify the lookup table type, in the Mapped CI Engine block, set the **Input command** parameter to `Fuel mass` or `Torque`.

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*

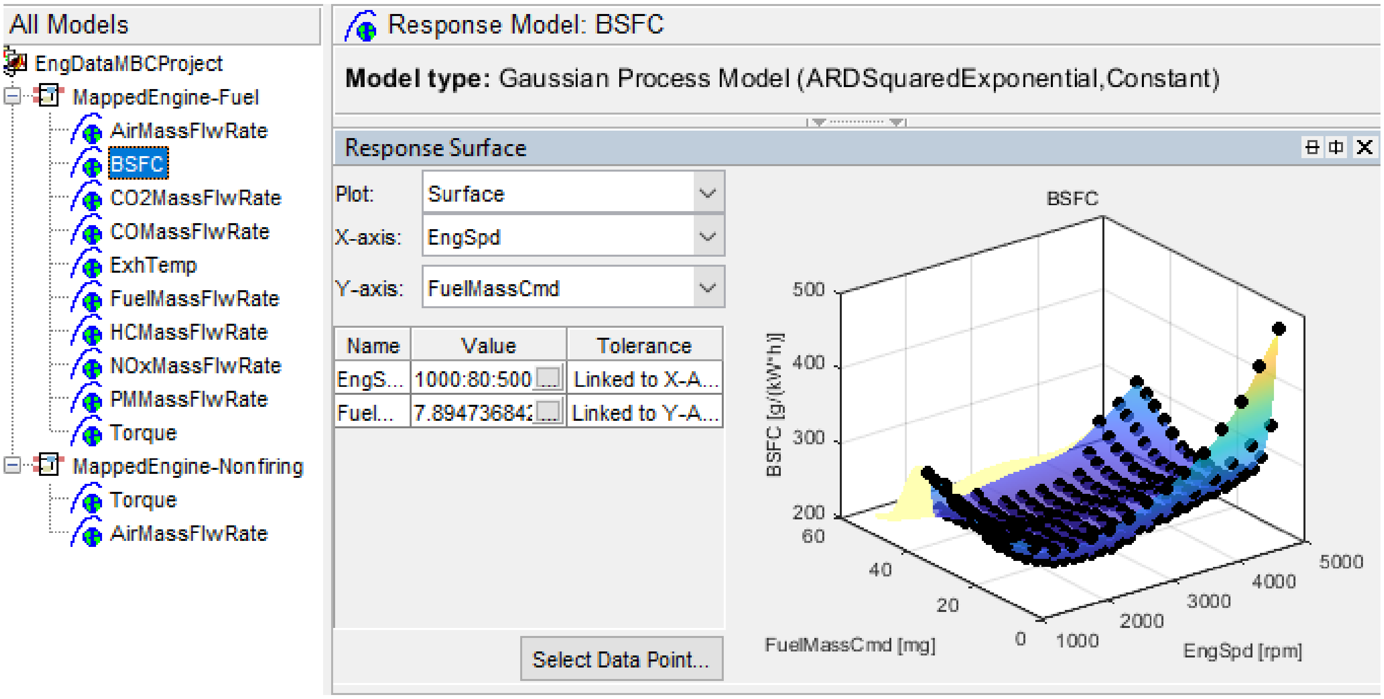
Firing Data	Description	Data Requirements for Generating Mapped Engine Tables	
		Function of Fuel Mass and Engine Speed	Function of Torque and Engine Speed
COMassFlwRate	Carbon monoxide emission mass flow, in kg/s	*Optional*	*Optional*
NOxMassFlwRate	Nitric oxide and nitrogen dioxide emissions mass flow, 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*

2 Click **Generate mapped engine calibration** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). CAGE and the model browser open when the process completes. To calibrate the data, Model-Based Calibration Toolbox uses templates.

- The Model Browser provides the response model fits for the data contained in the data file, for example:



All Models

- EngDataMBCProject
  - MappedEngine-Fuel
    - AirMassFlwRate
    - BSFC
    - CO2MassFlwRate
    - COMassFlwRate
    - ExhTemp
    - FuelMassFlwRate
    - HCMassFlwRate
    - NOxMassFlwRate
    - PMMassFlwRate
    - Torque
  - MappedEngine-Nonfiring
    - Torque
    - AirMassFlwRate

Response Model: BSFC

Model type: Gaussian Process Model (ARDSquaredExponential,Constant)

Response Surface

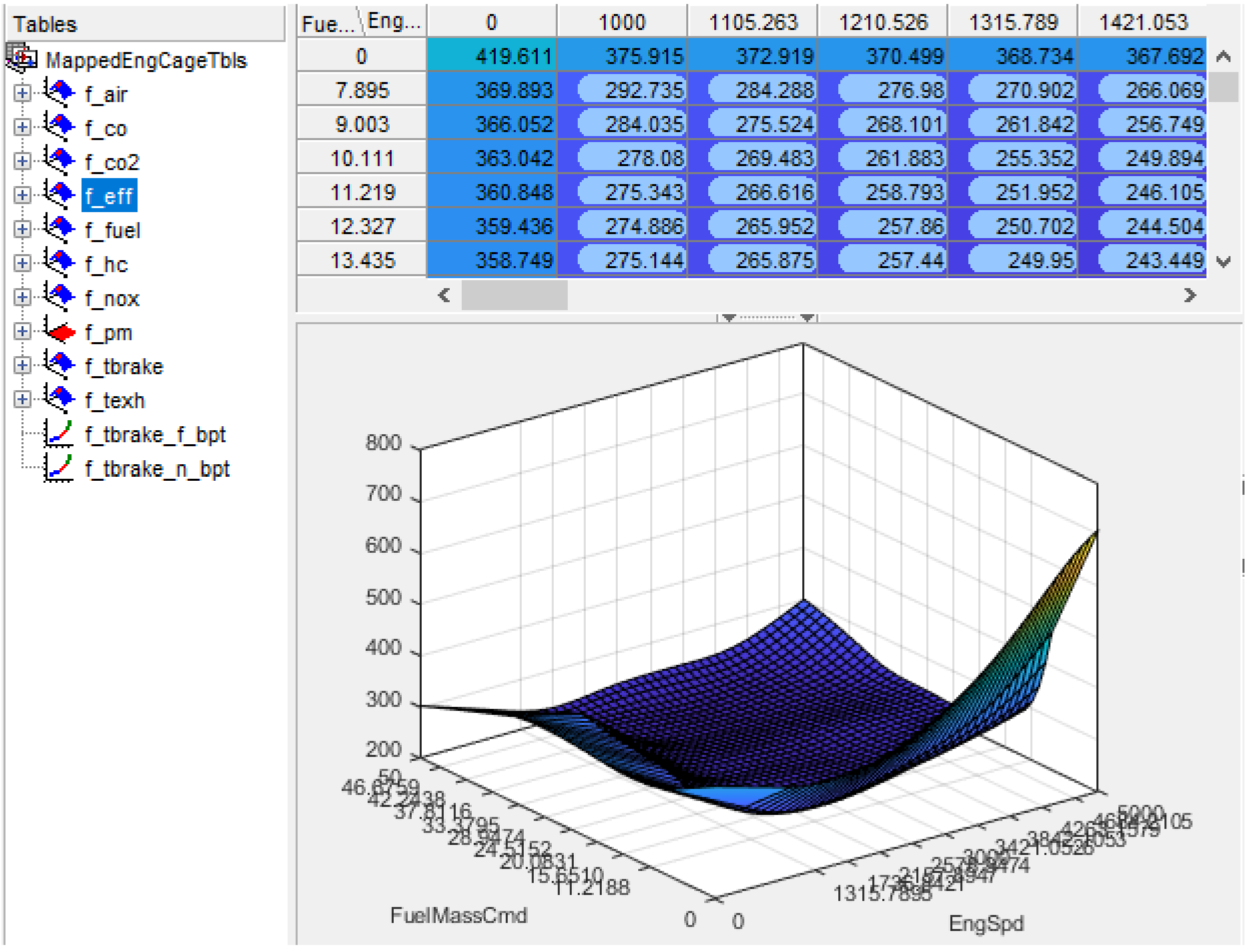
Plot: Surface

X-axis: EngSpd

Y-axis: FuelMassCmd

Name	Value	Tolerance
EngS...	1000:80:500	Linked to X-A...
Fuel...	7.89473684	Linked to Y-A...

- The CAGE Browser provides the calibrated data, for example:



Tables

- MappedEngCageTbls
  - f_air
  - f_co
  - f_co2
  - f_eff
  - f_fuel
  - f_hc
  - f_nox
  - f_pm
  - f_tbrake
  - f_texh
  - f_tbrake_f_bpt
  - f_tbrake_n_bpt

Fue...\Eng...	0	1000	1105.263	1210.526	1315.789	1421.053
0	419.611	375.915	372.919	370.499	368.734	367.692
7.895	369.893	292.735	284.288	276.98	270.902	266.069
9.003	366.052	284.035	275.524	268.101	261.842	256.749
10.111	363.042	278.08	269.483	261.883	255.352	249.894
11.219	360.848	275.343	266.616	258.793	251.952	246.105
12.327	359.436	274.886	265.952	257.86	250.702	244.504
13.435	358.749	275.144	265.875	257.44	249.95	243.449

Step 2: Apply Calibration to Mapped Engine Model

When you click **Apply calibration to mapped engine model**, Powertrain Blockset:

- Updates the Mapped CI Engine block parameters with the calibrated table and breakpoint data.
- Updates the CI Controller with the fuel mass per injection table if the Mapped CI Engine block tables are functions of fuel mass and engine speed.
- Sets the Mapped CI Engine as the active variant.
- Executes the engine mapping experiment.

When the dynamometer engine mapping completes, use the Simulation Data Inspector to verify the results.

See Also

Mapped CI Engine | CI Core Engine | CI Controller

More About

- “Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness” on page 3-11

- “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 SI Engine from a Spreadsheet

If you have Model-Based Calibration Toolbox and Stateflow, you can use the engine dynamometer reference application to generate lookup tables for the Mapped SI Engine block. The reference application uses engine data to calibrate the engine and generate the tables.

1. If it is not opened, open the reference application.

   ```
   autoblkSIDynamometerStart
   ```
2. Click **Generate Mapped Engine from Spreadsheet**.

Step 1: Generate Mapped Engine Calibration

1. Use the **Spreadsheet file** field to provide a data file. By default, the reference application has `SiEngineData.xlsx` containing 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.

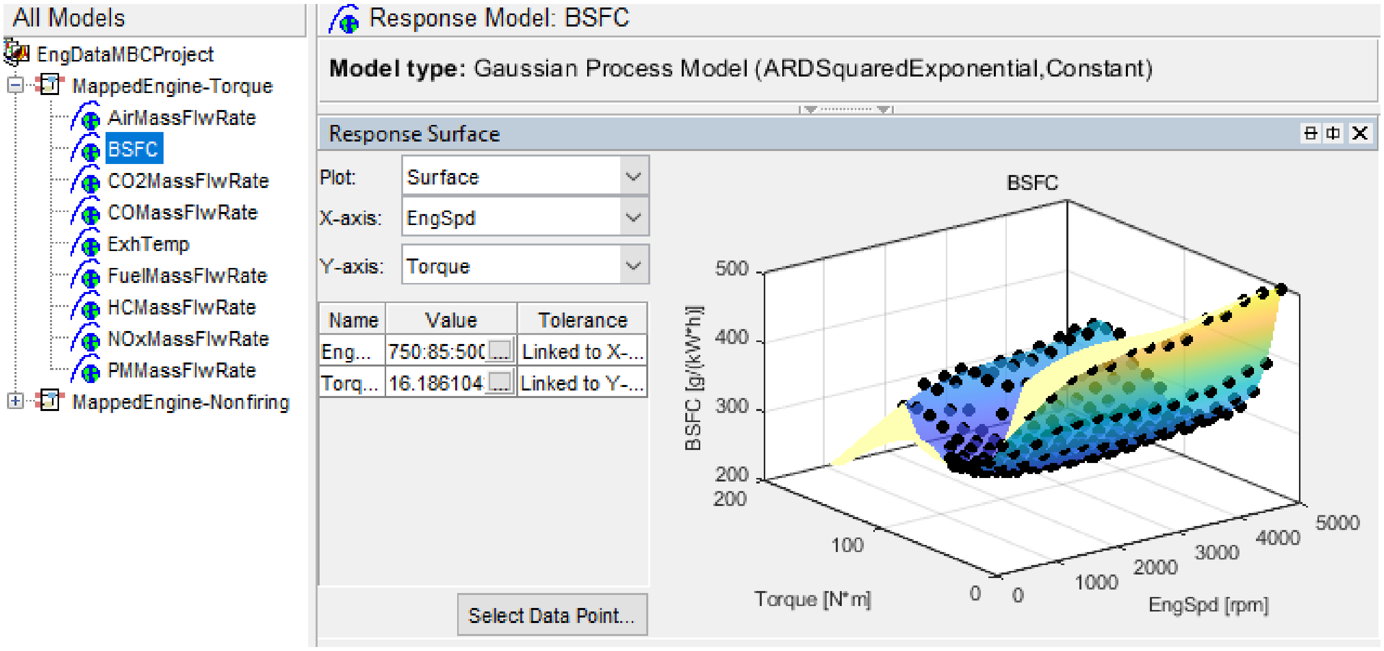
Firing Data	Description	Data Requirements for Generating Mapped Engine Tables
FuelMassCmd	Injected fuel mass, in mg per injection	*Not Used*
Torque	Engine torque command, in N·m	*Required*
EngSpd	Engine speed, in rpm	*Required*
AirMassFlwRate	Air mass flow, in kg/s	*Optional*
FuelMassFlwRate	Fuel mass 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*
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*

2 Click **Generate mapped engine calibration** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). CAGE and the model browser open when the process completes. To calibrate the data, Model-Based Calibration Toolbox uses templates.

- The Model Browser provides the response model fits for the data contained in the data file, for example:



Model type: Gaussian Process Model (ARDSquaredExponential,Constant)

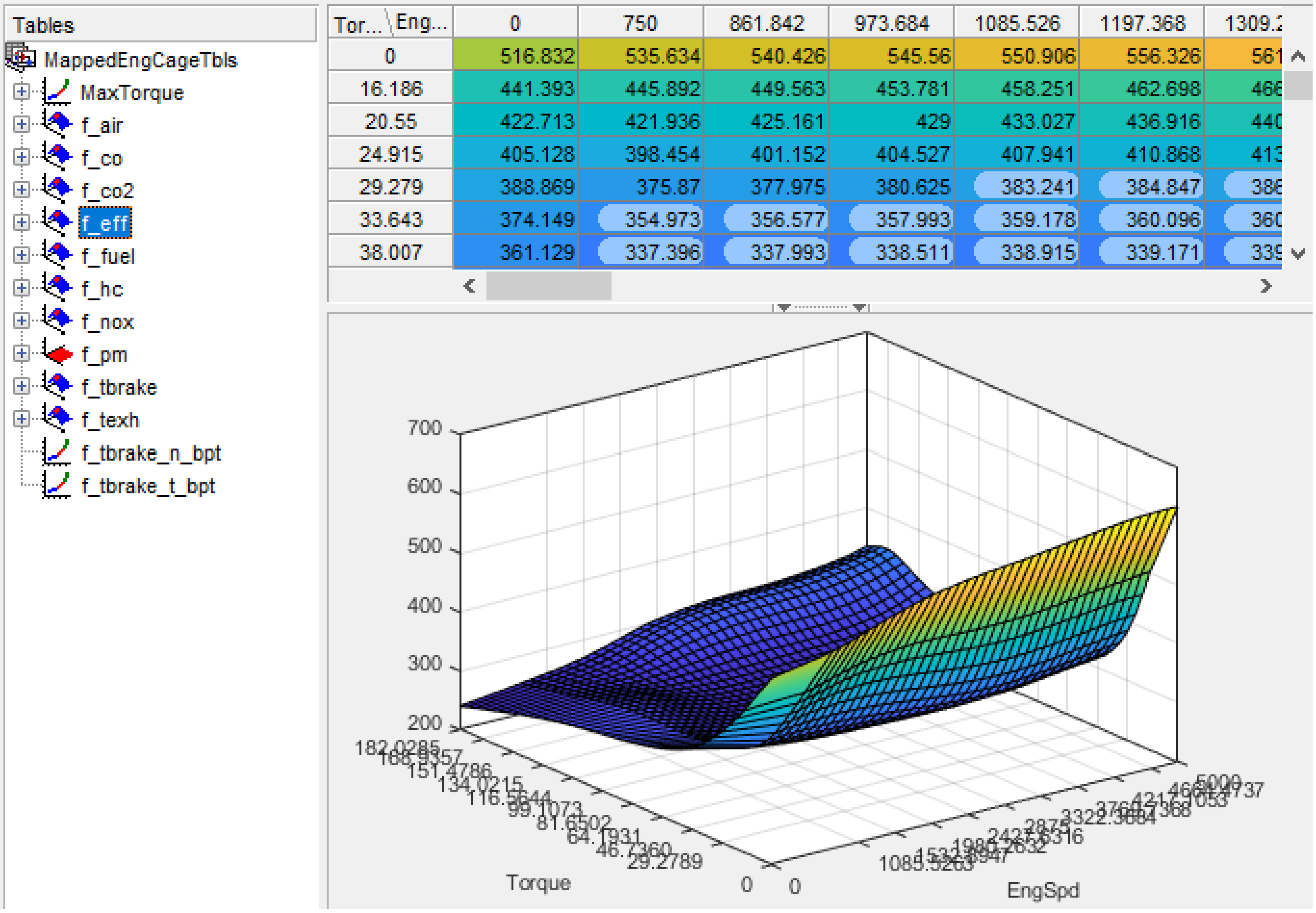
Plot: Surface

X-axis: EngSpd

Y-axis: Torque

Name	Value	Tolerance
Eng...	750:85:50(	Linked to X-...
Torq...	16.186104	Linked to Y-...

- The CAGE Browser provides the calibrated data, for example:



Tables

- MappedEngCageTbls
  - MaxTorque
  - f_air
  - f_co
  - f_co2
  - f_eff
  - f_fuel
  - f_hc
  - f_nox
  - f_pm
  - f_tbrake
  - f_texh
  - f_tbrake_n_bpt
  - f_tbrake_t_bpt

Tor... \ Eng...	0	750	861.842	973.684	1085.526	1197.368	1309.2
0	516.832	535.634	540.426	545.56	550.906	556.326	561
16.186	441.393	445.892	449.563	453.781	458.251	462.698	466
20.55	422.713	421.936	425.161	429	433.027	436.916	440
24.915	405.128	398.454	401.152	404.527	407.941	410.868	413
29.279	388.869	375.87	377.975	380.625	383.241	384.847	386
33.643	374.149	354.973	356.577	357.993	359.178	360.096	360
38.007	361.129	337.396	337.993	338.511	338.915	339.171	339

Step 2: Apply Calibration to Mapped Engine Model

When you click **Apply calibration to mapped engine model**, Powertrain Blockset:

- Updates the Mapped SI Engine block parameters with the calibrated table and breakpoint data.
- Sets the Mapped SI Engine as the active variant.
- Executes the engine mapping experiment.

When the dynamometer engine mapping completes, use the Simulation Data Inspector to verify the results.

See Also

SI Core Engine | Mapped SI Engine

More About

- “Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15
- “What Is CAGE?” (Model-Based Calibration Toolbox)

- “Mapped SI Lookup Tables as Functions of Engine Torque and Speed” (Model-Based Calibration Toolbox)

Generate Deep Learning SI Engine Model

If you have Deep Learning Toolbox, and Statistics and Machine Learning Toolbox, you can generate a dynamic (transient) deep learning spark-ignition (SI) engine model to use for hardware-in-the-loop (HIL) testing, powertrain control, diagnostics, and estimator algorithm design. For example, from physical hardware or from a high-fidelity model, you can fit a deep learning model to measured engine-out transient emissions data and use it for aftertreatment control and diagnostic algorithm development.

To train the deep learning SI engine model, Powertrain Blockset uses this transient SI engine dataset with fixed sampling.

Input Data	Output Data
• Throttle position (%) • Wastegate area (%) • Engine speed (RPM) • Intake cam advance from park (deg) • Exhaust cam retard from park (deg) • Spark retard from nominal (deg) • Lambda — Air/Fuel ratio, normalized to stoichiometric Air/Fuel ratio	• Airflow — Intake air mass flow rate (kg/s) • Torque (N·m) • Intake manifold gas pressure (Pa) • Exhaust gas temperature (K)

To generate the deep learning engine model,

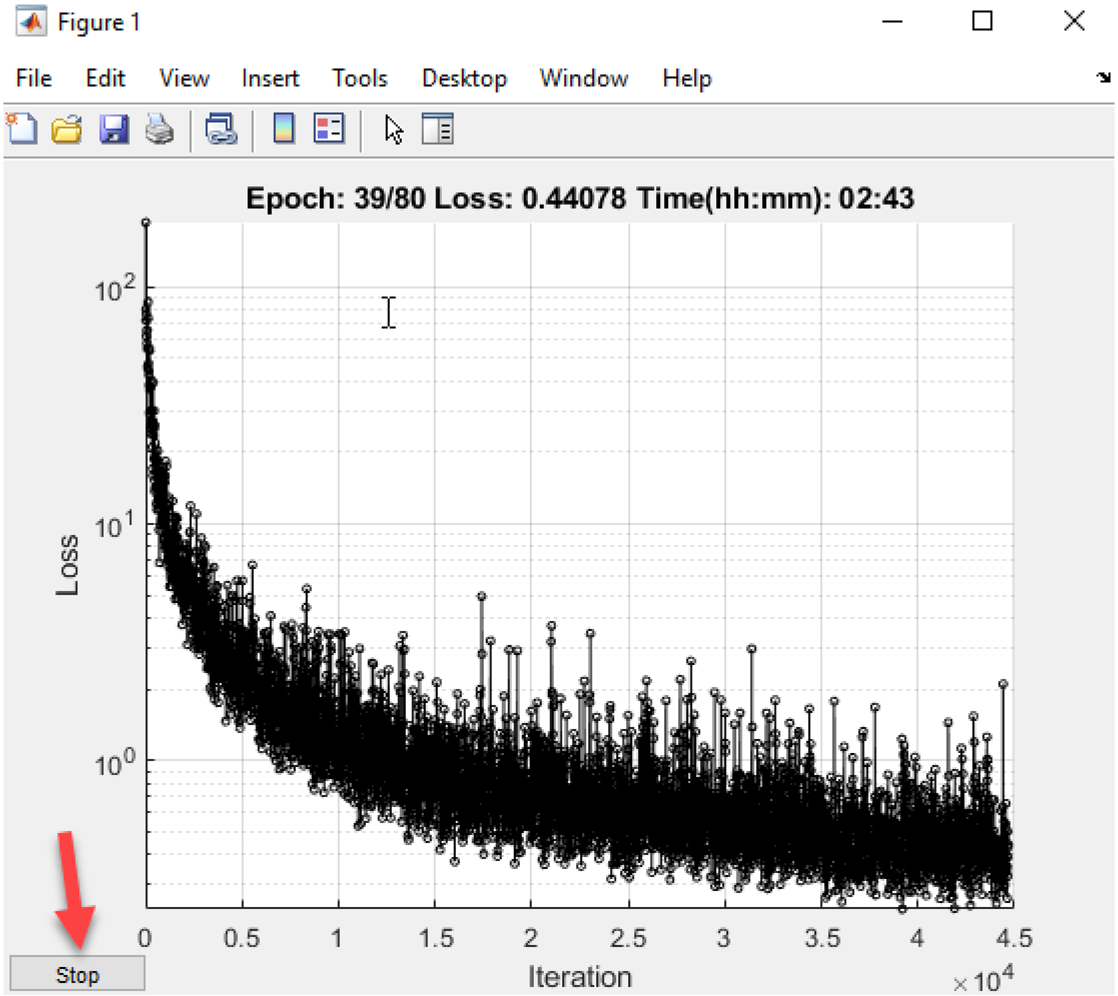
1. Start the SI engine dynamometer reference application. Enter this command in the MATLAB Command Window.

   ```
   autoblkSIDynamometerStart
   ```

   The data used for the Engine Dynamometer unpacks.
2. When the Engine Dynamometer window appears, double-click the **Generate Deep Learning Engine Model** box in the lower part of the window. Model generation can take several hours.

   By default, to train the deep learning engine model, the reference application supplies input and response data from a design of experiment (DoE) set of input data from the SI Core Engine block. Alternatively, you can use data acquired by physical testing, or generated by Powertrain Blockset from Gamma Technologies LLC engine models or other high-fidelity engine models.

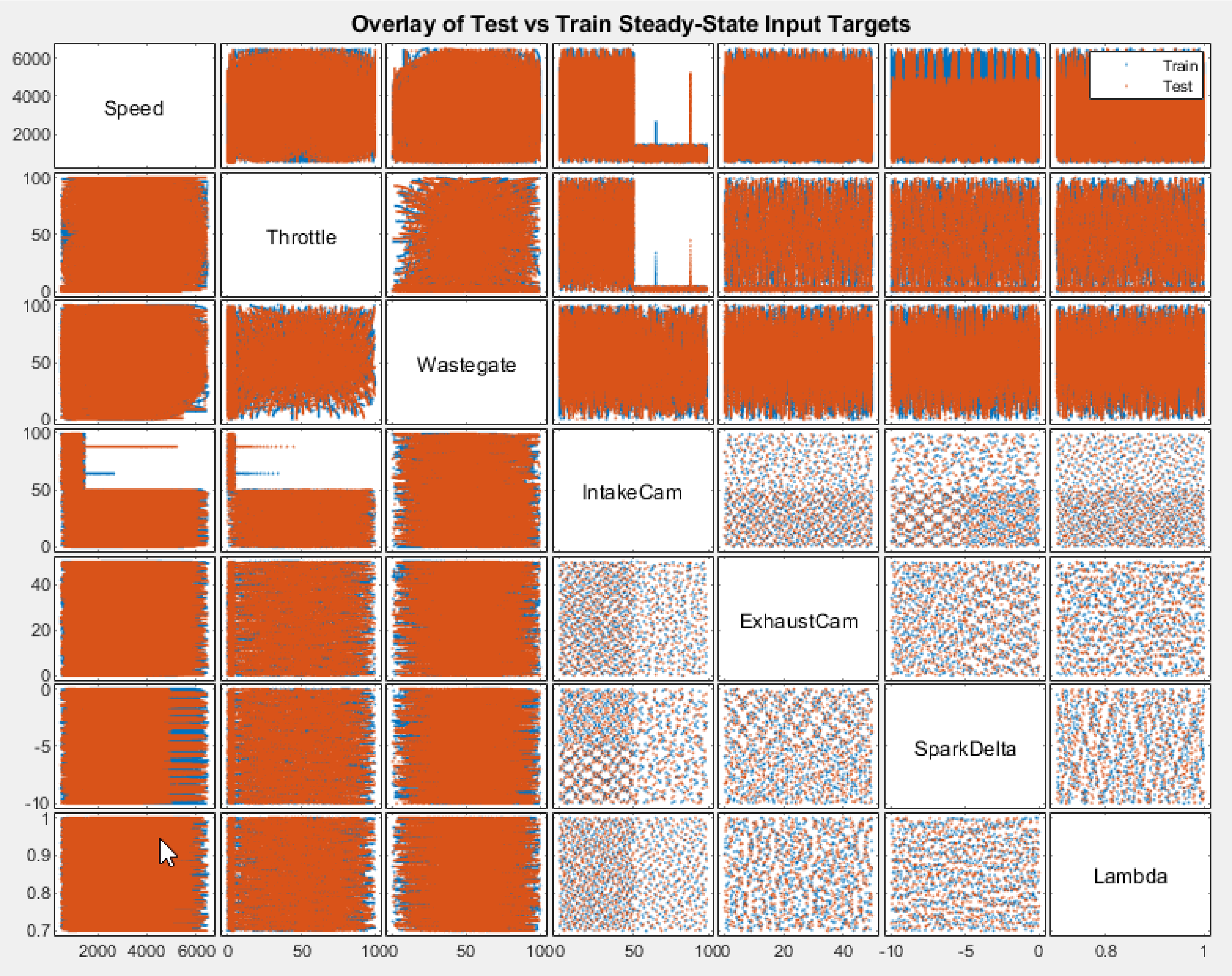
   - During model generation, the training progress window shows how the deep learning loss function (cost function) varies vs. iterations. You can also stop the training process from this window. When processing is complete, the **Stop** button turns green.



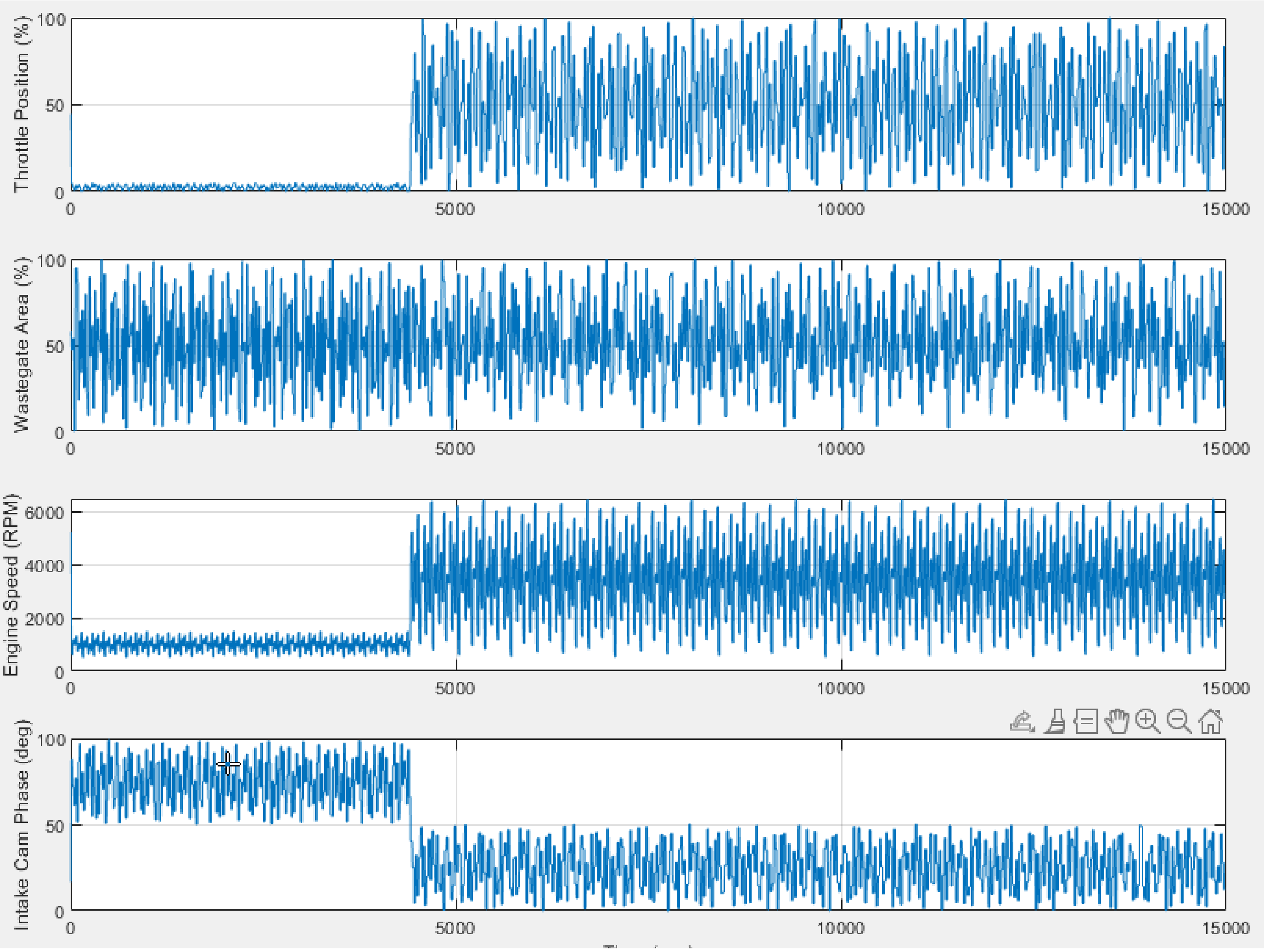
Note that the Powertrain Blockset uses half the data to train the model, and half to test the model.

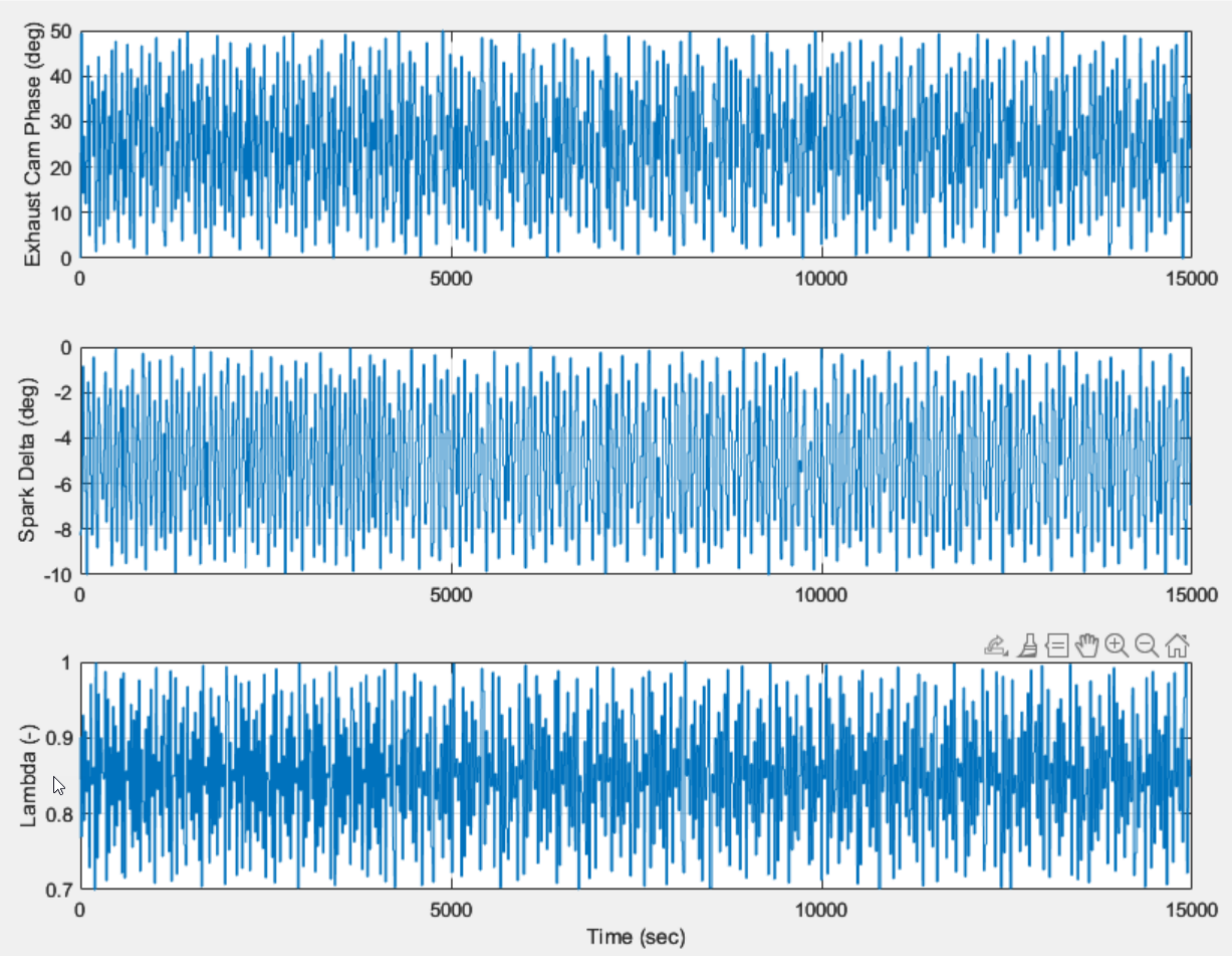
3 Once you generate the deep learning SI model, you can view the results.

- These pairwise overlays show “test versus train dataset input” at steady state. Use them to check that the data used to train and to test the model span the same space, with roughly the same density.

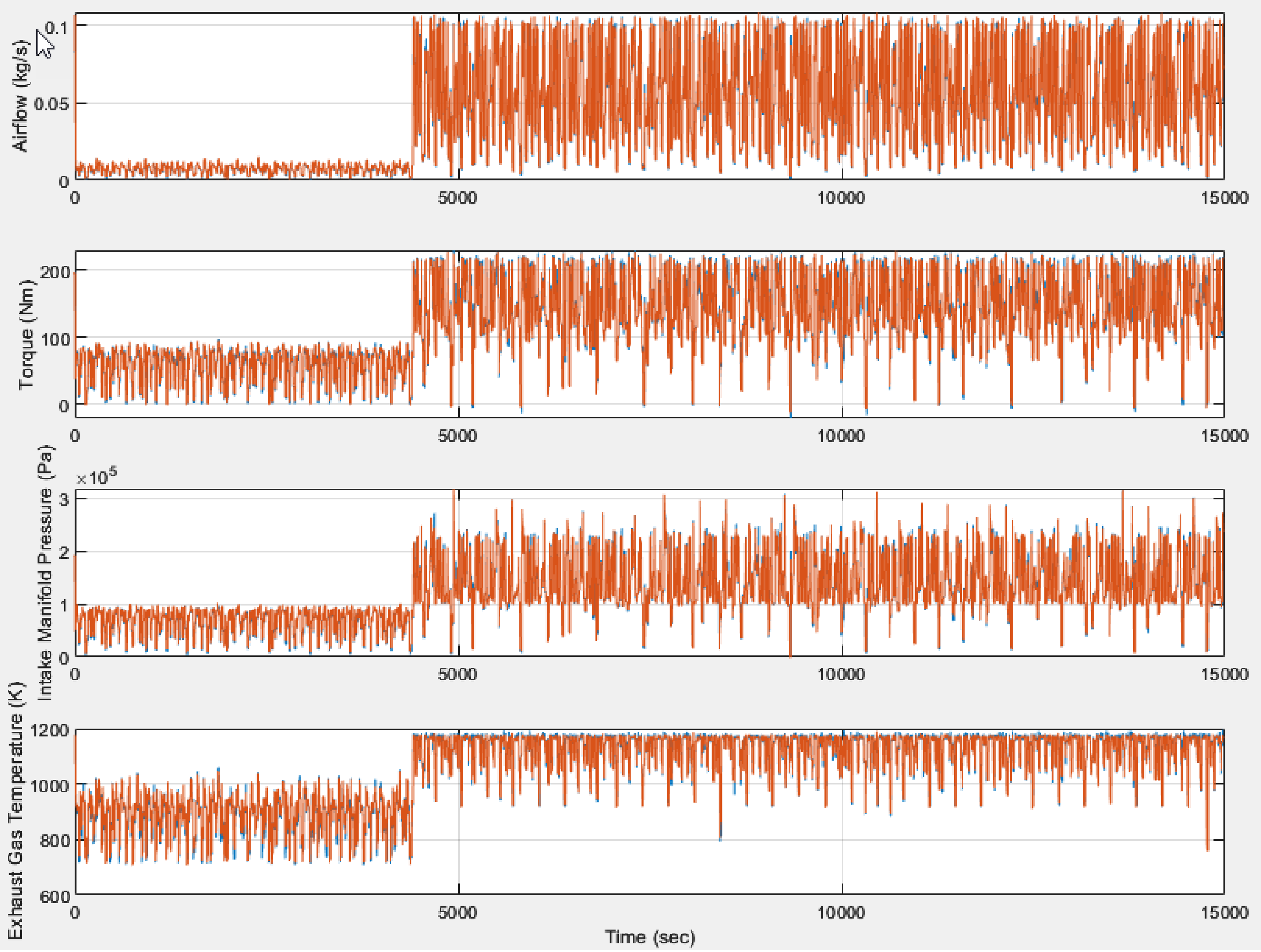


- These plots show the seven engine input signals the deep learning model uses to test its ability to create the output responses.

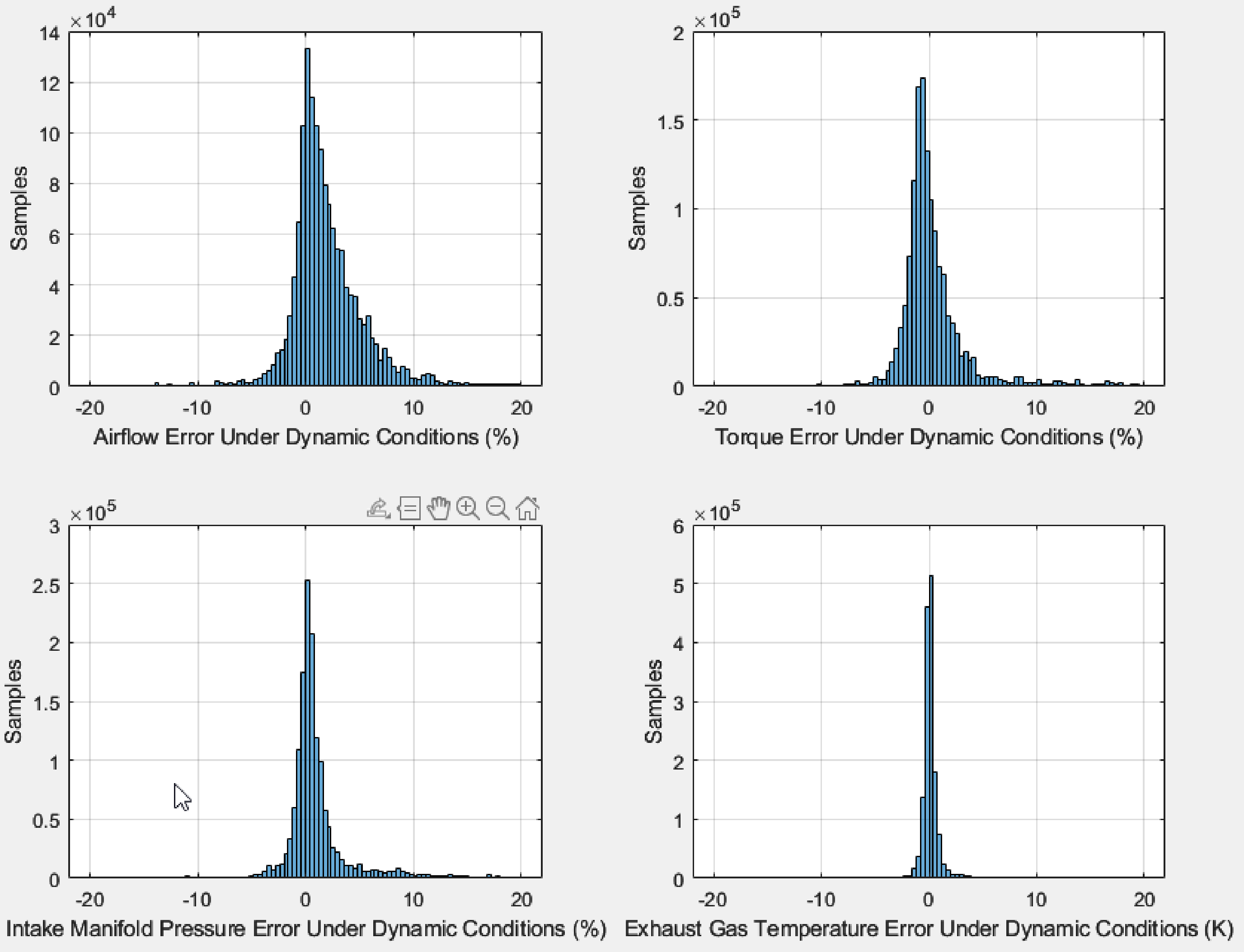

Throttle Position (%)
100
50
0
0
5000
10000
15000
Wastegate Area (%)
100
50
0
0
5000
10000
15000
Engine Speed (RPM)
6000
4000
2000
0
0
5000
10000
15000
Intake Cam Phase (deg)
100
50
0
0
5000
10000
15000



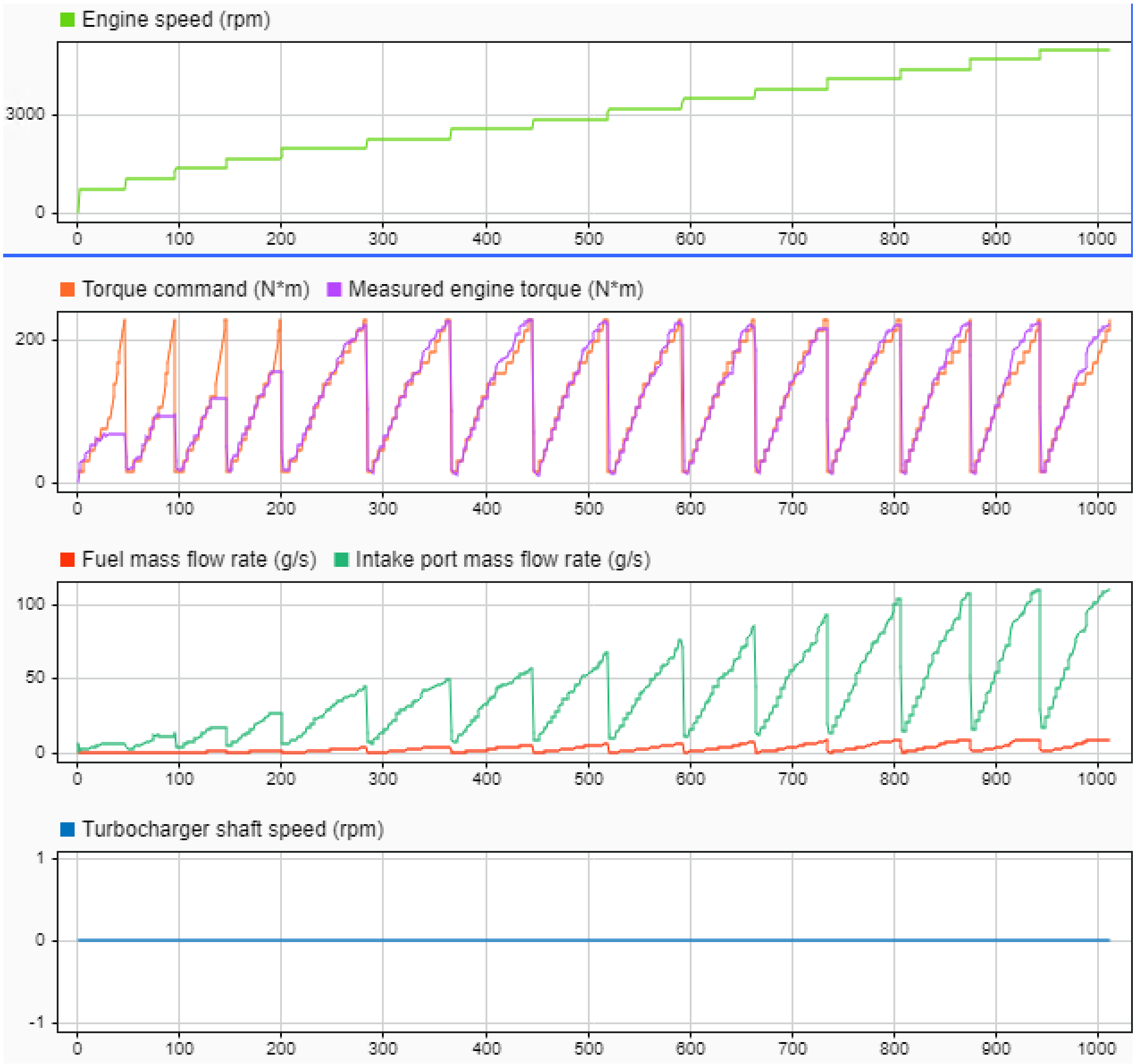
- These four plots show the engine outputs. Each plot displays the SI engine deep learning model predicted output in red and the test data in blue.



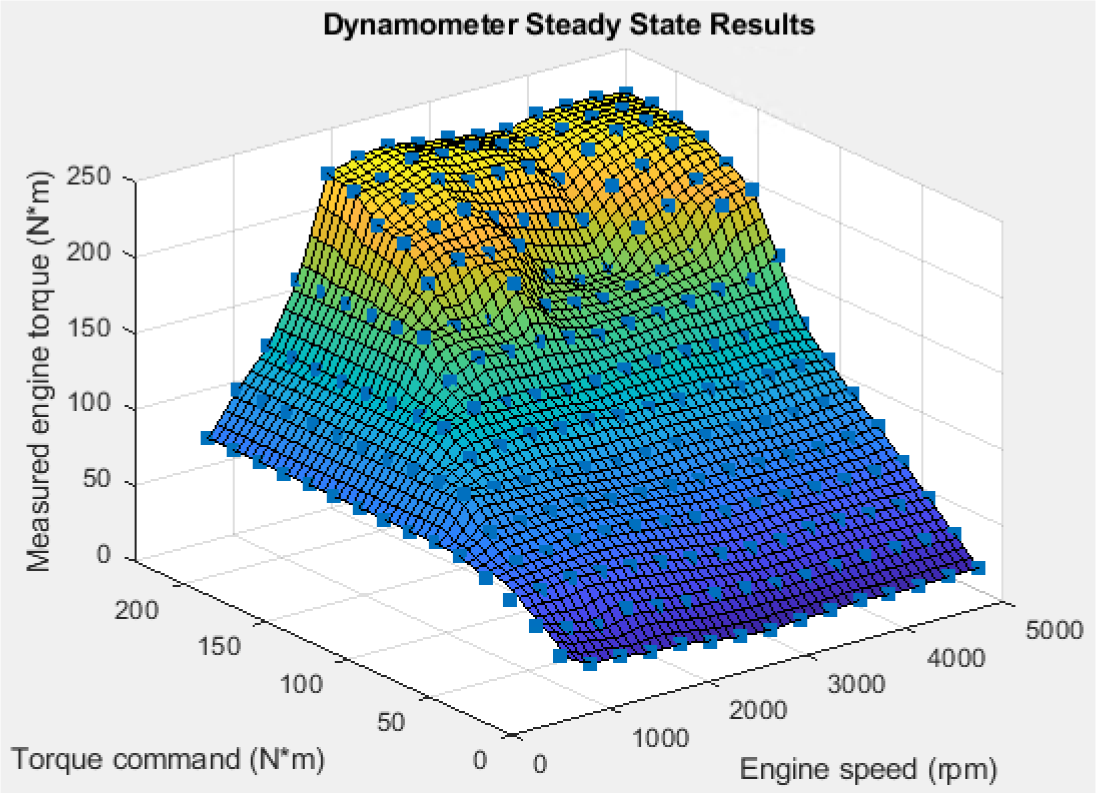
- These histograms display the modeling error distribution for the four engine outputs, under dynamic (transient) conditions. The error is the difference between the response predicted by the deep learning model and the measured test response of the engine.



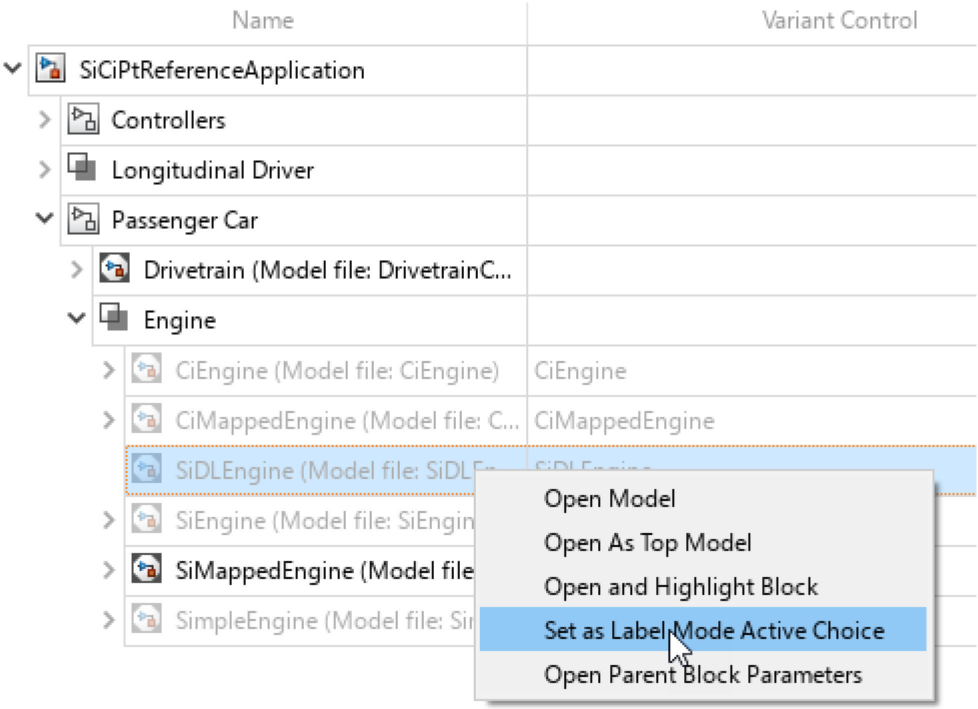
- The Simulation Data Inspector displays the results of an engine performance test of the trained SI engine deep learning model over a matrix of engine speeds and commanded engine torque. You can use the commanded versus measured torque response comparisons to assess the suitability of the deep learning model for use in a vehicle model.



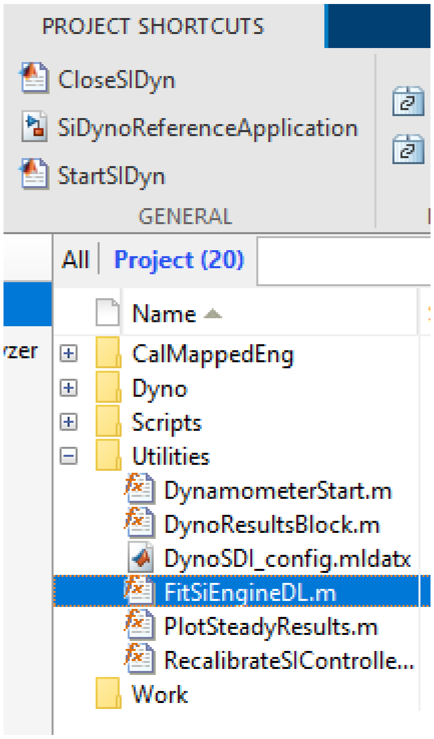
- This 3-D plot shows output torque versus commanded torque from the quasi-steady state portions of the previous plot. The plot shows each state as a blue dot. The mesh shows a curve fit to those states, so you can visualize the overall output of the SI deep learning model.



4 You can choose the deep learning SI model `SiDLEngine` as your engine plant model variant in the conventional vehicle and hybrid electric vehicle (HEV) reference applications in the Powertrain Blockset. For example, in the conventional vehicle reference application, on the **Modeling** tab, in the **Design** section, open the Variant Manager. Navigate to **Passenger Car** > **Engine**. Right-click `SiDLEngine` and select Set as Label Mode Active Choice.


Name
Variant Control
SiCiPtReferenceApplication
Controllers
Longitudinal Driver
Passenger Car
Drivetrain (Model file: DrivetrainC...
Engine
CiEngine (Model file: CiEngine)
CiEngine
CiMappedEngine (Model file: C...
CiMappedEngine
SiDLEngine (Model file: SiDLE...
SiDLEngine
SiEngine (Model file: SiEngin
SiMappedEngine (Model file
SimpleEngine (Model file: Si
Open Model
Open As Top Model
Open and Highlight Block
Set as Label Mode Active Choice
Open Parent Block Parameters

5 To fit your own deep learning SI engine model or adjust the deep learning training settings, go to the reference application project folder and run the `FitSiEngineDL.m` script in the reference application project folder.



See Also

SI Core Engine | Mapped SI Engine

More About

- “Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15
- “Deep Learning Toolbox”
- “Statistics and Machine Learning Toolbox”

Internal Combustion Mapped and Dynamic Engine Models

When you customize a SI or CI reference application, you can use either a dynamic or mapped engine model. The table provides considerations for using either implementation.

Type		Implementation	When to Use
Mapped	`CiMappedEngine` `SiMappedEngine`	Model uses a set of steady-state lookup tables to characterize engine performance. The tables provide overall engine characteristics, including actual torque, fuel flow rate, BSFC, and engine-out exhaust emissions.	• If you have engine data from a dynamometer or a design tool like GT-POWER. • For quasi steady-state engine simulations.
Dynamic	`CiEngine` `SiEngine`	Model decomposes the engine behavior into engine characteristics that are separated into lower-level components. By combining components in this way, the models capture the dynamic effects.	• If you need a more detailed dynamic model and have component-level data. • To analyze the impact of individual engine components on the overall performance.

See Also

More About

- Mapped CI Engine
- Mapped SI Engine
- CI Core Engine
- SI Core Engine
- “Engine Calibration Maps” on page 2-31

Analyze Power and Energy

To assess powertrain efficiency, you can evaluate and report power and energy for component-level blocks and system-level reference applications.

These reference applications include live scripts that analyze the energy consumption. After you open the reference applications, double-click **Analyze Power and Energy** to open the live script. To generate the energy report, select **Run**.

- "Build a Conventional Vehicle Model" on page 3-5
- "Build Hybrid Electric Vehicle Multimode Model" on page 3-19
- "Build Hybrid Electric Vehicle Input Power-Split Model" on page 3-42
- "Build Hybrid Electric Vehicle P0 Model" on page 3-51
- "Build Hybrid Electric Vehicle P1 Model" on page 3-58
- "Build Hybrid Electric Vehicle P2 Model" on page 3-65
- "Build Hybrid Electric Vehicle P3 Model" on page 3-75
- "Build Hybrid Electric Vehicle P4 Model" on page 3-82
- "Build Full Electric Vehicle Model" on page 3-26

The plant model blocks calculate transferred, stored, and not transferred power. The blocks use the Power Accounting Bus Creator to log the power signals that the live script uses. If you use your own block in the reference application, add the Power Accounting Bus Creator to your subsystem to log the power signals.

The live script provides:

- An overall energy summary that the script exports to an Excel spreadsheet.
- Engine plant, electric plant, and drivetrain efficiencies, including an engine plant histogram of time spent at different efficiencies.
- Data logging so that you can use the Simulation Data Inspector to analyze the powertrain efficiency, power, and energy signals.

Live Script

The live script uses the `autoblks.pwr.PlantInfo` class to turn on data logging, run the simulation, and report power and energy results. Before running the simulation, the script finds all of the Power Accounting Bus Creator blocks in the model and turns on data logging. During the simulation, the model logs the transferred, not transferred, and stored power. The script uses the logged data to calculate efficiency, energy loss, energy input, and energy output for each component and subsystem. If the component does not conserve energy, the script issues warnings. Finally, the script provides an overall vehicle energy summary, a detailed subsystem summary, and Simulation Data Inspector time series plots.

Run Simulation

When you run the simulation, the script creates the `autoblks.pwr.PlantInfo` object to analyze the model energy and power consumption. Use these properties to set the units:

- `PwrUnits`

- `EnrgyUnits`

When the script creates the `autoblks.pwr.PlantInfo` object, the constructor searches the model for Power Accounting Bus Creator blocks. Starting at the top-level model, the constructor creates a child object for each subsystem that contains Power Accounting Bus Creator blocks. The constructor stops at the blocks that have a Power Accounting Bus Creator.

To track the power transferred between the components, the constructor uses the transferred power ports defined in the Power Accounting Bus Creator block mask.

To calculate the efficiency, the `autoblks.pwr.PlantInfo` class `Eff` property implements this equation.

$$\eta = \left| \frac{\sum P_{output} - \sum P_{store}(P_{store} > 0)}{\sum P_{input} - \sum P_{store}(P_{store} < 0)} \right|$$

To determine if the system conserves energy, the `isEnrgyBalanced` method checks the energy conservation at each time step. If the energy conservation error is within an error tolerance, the method returns true.

Overall Summary

The overall summary provides the efficiency, energy loss, energy input, energy output, and energy stored at the component- and system-level. The summary includes hyperlinks that you can use to investigate model blocks and subsystems.

The script uses the `autoblks.pwr.PlantInfo` class `xlsSysSummary` method to export the analysis to an Excel spreadsheet.

Plant Summary

The script provides engine plant, electric plant, and drivetrain efficiencies. Specifically, the script includes the signal energy, and an engine efficiency histogram.

Simulation Data Inspector Summary

The script includes the `autoblks.pwr.PlantInfo` class `sdiSummary` method to create Simulation Data Inspector power, energy, and efficiency signal plots.

Power Signals

The system-level power and energy accounting tests that the system satisfies the conservation of energy. If the component does not conserve energy, the live script issues warnings.

The Power Accounting Bus Creator for the plant blocks in the reference applications sort the signals into three power types.

Power Type		Description	Examples
P_{trans}	Transferred	Power transferred between blocks: • Positive signals indicate flow into block • Negative signals indicate flow out of block	• Crankshaft power transferred from mapped engine to transmission. • Road load power transferred from wheel to vehicle. • Rate of heat flow transferred from throttle to manifold volume.
$P_{nottrans}$	Not transferred	Power crossing the block boundary, but not transferred: • Positive signals indicate an input • Negative signals indicate a loss	• Rate of heat transfer with the environment. • From environment is an input (positive signal) • To environment is a loss (negative signal) • Flow boundary with the environment. • From environment is an input (positive signal) • To environment is a loss (negative signal) • Mapped engine fuel flow.
P_{store}	Stored	Stored energy rate of change: • Positive signals indicate an increase • Negative signals indicate a decrease	Energy rate of change: • Battery storage • Kinetic energy in drivetrain components • Vehicle potential energy • Vehicle velocity

The power signals satisfy this equation.

$$\sum P_{trans} + \sum P_{nottrans} = \sum P_{store}$$

To conserve energy, sum of transferred power signals must be near zero.

The equations use these variables.

P_{trans}	Transferred power
$P_{nottrans}$	Not transferred power
P_{store}	Stored power
P_{input}, P_{output}	Input and output power logged by Power Accounting Bus Creator block

See Also

Power Accounting Bus Creator | `autoblks.pwr.PlantInfo`

Related Examples

- “Conventional Vehicle Powertrain Efficiency” on page 1-15

More About

- Simulation Data Inspector

Generate Mapped Fuel Cell from a Spreadsheet

If you have Model-Based Calibration Toolbox and Stateflow, you can use the fuel cell electric vehicle reference application to generate lookup tables for the Mapped Fuel Cell block. The reference application uses fuel cell data to calibrate the fuel cell and generate the tables.

1. If it is not opened, open the reference application.

   ```
   autoblkFCEvStart
   ```
2. Click **Generate Mapped Fuel Cell from Spreadsheet**.

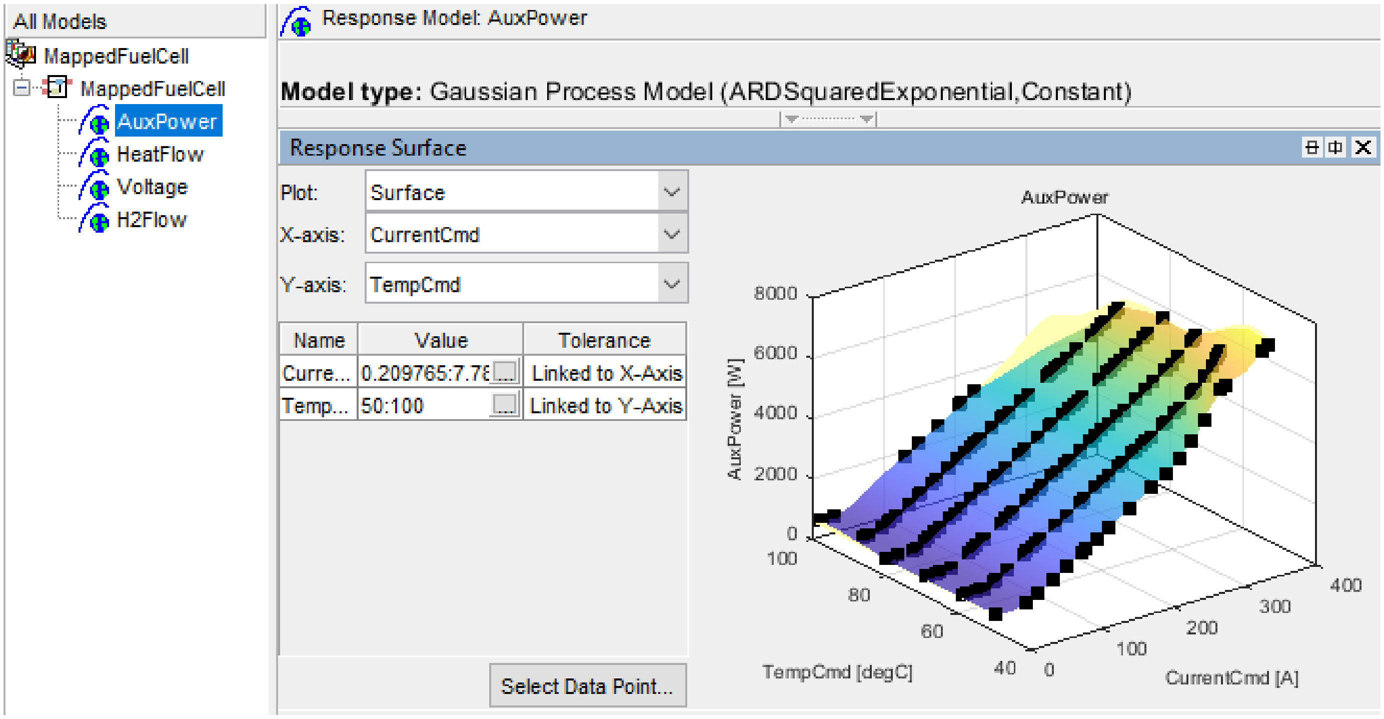
Step 1: Generate Mapped Fuel Cell Calibration

1. Use the **Spreadsheet file** field to provide a data file. By default, the reference application has `FuelCellPerformanceData.xlsx` containing the data. The tables summarize the data file requirements for generating calibrated tables.

Data	Description
CurrentCmd	Current command, in A
TempCmd	Temperature command, in C
AuxPower	Auxiliary power, in W
HeatFlow	Heat flow, in W
Voltage	Voltage, in V
H2Flow	Hydrogen flow, in kg/s

2. Click **Generate mapped fuel cell calibration** to generate response surface models in the Model-Based Calibration Toolbox and calibration in CAGE (CAlibration GEneration). CAGE and the model browser open when the process completes. To calibrate the data, Model-Based Calibration Toolbox uses templates.

   - The Model Browser provides the response model fits for the data contained in the data file, for example:



Response Model: AuxPower

Model type: Gaussian Process Model (ARDSquaredExponential,Constant)

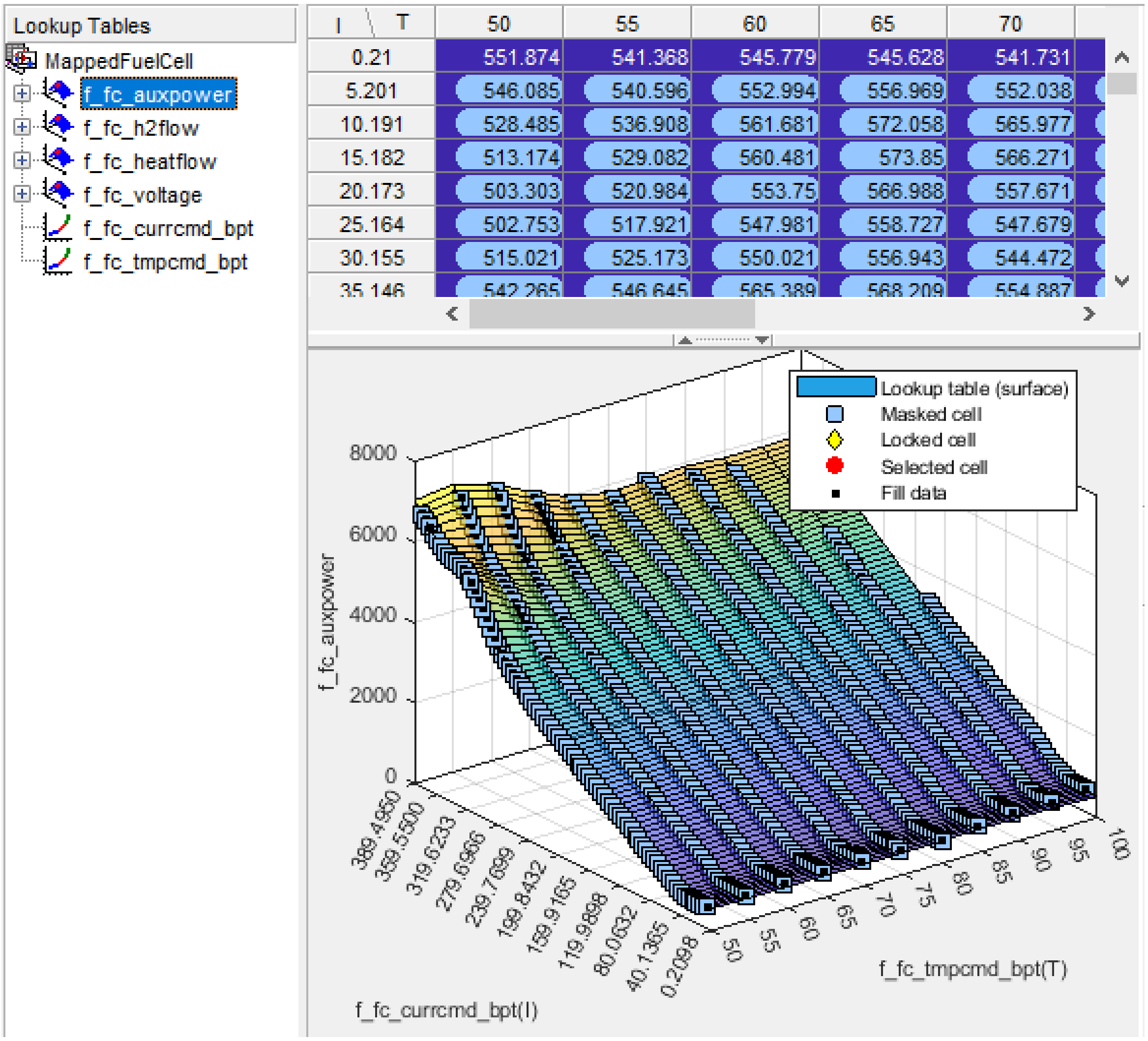
Plot: Surface

X-axis: CurrentCmd

Y-axis: TempCmd

Name	Value	Tolerance
Curre...	0.209765:7.7ε	Linked to X-Axis
Temp...	50:100	Linked to Y-Axis

- The CAGE Browser provides the calibrated data, for example:



Lookup Tables

- MappedFuelCell
  - f_fc_auxpower
  - f_fc_h2flow
  - f_fc_heatflow
  - f_fc_voltage
  - f_fc_currcmd_bpt
  - f_fc_tmpcmd_bpt

I \ T	50	55	60	65	70
0.21	551.874	541.368	545.779	545.628	541.731
5.201	546.085	540.596	552.994	556.969	552.038
10.191	528.485	536.908	561.681	572.058	565.977
15.182	513.174	529.082	560.481	573.85	566.271
20.173	503.303	520.984	553.75	566.988	557.671
25.164	502.753	517.921	547.981	558.727	547.679
30.155	515.021	525.173	550.021	556.943	544.472
35.146	542.265	546.645	565.389	568.209	554.887

Step 2: Apply Calibration to Mapped Fuel Cell Model

When you click **Apply calibration to mapped fuel cell model**, Powertrain Blockset:

- Updates the Mapped Fuel Cell block parameters with the calibrated table and breakpoint data.
- Sets the Mapped Fuel Cell as the active variant.
- Executes the fuel cell mapping experiment.

When the reference application fuel cell mapping completes, use the Simulation Data Inspector to verify the results.

See Also

Datasheet Battery | Mapped Motor

Related Examples

- “FCEV Reference Application” on page 7-12

More About

- “Build Fuel Cell Electric Vehicle” on page 3-37
- “What Is CAGE?” (Model-Based Calibration Toolbox)

4

Project Templates

CI Engine Project Template

The Powertrain Blockset software provides a project template for compression-ignition (CI) engines. Use the template to create engine variants that you can use with the internal combustion engine reference application projects. The project template contains CI engine controller and plant models.

Use the project template to create CI engine variants for these reference applications:

- Conventional vehicle
- Hybrid electric vehicles
- CI engine dynamometer

To open the CI engine project template:

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

   In the Simulink start page, browse to Powertrain Blockset and select **CI Engine Project**.
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 CI or SI engine model.

Controller

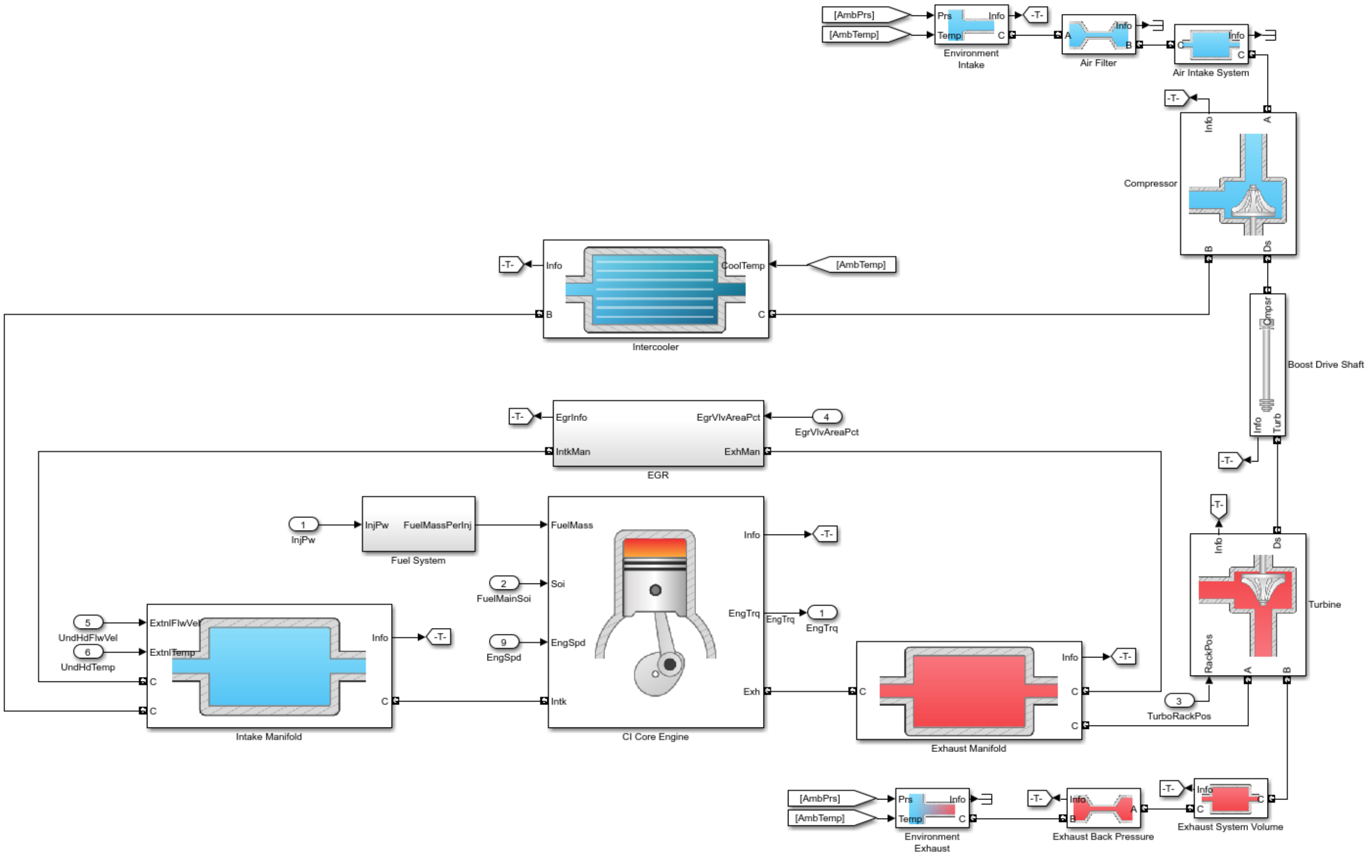
The `Controller` folder contains the `CiEngineController.slx` model. The model uses the CI Controller block and a `Start Stop Logic` subsystem to control the CI engine plant model.

Plant

The `Plant` folder contains models that represent dynamic and mapped CI engines. By default, the dynamic and mapped engines are configured for a 1.5–L engine with a variable geometry turbocharger (VGT).

Dynamic

`CiEngineCore.slx` contains the engine intake system, exhaust system, exhaust gas recirculation (EGR), fuel system, core engine, and turbocharger subsystems.



Mapped

`CiMappedEngine.slx` uses the Mapped CI Engine block to look up power, air mass flow, fuel flow, exhaust temperature, efficiency, and emission performance as functions of engine speed and injected fuel mass.

See Also

Mapped CI Engine | CI Core Engine | CI Controller

More About

- “Internal Combustion Engine Reference Application Projects” on page 3-2
- Simulink Projects
- “Variant Systems”

SI Engine Project Template

The Powertrain Blockset software provides a project template for spark-ignition (SI) engines. Use the template to create engine variants that you can use with the internal combustion engine reference application projects. The project template contains SI engine controller and plant models.

Use the project template to create CI engine variants for these reference applications:

- Conventional vehicle
- Hybrid electric vehicles
- SI engine dynamometer

To open the SI engine project template:

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

   In the Simulink start page, browse to Powertrain Blockset and select **SI Engine Project**.
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 CI or SI engine model.

Controller

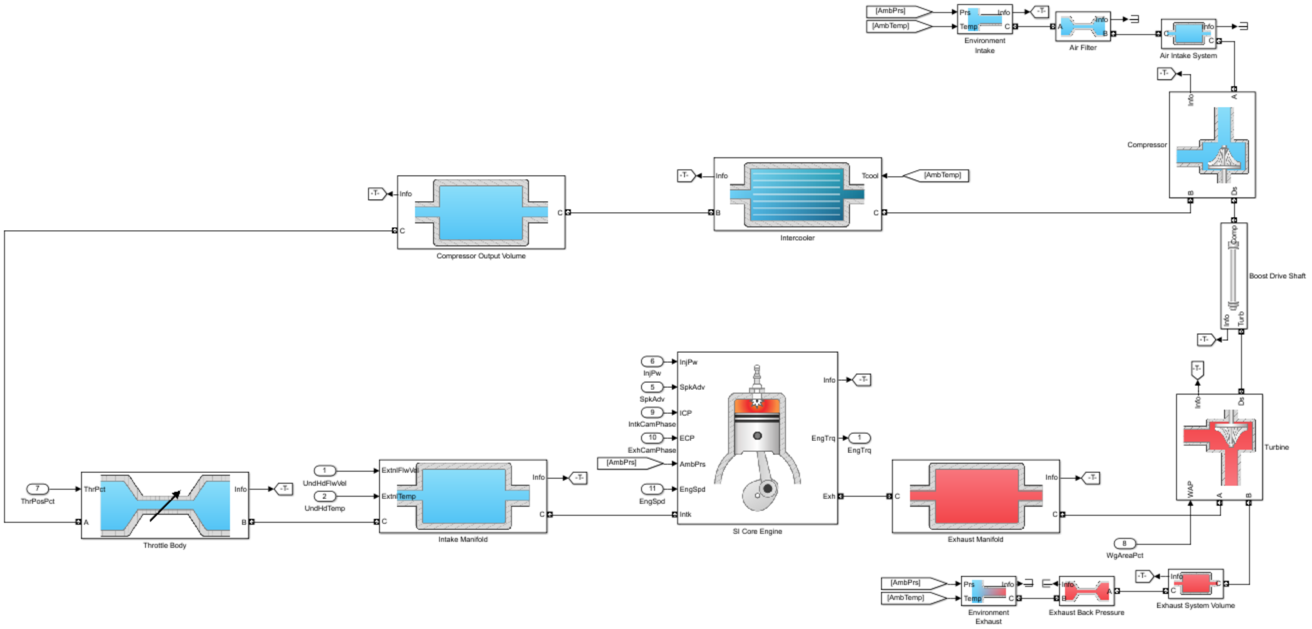
The `Controller` folder contains the `SiEngineController.slx` model. The model uses the SI Controller block and a `Start Stop Logic` subsystem to control the SI engine plant model.

Plant

The `Plant` folder contains models that represent dynamic and mapped SI engines. By default, the dynamic and mapped engines are configured for a 1.5–L turbocharged engine.

Dynamic

`SiEngineCore.slx` contains the engine intake system, exhaust system, core engine, and turbocharger subsystems.



Mapped

`SiMappedEngine.slx` uses the Mapped SI Engine block to look up power, air mass flow, fuel flow, exhaust temperature, efficiency, and emission performance as functions of engine speed and commanded torque.

See Also

SI Core Engine | Mapped SI Engine | SI Controller

More About

- “Internal Combustion Engine Reference Application Projects” on page 3-2
- Simulink Projects
- “Variant Systems”

5

Supporting Data

Install Drive Cycle Data

This example shows how to install additional drive cycle data for the Drive Cycle Source block. By default, the 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 privileges for the Installation folder.

See Also

Drive Cycle Source

More About

- "Track Drive Cycle Errors" on page 5-3

Track Drive Cycle Errors

This example shows how to use the Drive Cycle Source block to identify drive cycle faults when you run the conventional vehicle reference application with the `FTP—75` drive cycle.

1. Open the conventional vehicle reference application project. By default, the application has a `FTP—75` drive cycle with error tracking disabled.

   ```
   autoblkConVehStart
   ```

   Project files open in a writable location.
2. Open the Drive Cycle Source block. On the **Fault Tracking** tab, select these parameters:

   - **Enable fault tracking**
   - **Enable failure tracking**
3. Review the parameters that specify the fault and failure conditions. If the vehicle speed is not within the allowable speed range during the time tolerance, the block sets a fault condition. Accept the default EPA dynamometer driving schedule parameter settings by clicking **OK**.

   This table provides the settings for the EPA standard and the Worldwide Harmonised Light Vehicle Test Procedure (WLTP) laboratory tests.

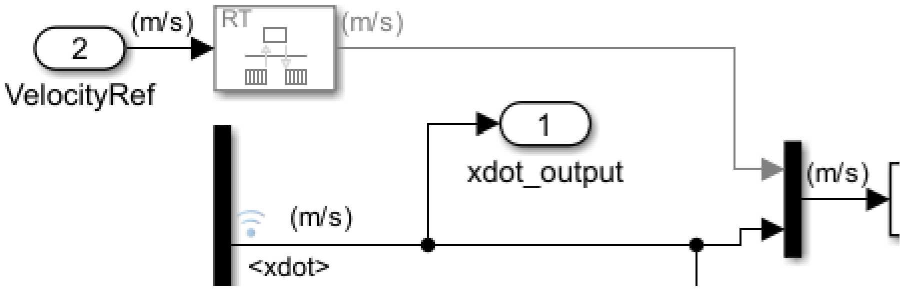
Parameter	Description	Setting	
		EPA Standard[1]	WLTP Tests[2]
Speed tolerance	Speed tolerance above the highest point and below the lowest point of the drive cycle speed trace within the time tolerance.	2.0 mph	2.0 km/h
Time tolerance	Time that the block uses to determine the speed tolerance.	1.0 s	1.0 s
Maximum number of faults	Maximum number of faults during the drive cycle.	*Not specified*	10
Maximum single fault time	Maximum fault duration.	2.0 s	1.0 s
Maximum total fault time	Maximum accumulated time spent under fault condition.	*Not specified*	*Not specified*

4. Connect the vehicle longitudinal velocity signal to the Drive Cycle Source block `VelFdbk` input port.

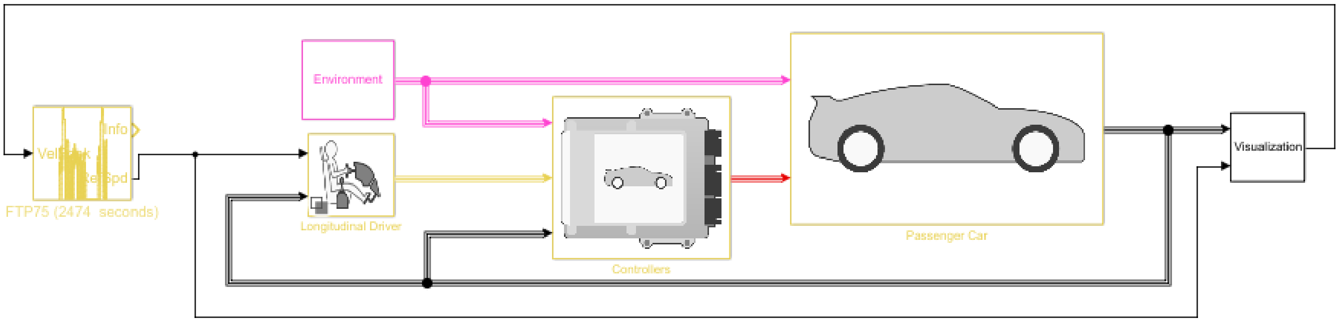
   a. In the Visualization subsystem, connect the longitudinal velocity signal, `<xdot>`, to an Outport named `xdot_output`.

   b. Determine the `<xdot>` signal units. To display signal units, on the **Debug** tab, select **Information Overlays** > **Units**. The `<xdot>` signal units are m/s.

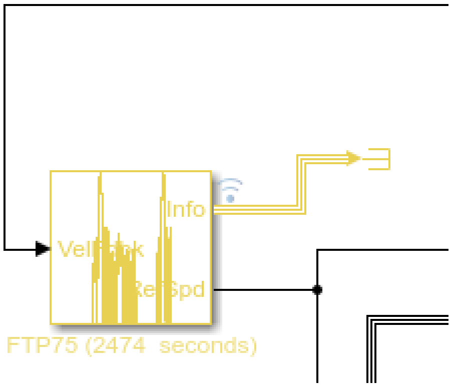
c Select the `<xdot>` signal line and `Enable Data Logging`.



d On the top level of the model, connect the Visualization output to the Drive Cycle Source block input.



5 Connect the Drive Cycle Source block `Info` output port to a Terminator port. Enable data logging.



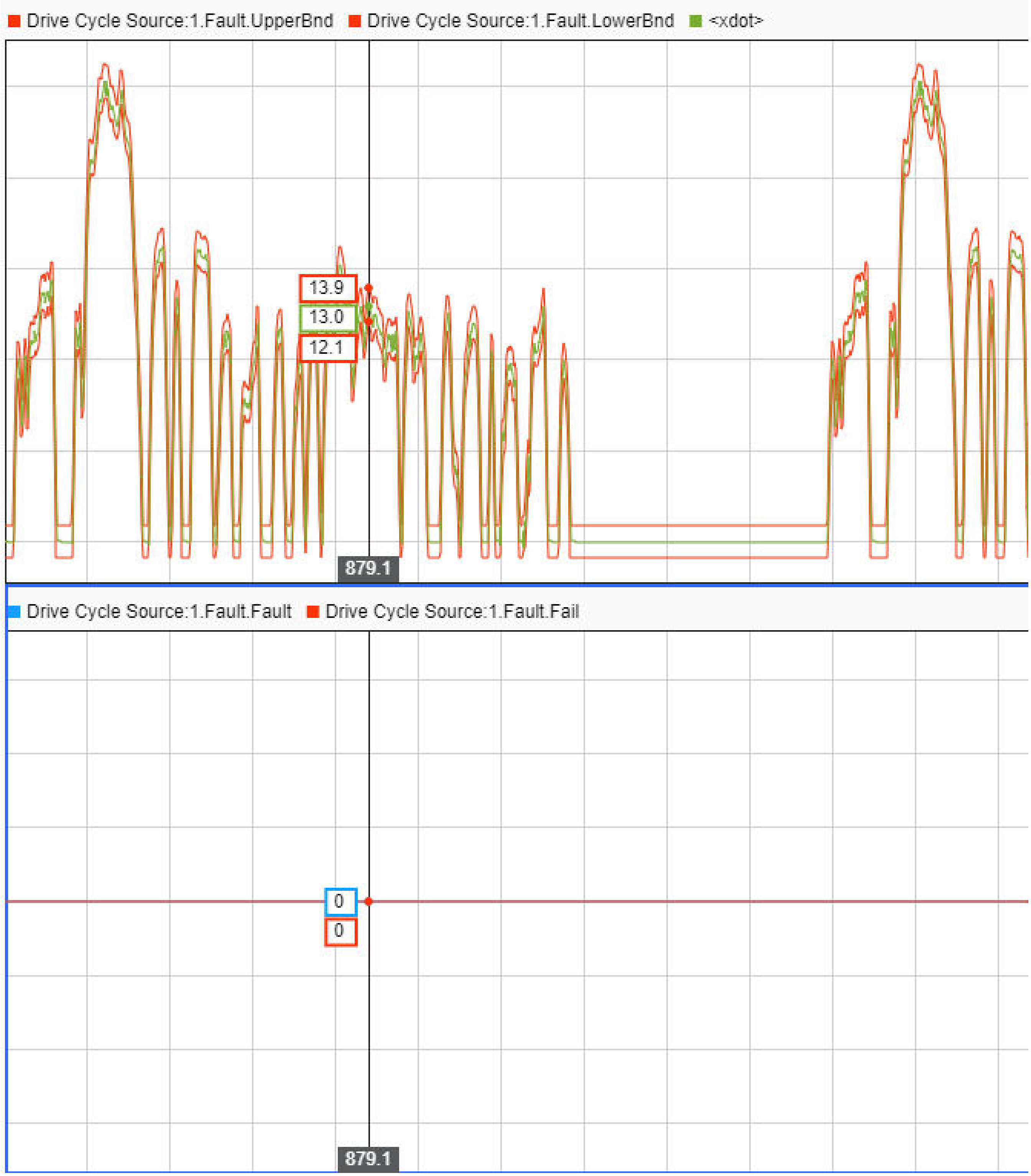
6 Save the model and run the simulation.

7 To inspect the results, use the Data Inspector. In the Simulink Toolstrip, on the **Simulation** tab, under **Review Results**, click **Data Inspector**.

These results indicate that the Drive Cycle Source block did not detect faults or failures during the drive cycle.

- `Fault` — Vehicle speed, `<xdot>`, stayed within the upper and lower bounds of the allowable speed range.

- `Fail` — Fault conditions did not exceed the maximum number of faults, maximum single fault time, or maximum total fault time.

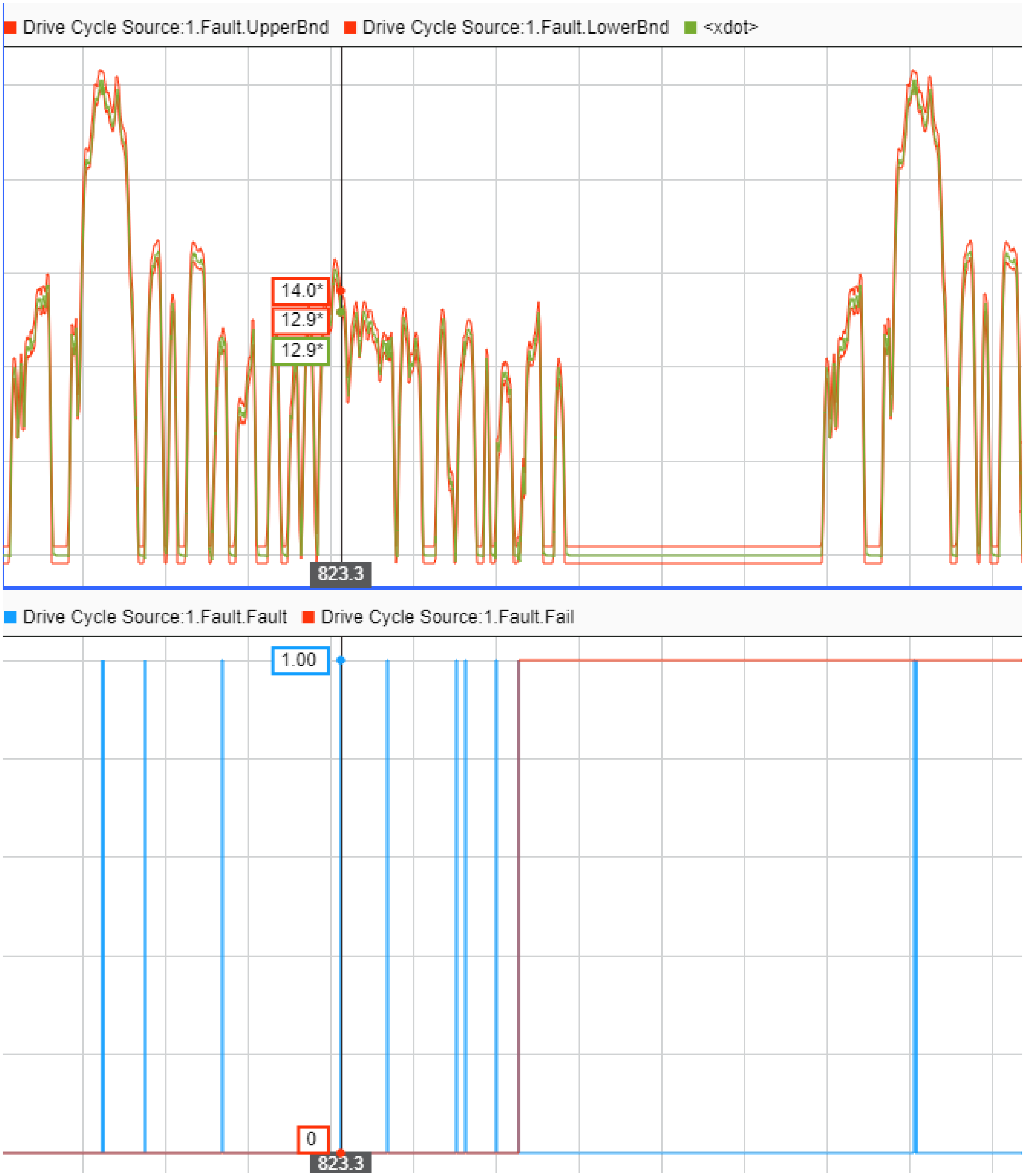


8 In the Drive Cycle Source block, set the **Speed tolerance** parameter to a tighter tolerance, for example 1 mph. The block calculates new error bounds for the speed.

9 Rerun the simulation.

10 To inspect the results, use the Data Inspector. These results indicate that the Drive Cycle Source block did detect failures and faults during the drive cycle.

- `Fault` — Vehicle speed, `<xdot>`, did not stay within the upper and lower bounds of the allowable speed range.
- `Fail` — Fault conditions exceeded the maximum number of faults, maximum single fault time, or maximum total fault time.


Drive Cycle Source:1.Fault.UpperBnd
Drive Cycle Source:1.Fault.LowerBnd
<xdot>
14.0*
12.9*
12.9*
823.3
Drive Cycle Source:1.Fault.Fault
Drive Cycle Source:1.Fault.Fail
1.00
0
823.3

References

[1] Environmental Protection Agency (EPA). *EPA urban dynamometer driving schedule*. 40 CFR 86.115-78, July 1, 2001.

[2] European Union Commission. "Speed trace tolerances". *European Union Commission Regulation*. 32017R1151, Sec 1.2.6.6, June 1, 2017.

See Also

Drive Cycle Source

More About

- “Build a Conventional Vehicle Model” on page 3-5
- “Install Drive Cycle Data” on page 5-2

6

Calibration

Generate Parameter Data for Datasheet Battery Block

This example shows how to import lithium-ion battery sheet data and generate parameters for the Datasheet Battery block.

In step 1, you import the datasheet data. Steps 2-5 show how to use curve-fitting techniques to obtain the open circuit voltage and battery resistance from the datasheet data. In steps 6-8, you validate the curve-fit voltage and battery values by comparing them to the Arrhenius behavior and the datasheet data. Finally, in step 9, you specify these Datasheet Battery block parameters:

- Rated capacity at nominal temperature
- Open circuit voltage table data
- Open circuit voltage breakpoints 1
- Internal resistance table data
- Battery temperature breakpoints 1
- Battery capacity breakpoints 2
- Initial battery charge

Step 1: Import Battery Datasheet Data

Import the battery discharge and temperature datasheet into MATLAB. Ensure that each dataset in the datasheet includes a starting battery cell output voltage. Typically, data collected at different temperatures has the same reference current. Data collected at different currents has the same reference temperature.

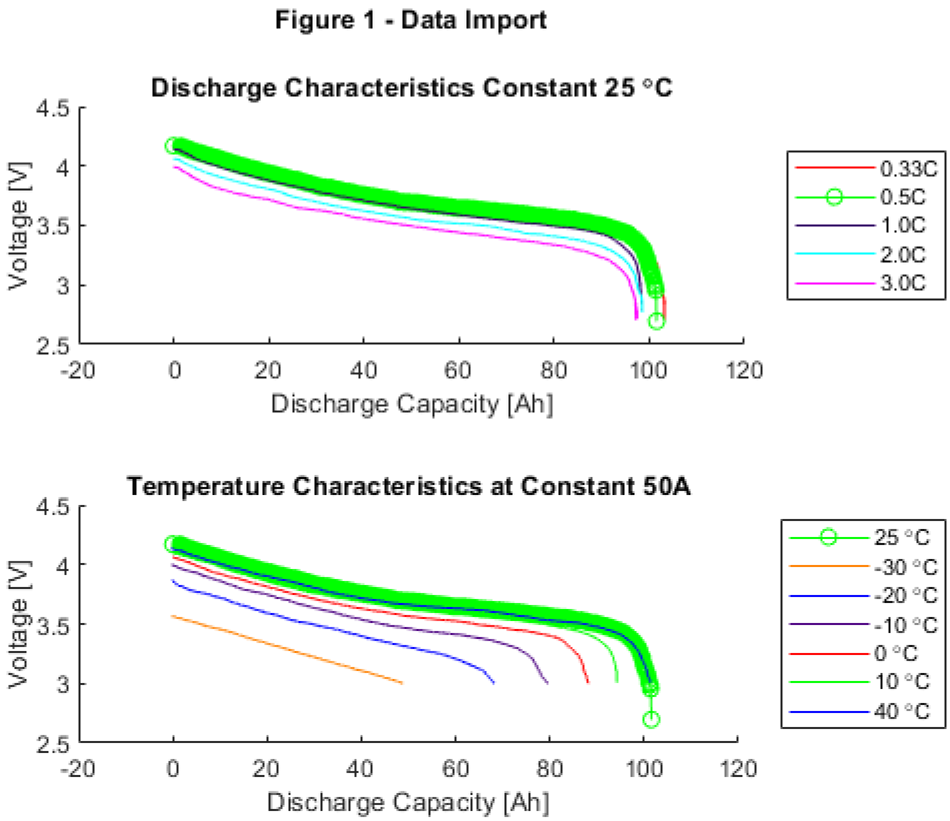
For this example, load the battery datasheet discharge and temperature data for a lithium-ion battery from a file that contains 12 data sets. Each data set corresponds to battery data for a specific current and temperature. The data sets each have two columns. The first column contains the discharge capacity, in percent. The second column contains the corresponding battery cell voltage.

```
exp_data=load('ex_datasheetbattery_liion_100Ah.mat');
```

The example does not use the data set that corresponds to a current of 500 A at 25 ºC.

Plot the discharge and temperature curves. Figure 1 shows the lithium-ion battery discharge characteristics at constant temperature (at five levels of current, shown as C-rate) and constant current (at six temperatures). Figure 1 indicates the curve that corresponds to the reference temperature of 25 ºC and the reference current of 50 A.

```
ex_datasheetbattery_plot_data
```



Step 2: Normalize State-of-Charge (SOC) Data

To represent 1-SOC capacity at constant temperature, normalize the relative discharge capacity with values between 0 and 1. Let 1 represent a fully discharged battery.

Set `ref_exp` to the dataset that corresponds to the reference temperature of 25 °C and the reference current of 50 A. Typically, the reference temperature is room temperature.

```
ref_exp = 2;
```

If you have several data sets, use a few for validation. Do not include them as part of the estimation dataset.

For this example, use `val_exp` to set up the validation and estimation data sets. Let 1 represent a validation dataset and 0 represent an estimation dataset.

```
val_exp = logical([1 0 0 0 1 0 0 0 0 1 0]);
```

Define reference current and temperature. For this example, the reference temperature is 25 °C and the reference current is 50 A.

```
ref_curr = current == current(ref_exp);
ref_temp = temperature == temperature(ref_exp);

[sort_current, sort_index_current] = sort(current(ref_temp));
[sort_temp, sort_index_temp] = sort(temperature(ref_curr));
N = length(current); % Number of experiments
```

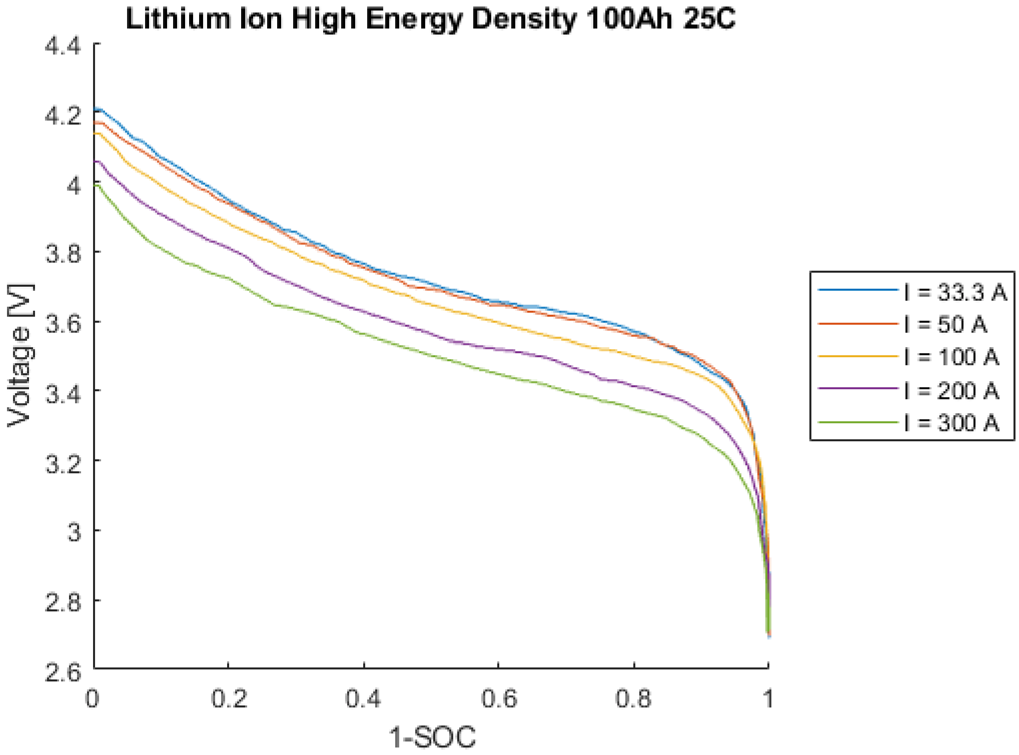
Prepare normalized x axes for each data set and find the actual capacity. x is a structure with as many fields as data sets and values between 0 and 1.

```
for i=1:N
    x.(['curr' current_label{i} '_temp' temperature_label{i}]) = ...
            exp_data.([label '_' current_label{i} '_' temperature_label{i}])(:,1)/...
            exp_data.([label '_' current_label{i} '_' temperature_label{i}])(end,1);
    % Calculate actual capacity for each datasheet
    correct_cap.(['curr' current_label{i} '_temp' temperature_label{i}]) = ...
            exp_data.([label '_' current_label{i} '_' temperature_label{i}])(end,1);
end
```

Plot the normalized SOC data.

```
ex_datasheetbattery_plot_soc
```

Figure 2 - Normalized SOC Data



Step 3: Fit Curves

Create `fitObj` curves for constant temperatures at different discharge rates and constant discharge rates at different temperatures. Use the `fitObj` curves to create a matrix of cell/module voltage versus discharge current at varying levels of SOC.

`fitObj` is a structure of fit objects that contains as many fields as data sets. The structure fits a discharge voltage to the normalized `([0,1])` extracted Ah. This allows the discharge curves to be algebraically combined to calculate resistance at each SOC level.

Define state of charge vector and breakpoints.

```
SOC_LUT = (0:.01:1)';
SOCbkpts = 0:.2:1;
```

Fit the discharge curves at different currents for reference temperature.

```
for i=find(ref_temp)
    fitObj.(['fit' current_label{i}]) = ...
        fit(x.(['curr' current_label{i} '_temp' temperature_label{i}]),...
        exp_data.([label '_' current_label{i} '_' temperature_label{ref_exp}])(:,2),'smoothingsp
end
```

Fit the discharge curves at different temperatures for reference current.

```
for i=find(ref_curr)
    fitObj.(['fit' temperature_label{i}]) = ...
        fit(x.(['curr' current_label{i} '_temp' temperature_label{i}]),...
        exp_data.([label '_' current_label{ref_exp} '_' temperature_label{i}])(:,2),'smoothingsp
end
```

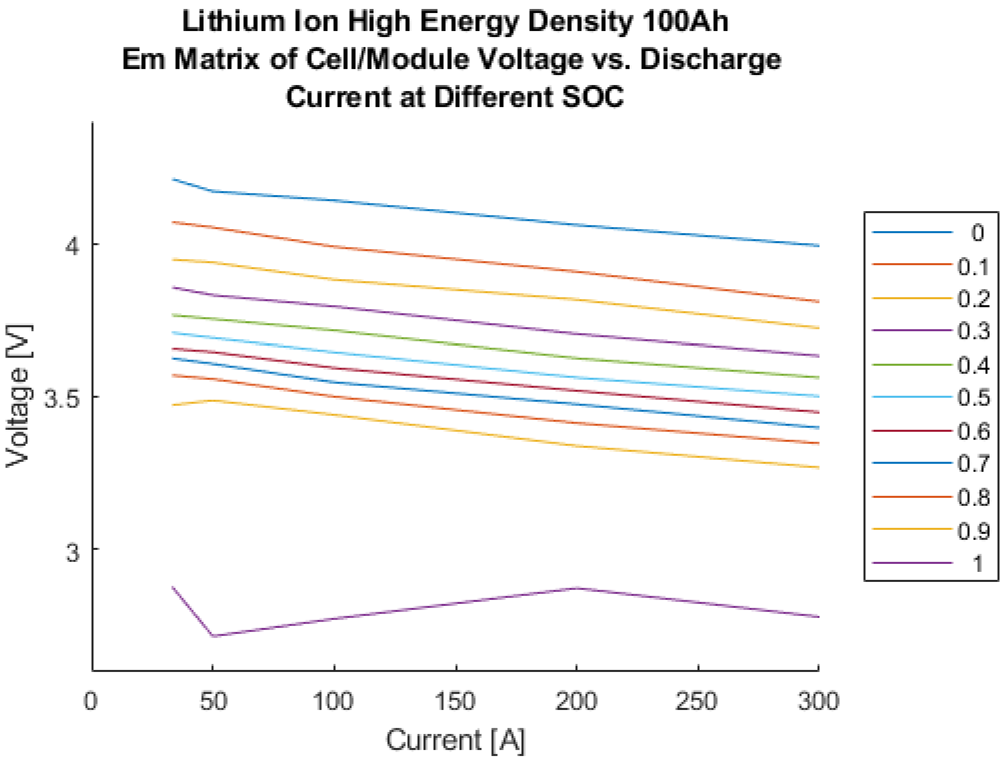
Construct the voltage versus discharge current for different SOC levels. Em_MAT is a matrix with the SOC in rows and the current in columns.

```
Em_MAT = [];
for i=find(ref_temp)
    Em_MAT = [Em_MAT fitObj.(['fit' current_label{i}])(SOC_LUT)];
end
```

Figure 3 shows the voltage versus current at different SOCs.

```
ex_datasheetbattery_plot_curves
```

Figure 3 - Curve Fitting



Step 4: Extrapolate Open Circuit Voltage

To obtain the open circuit voltage, `Em` , fit a line to the voltage versus current curve and extrapolate to `i=0` .

```
R0_refTemp = [];
for i=1:length(SOC_LUT)
    % Fit a line to V=f(I)
    fitSOC.(['SOC' num2str(i)]) = fit(sort_current',Em_MAT(i,sort_index_current)','poly1');
end
```

To estimate open circuit voltage, `Em` , at all SOC levels, extrapolate the values of voltage to `i=0` .

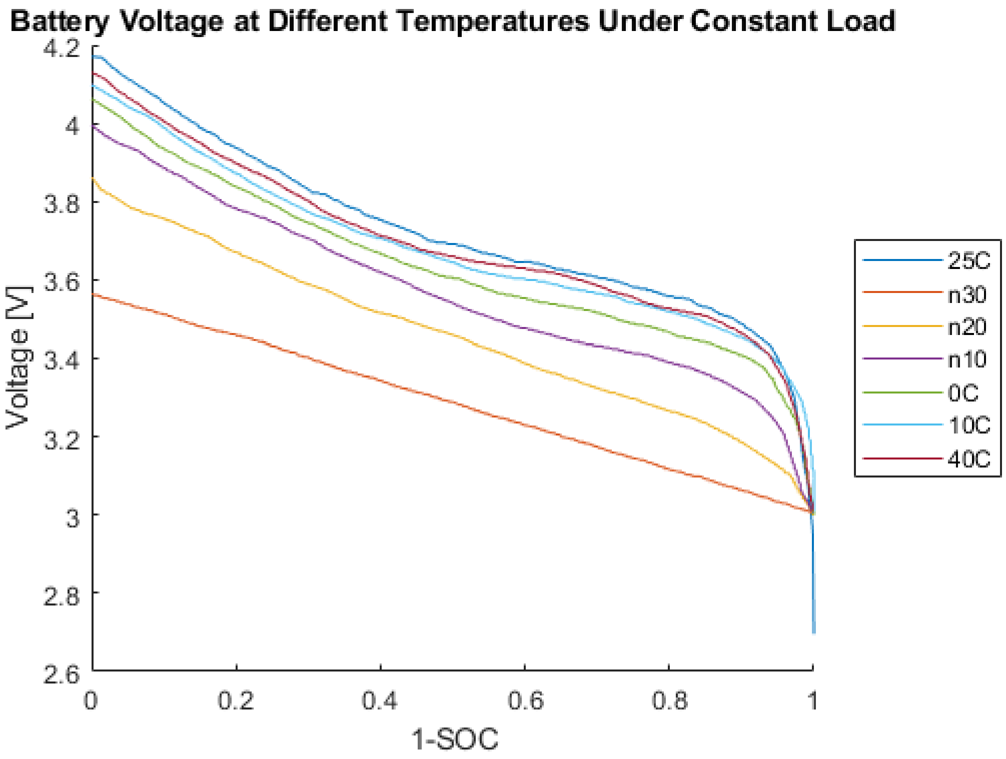
```
Em = [];
for i=1:length(SOC_LUT)
    % Em = f(0)
    Em = [Em fitSOC.(['SOC' num2str(i)])(0)];
end
Em = Em';
```

Step 5: Determine Battery Voltage and Resistance at Different Temperatures

Use the discharge and temperature data to determine the battery resistance as a function of current and SOC at varying temperatures. The validation data is not included. Figure 4 shows the battery voltage at different temperatures.

```
ex_datasheetbattery_plot_voltage
```

Figure 4 - Battery Voltage



Calculate the resistance at different temperatures using the reference current data set.

```
R0_LUT = [];
for i=find(ref_curr & ~val_exp)
    % Create fit object for V vs. SOC
    voltVsSOC.(['temp' temperature_label{i}]) = fitObj.(['fit' temperature_label{i}])(SOC_LUT);
    % Calculate R0(SOC,T) assuming linear behavior R0 = DeltaV / I
    R0.(['temp' temperature_label{i}]) = (Em - voltVsSOC.(['temp' temperature_label{i}]))./curren
    % Construct LUT
    R0_LUT = [R0_LUT R0.(['temp' temperature_label{i}])];
end
```

To avoid the abrupt R change close to `SOC=0` , extend R(0.9) all the way up to R(1). This is needed because of the way R is calculated. Make algorithm robust in case 0.9 is not an actual breakpoint

```
if ~isempty(find(SOC_LUT==0.9, 1))
    R0_LUT(SOC_LUT>0.9,:) = repmat(R0_LUT(SOC_LUT == 0.9,:),length(R0_LUT(SOC_LUT>0.9,:)),1);
else
    [closestTo0p9, locClosestTo0p9] = min(abs(SOC_LUT-0.9));
    R0_LUT(SOC_LUT>closestTo0p9,:) = repmat(R0_LUT(locClosestTo0p9,:),...
                                     length(R0_LUT(SOC_LUT>closestTo0p9,:)),1);
end
```

Determine the battery resistance at different temperatures.

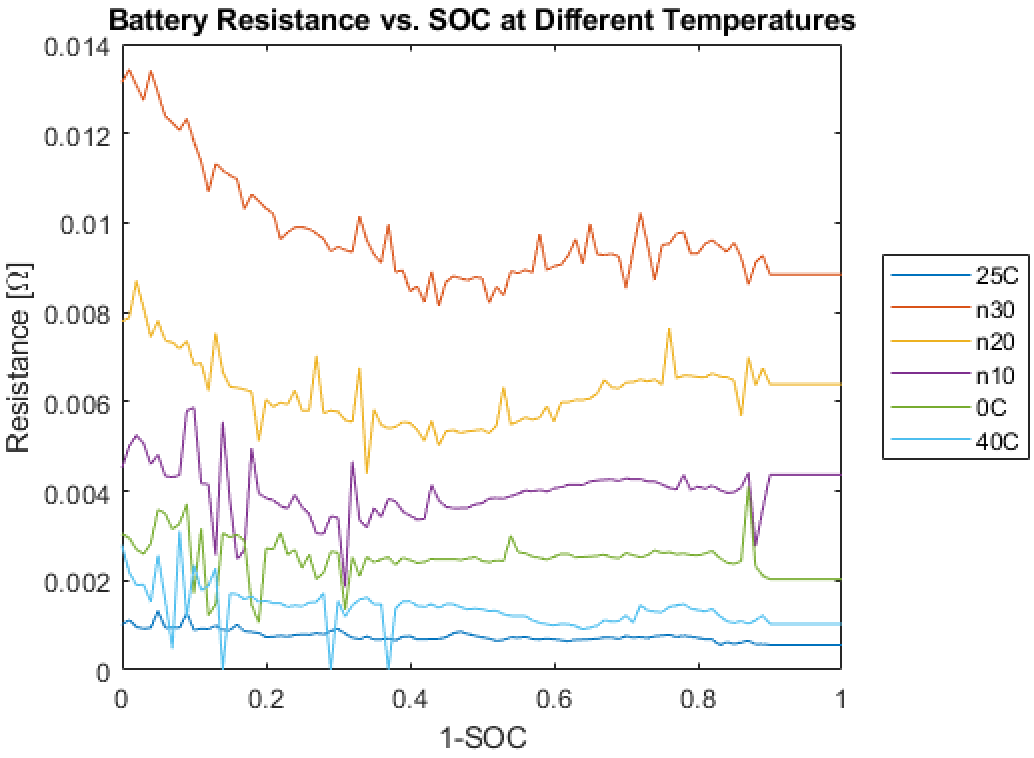
```
R0_LUT = max(R0_LUT,0);
T_LUT = 273.15 + temperature(ref_curr & ~val_exp);
[T_LUT1,idx] = sort(T_LUT);
```

```
xtmp=R0_LUT';
R0_LUT1(1:length(T_LUT),:) = xtmp(idx,:);
```

Figure 5 shows the battery resistance at different temperatures.

```
ex_datasheetbattery_plot_resistance
```

Figure 5 - Battery Resistance



Step 6: Compare to Arrhenius Behavior

Since the temperature-dependent reaction rate for the lithium-ion battery follows an Arrhenius behavior, you can use a comparison to validate the curve fit.

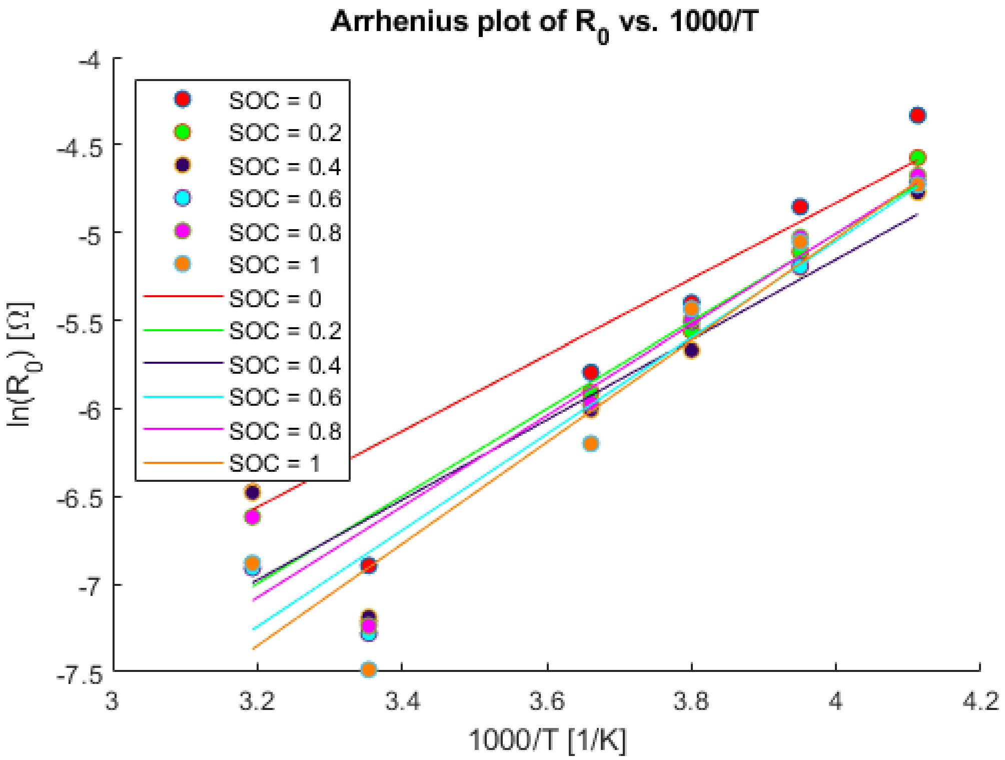
To determine the curve-fit prediction for the Arrhenius behavior, examine the activation energy, Ea . Obtain the activation energy via the slope of the internal resistance, Ro , versus 1000/T curve for different SOCs. The slope equals the activation energy, Ea , divided by the universal gas constant, Rg .

For a lithium-ion battery, a typical value of Ea is 20 kJ/mol[2]. Figure 6 indicates that the activation energy, Ea , obtained via the slope compares closely with 20 kJ/mol.

```
ex_datasheetbattery_plot_arrhenius
```

```
Activation energy for Li ion conduction
Ea = 17.9958       20.669      18.9557      22.8107      21.5289      24.0987 kJ/mol
Ea for electrolyte transport in Li ion battery = 20 kJ/mol
```

Figure 6 - Arrhenius Behavior



Step 7: Fit Battery Resistance

Fit the battery resistance to the validated temperature data as a function of SOC and temperature.

```
R0_LUT_bkpts = [];
counter = 1;
[SOC_LUT_index, ~] = find(abs(SOC_LUT-SOCbkpts)<0.001);

for i=find(ref_curr & ~val_exp)
    R0_LUT_bkpts = [R0_LUT_bkpts R0_LUT(SOC_LUT_index,counter)];
    counter = counter+1;
end

[xx,yy,zz] = prepareSurfaceData(1000./T_LUT,SOCbkpts,log(R0_LUT_bkpts));
[R0_vs_T_SOC_fit, gof] = fit([xx,yy],zz,'linearinterp');
% [R0_vs_T_SOC_fit, gof] = fit([xx,yy],zz,'poly12');
[xx1,yy1,zz1] = prepareSurfaceData(T_LUT,SOCbkpts,R0_LUT_bkpts);
[R0_vs_T_SOC_fit1, gof] = fit([xx1,yy1],zz1,'linearinterp');
```

Figures 7 and 8 show the surface plots of the battery resistance as a function of SOC and temperature.

```
ex_datasheetbattery_plot_surface
```

Figure 7 - Surface Fit of Battery Resistance

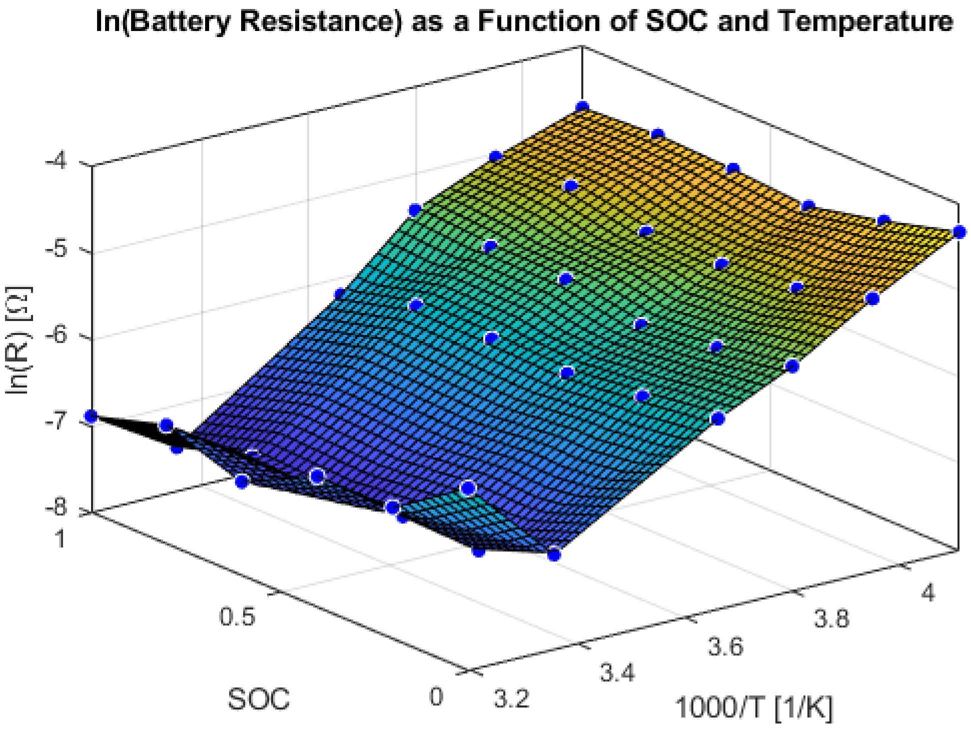
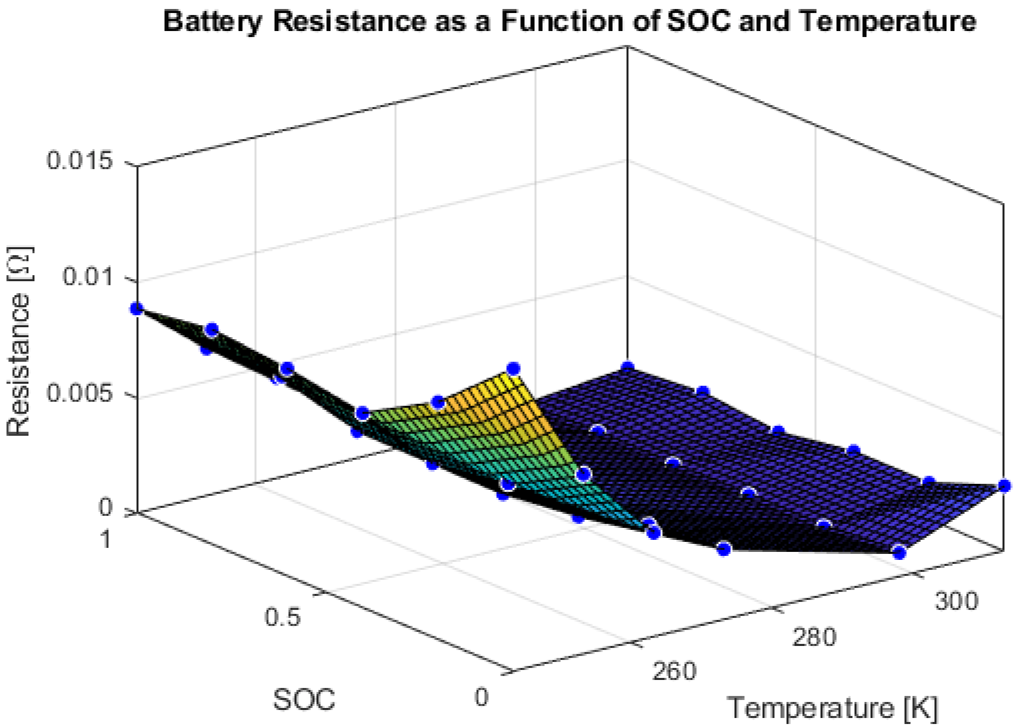


Figure 8 - Surface Fit of Battery Resistance

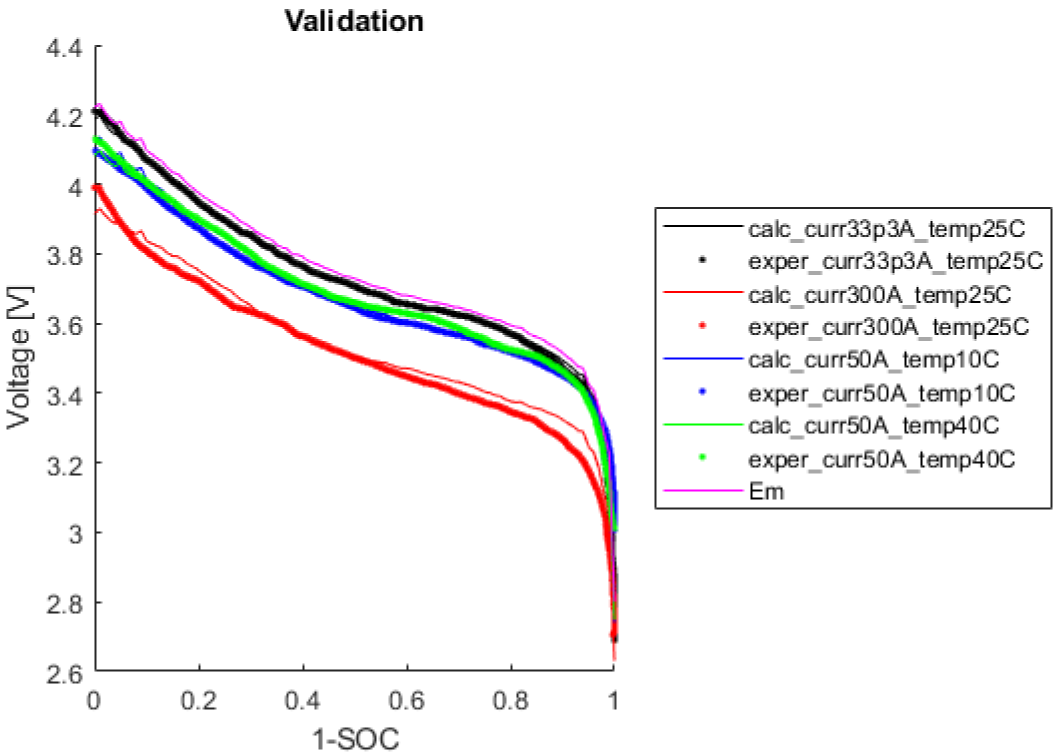


Step 8: Validate Battery Model Fit

Figure 9 shows the calculated data and the experimental data set data.

```
ex_datasheetbattery_plot_validation
```

Figure 9 - Validation of Battery Model Fit



Step 9: Set the Datasheet Battery Block Parameters

Set the **Rated capacity at nominal temperature** parameter to the capacity provided by the datasheet.

```
BattChargeMax = 100; % Ah Capacity from datasheet
```

Set the **Open circuit voltage table data** parameter to `Em`.

```
Em=flipud(Em);
```

Set the **Open circuit voltage breakpoints 1** parameter to the state of charge vector.

```
CapLUTBp=SOC_LUT;
```

Set the **Internal resistance table data** parameter to the fitted battery resistance data as a function of SOC and temperature.

```
RInt=R0_LUT_bkpts';
```

Set the **Battery temperature breakpoints 1** parameter to the temperature vector.

```
BattTempBp=T_LUT1;
```

Set the **Battery capacity breakpoints 2** parameter to the SOC vector.

```
CapSOCBp=SOCbkpts;
```

Set the **Initial battery charge** parameter to the value provided by the datasheet.

```
BattCapInit=100;
```

Clean up.

```
clear x xx xx1 yy yy1 zz zz1;
clear batt_id col correct cap count counter current;
clear correct_cap current_label data exp_data fitObj fitSOC gof;
clear i I idx indicot j k label leg line_colors;
clear indigo N orange p1 p2 purple ref_curr ref_exp ref_temp row colorV f9 p10 p9;
clear sort_current sort_index_current sort_index_temp sort_temp;
clear temperature temperature_lable V val_exp valIdx voltVsSOC xtmp temperature_label;
clear Ea Em_MAT markerType1 R0 R0_LUT R0_LUT1 R0_LUT_bkpts R0_refTemp R0_vs_T_fit;
clear T R R0_vs_T_SOC_fit R0_vs_T_SOC_fit1 SOC_LUT SOCbkpts T_LUT T_LUT1 SOC_LUT_index;
```

References

[1] Jackey, Robyn, Tarun Huria, Massimo Ceraolo, and Javier Gazzarri. "High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells." *IEEE International Electric Vehicle Conference*. March 2012, pp. 1-8.

[2] Ji, Yan, Yancheng Zhang, and Chao-Yang Wang. *Journal of the Electrochemical Society*. Volume 160, Issue 4 (2013), A636-A649.

See Also

Datasheet Battery | `Battery.MetaData` | `Battery.Parameters` | `Battery.PulseSequence` | `Battery.Pulse`

Generate Parameter Data for Equivalent Circuit Battery Block

Using MathWorks tools, estimation techniques, and measured lithium-ion or lead acid battery data, you can generate parameters for the Equivalent Circuit Battery block. The Equivalent Circuit Battery block implements a resistor-capacitor (RC) circuit battery with open circuit voltage, series resistance, and 1 through N RC pairs. The number of RC pairs reflects the number of time constants that characterize the battery transients. Typically, the number of RC pairs ranges from 1 through 5.

To create parameter data for the Equivalent Circuit Battery block, follow these workflow steps. The steps use numerical optimization techniques to determine the number of recommended RC pairs, provide initial estimates for the battery model circuit parameters, and estimate parameters to fit a model to experimental pulse discharge data. The results provide the open circuit voltage, series resistance, and RC pair parameter data for the Equivalent Circuit Battery block.

The workflow steps use this example script and models for a lithium-ion polymer (LiPo) battery:

- Estimate battery discharge script `Example_DischargePulseEstimation`.
- Model `BatteryEstim3RC_PTBS`.
- Model `BatteryEstim3RC_PTBS_EQ`.

The example battery discharge script uses a battery class to control the parameter estimation workflow.

Workflow	Description	Additional MathWorks Tooling
“Step 1: Load and Preprocess Data” on page 6-15	Load and preprocess time series battery discharge voltage and current data.	None
“Step 2: Determine the Number of RC Pairs” on page 6-17	Determine the number of necessary time constants (TC) for estimation.	Curve Fitting Toolbox
“Step 3: Estimate Parameters” on page 6-18	For battery discharge data, estimate and optimize: • Open-circuit voltage, Em • Series resistance, R0 • RC pair(s) time constant(s), Tau • RC pair(s) resistance(s), Rx Use a model that exercises the Estimation Equivalent Circuit Battery block.	Curve Fitting Toolbox, Parallel Computing Toolbox, Optimization Toolbox, and Simulink Design Optimization

Workflow	Description	Additional MathWorks Tooling
"Step 4: Set Equivalent Circuit Battery Block Parameters" on page 6-24	Set these block parameters: • **Open circuit voltage table data** • **Series resistance table data** • **State of charge breakpoints** • **Temperature breakpoints** • **Battery capacity table** • **Network resistance table data** • **Network capacitance table data**	None

Step 1: Load and Preprocess Data

Data Format and Requirements

The workflow supports pulse discharge sequences from 100% to 0% state-of-charge (SOC).

Data requirements include:

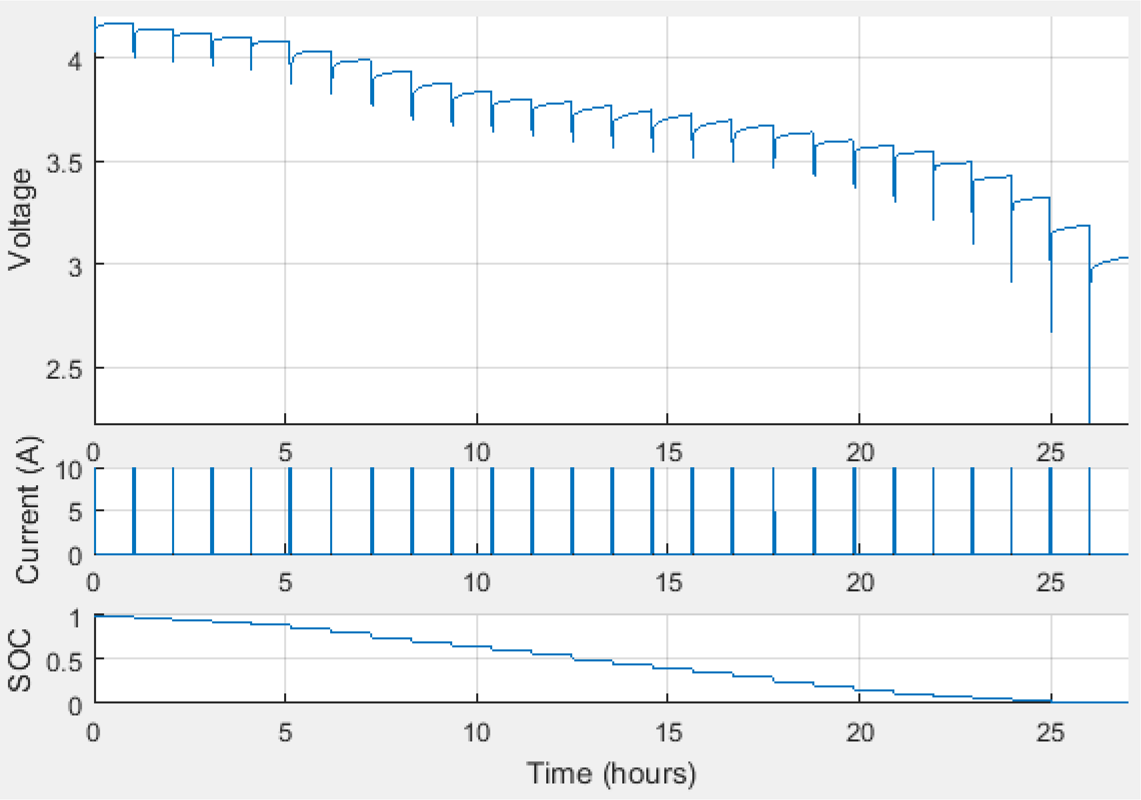
- Time series consisting of current and voltage from an experimental pulse discharge. For each experimental data set, the temperature is constant. The sample rate should be a minimum of 1 Hz, with an ideal rate at 10 Hz. This table summarizes the accuracy requirements.

Measurement	Accuracy	Ideal
Voltage	±5 mV	±1 mV
Current	±100 mA	±10 mA
Temperature	±1 °C	±1 °C

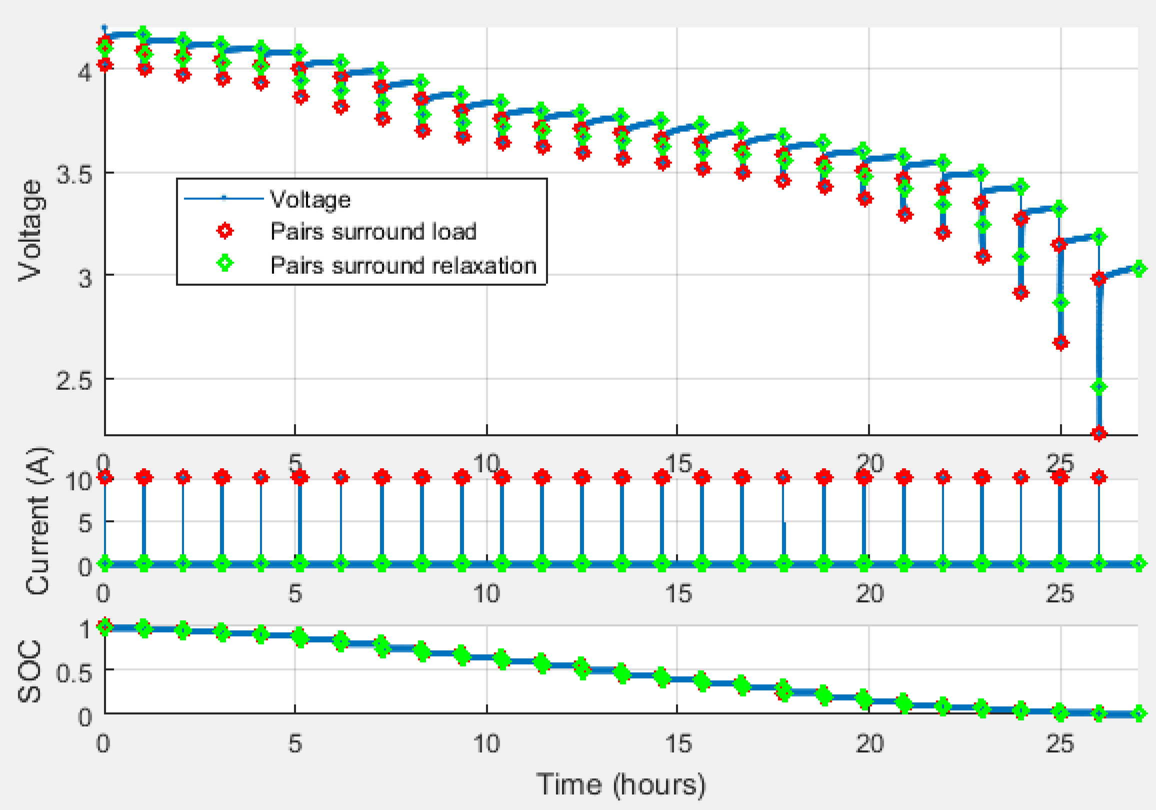
- Change in SOC for each pulse should not be greater than 5%.
- Data collection at high or low SOC might need modification to ensure safety.
- Sufficient relaxation time after each pulse to ensure battery approaches steady-state voltage.

Load and Preprocess Data

Load the battery time, voltage, and discharge data. Break up the data into `Battery.Pulse` objects. For example, load and preprocess the discharge data for a lithium-ion polymer (LiPo) battery using the `Step1: Load and Preprocess Data` commands in the `Example_DischargePulseEstimation` script.



Pulse Sequence



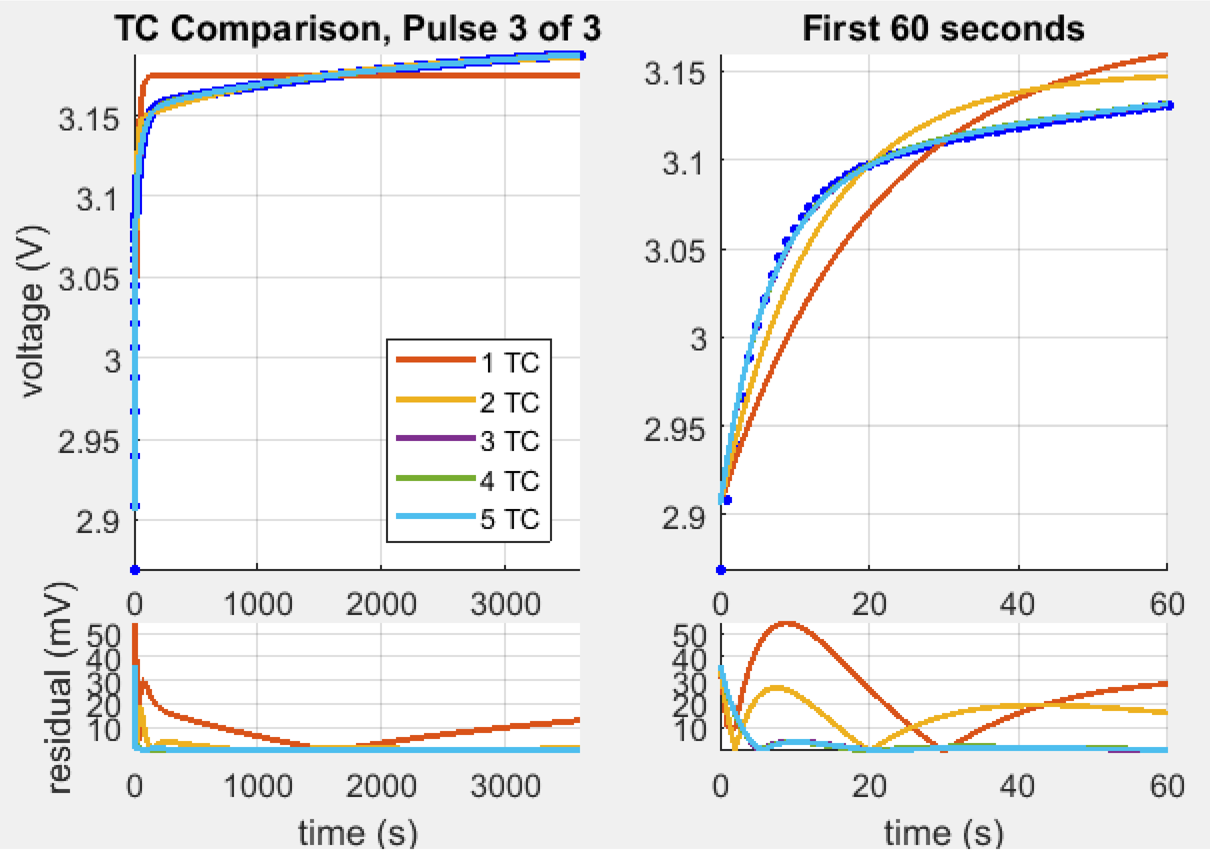
Pulse Identification

Step 2: Determine the Number of RC Pairs

Determine how many RC pairs to use in the model. You can investigate how many RC pairs to use by executing the `Step 2: Determine the Number of RC Pairs` commands in the `Example_DischargePulseEstimation` script. The example script uses the `BatteryEstim3RC_PTBS` model.

Compare Pulse Time Constants

Compare the time constants (TC) for each pulse. This example compares three pulses.



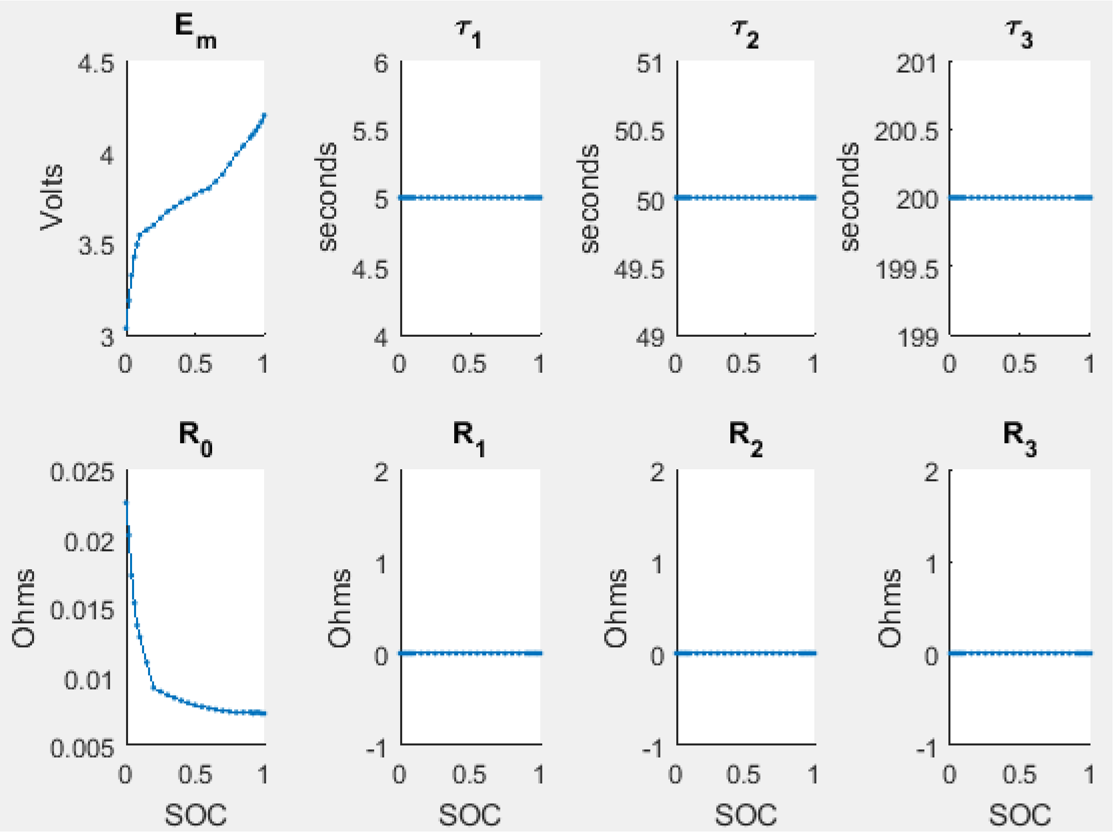
TC Comparison, Pulse 3 of 3

Step 3: Estimate Parameters

Estimate the parameters. You can investigate parameter estimation by executing the `Step 3: Estimate Parameters` commands in the `Example_DischargePulseEstimation` script.

Estimate Em and R0

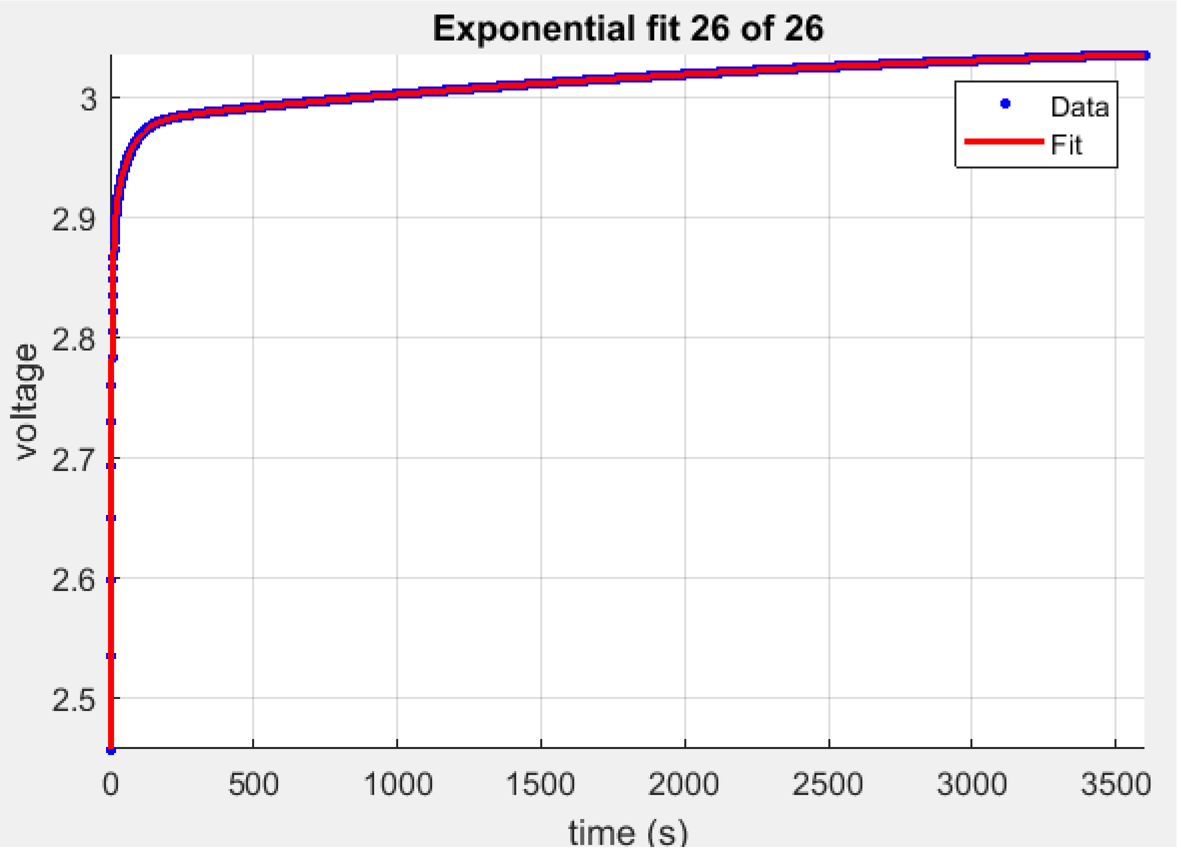
Inspect the voltage immediately before and after the current is applied and removed at the start and end of each pulse. The estimation technique uses the voltage for a raw calculation to estimate the open-circuit voltage (Em) and the series resistance (R0).



Parameter Tables

Estimate Tau

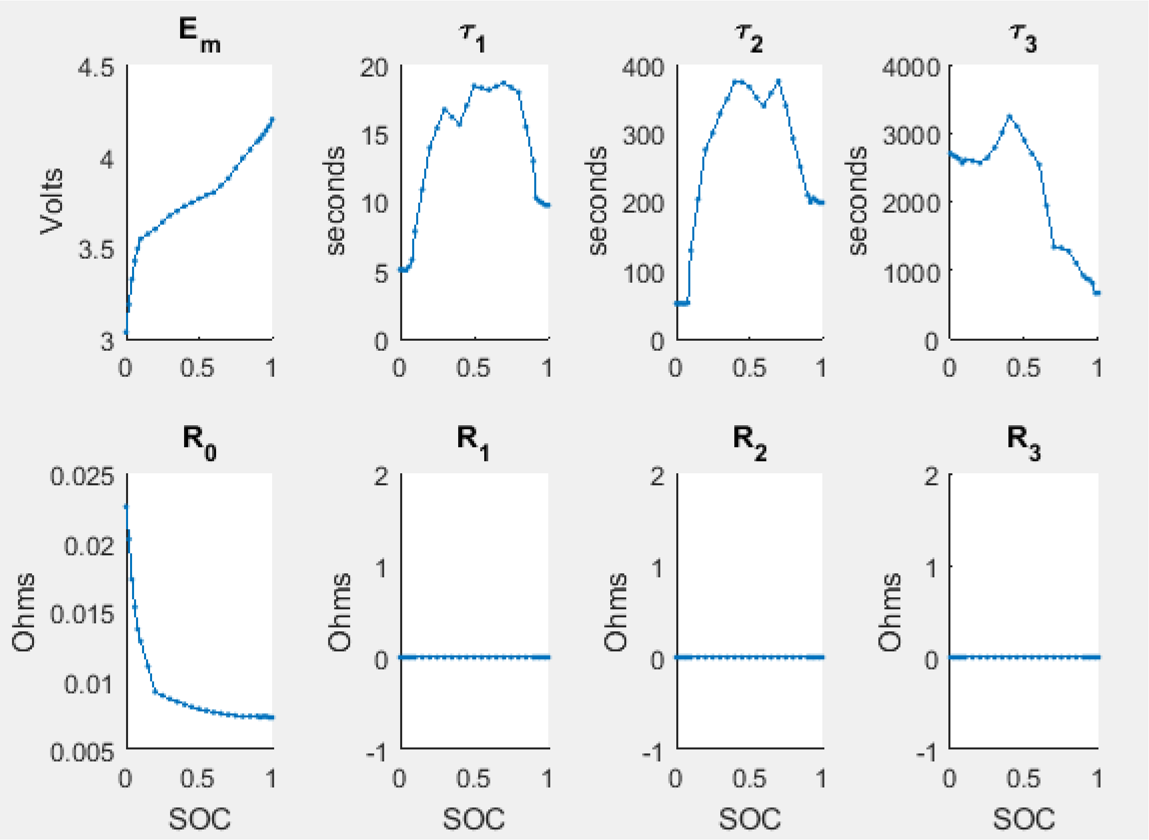
Use a curve-fitting technique on the pulse relaxation to estimate the RC time constant (Tau) at each SOC.



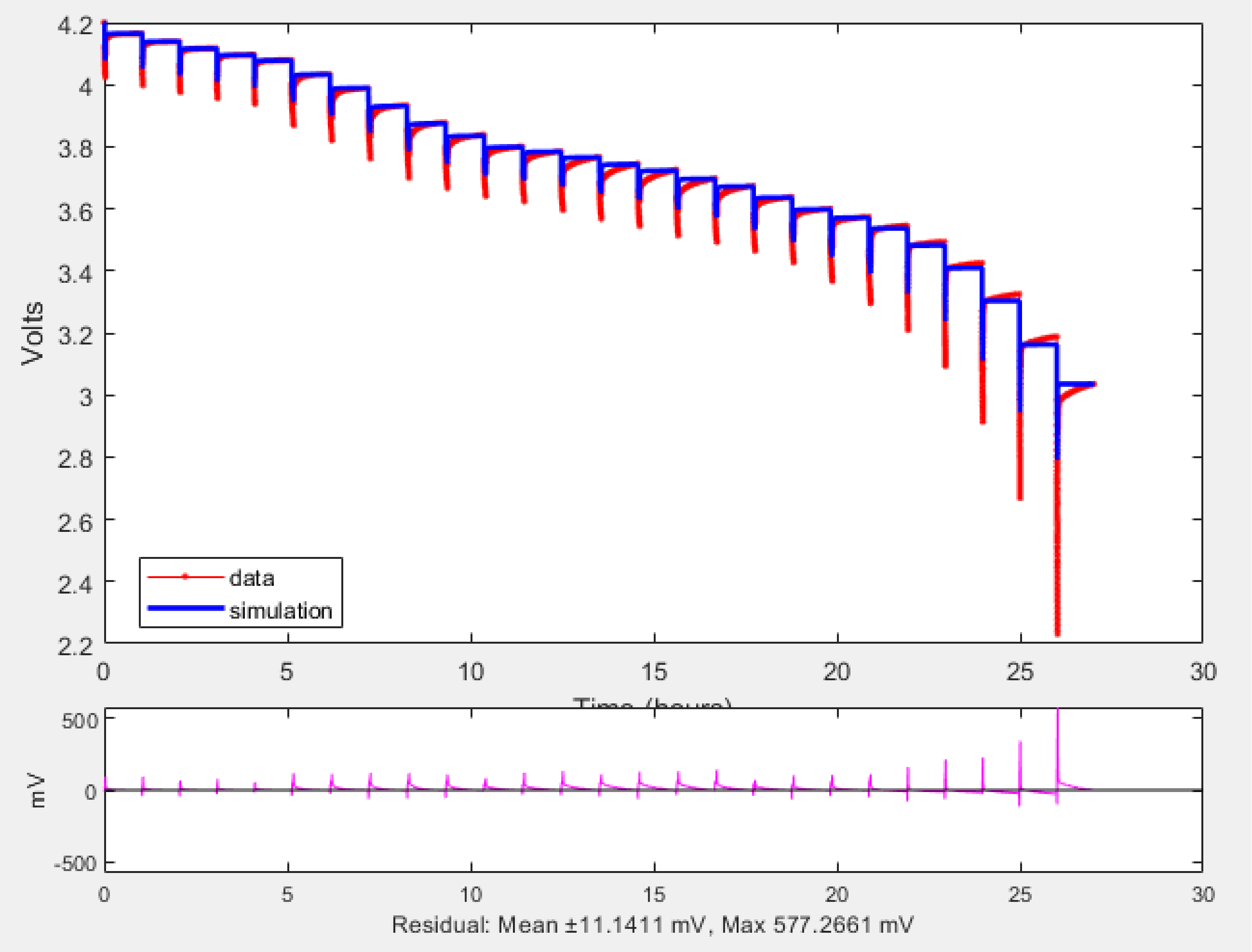
Relaxation Tau Fit

Plot Estimates

Plot the parameter and pulse sequence data and simulation comparison.



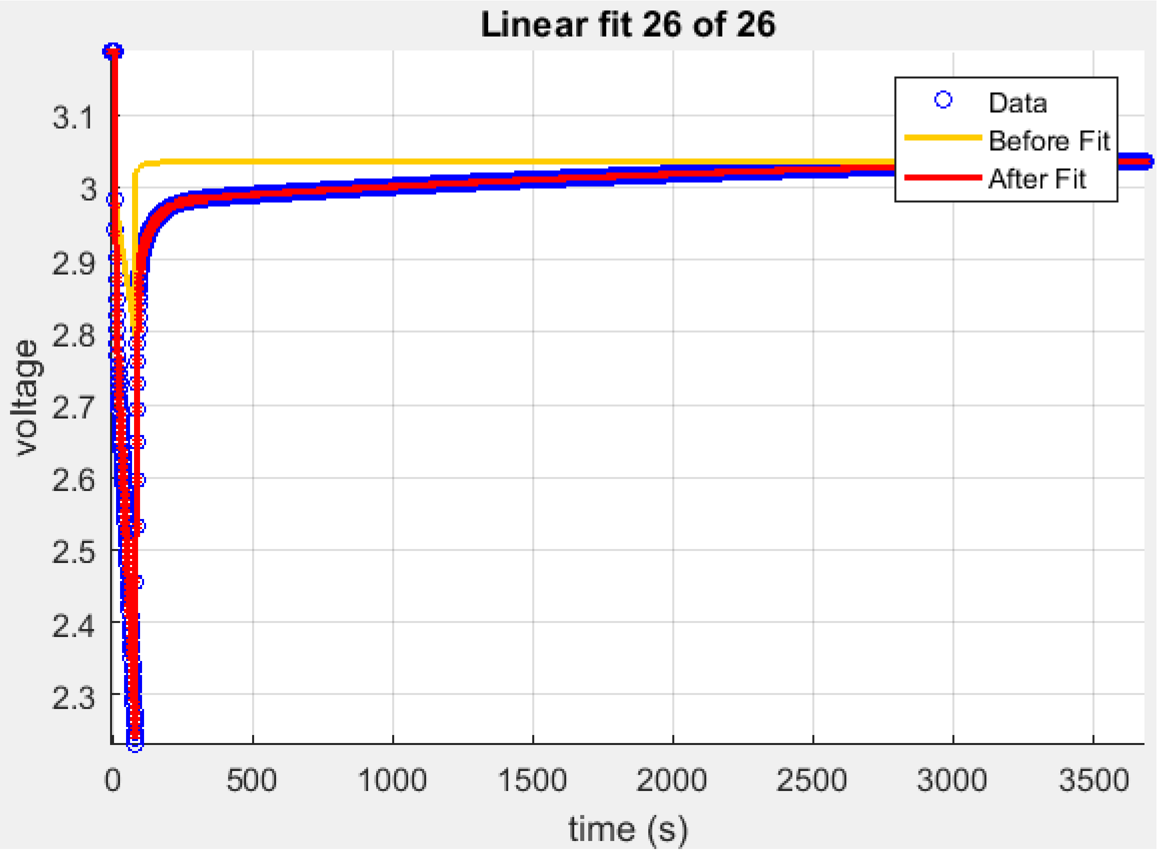
Parameter Tables



Pulse Sequence

Identify Parameters and Set Initial Values

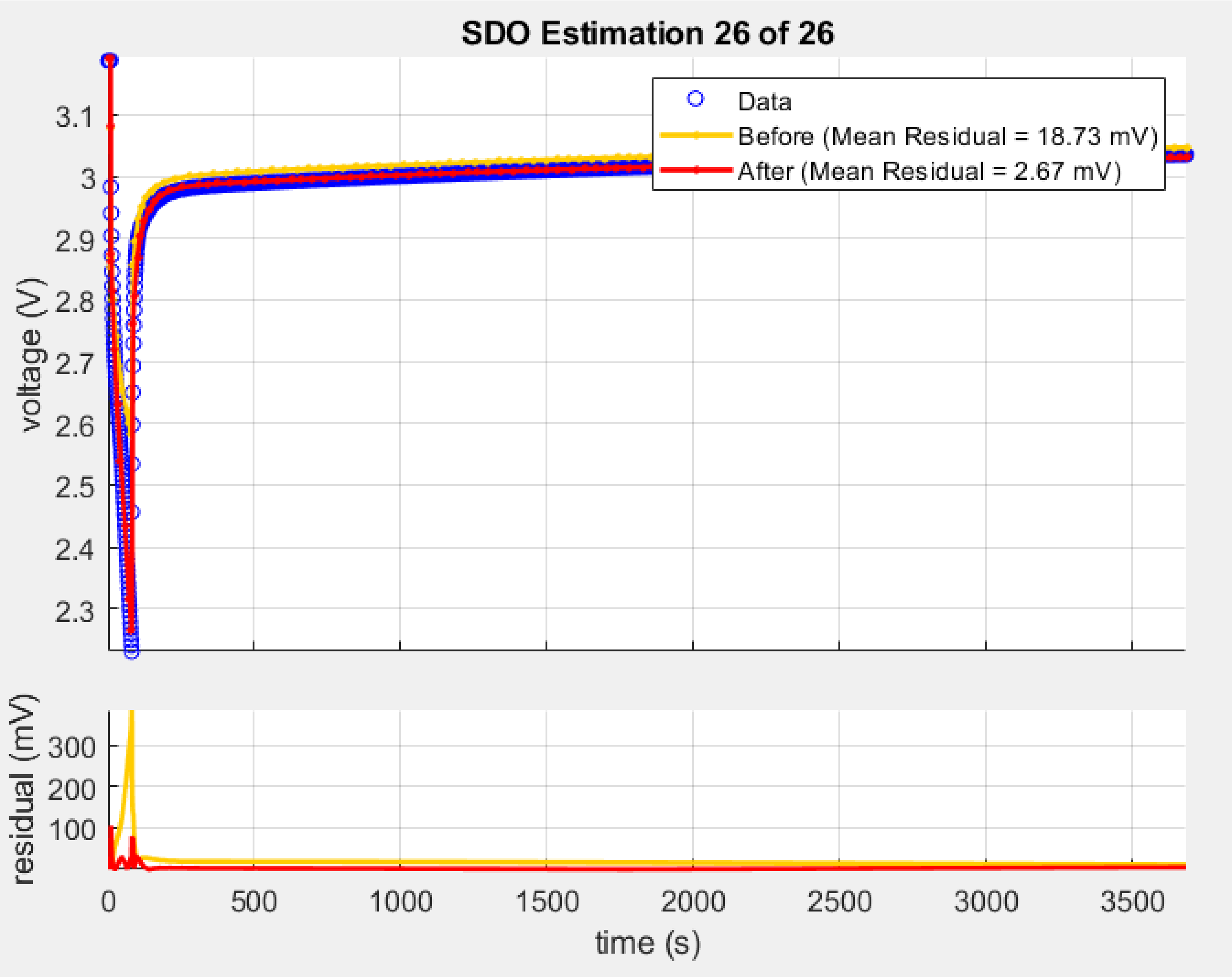
Identify parameters and set the initial values using a linear system approach, pulse-by-pulse.



Linear Fit

Optimize Estimates

Optimize the Em, R0, Rx, and Tau estimates using Simulink Design Optimization.



Pulse Identification

Step 4: Set Equivalent Circuit Battery Block Parameters

Set the Equivalent Circuit Battery block parameters to the values determined in step 3. To investigate setting the block parameters, execute the `Step 4: Set Equivalent Circuit Battery Block Parameters` commands in the `Example_DischargePulseEstimation` script. The experiment ran at two constant temperatures. There are three RC-pairs. The Equivalent Circuit Battery block parameter values are summarized in this table:

Parameter	Example Value
Number of series RC pairs	3
Open circuit voltage table data, EM	`EmPrime = repmat(Em,2,1)';`
Series resistance table data, R0	`R0Prime = repmat(R0,2,1)';`
State of charge breakpoints, SOC_BP	`SOC_LUTPrime = SOC_LUT;`

Parameter	Example Value
Temperature breakpoints, Temperature_BP	`TempPrime = [303 315.15];`
Battery capacity table	`CapacityAhPrime = [CapacityAh CapacityAh];`
Network resistance table data, R1	`R1Prime = repmat(Rx(1,:),2,1)';`
Network capacitance table data, C1	`C1Prime = repmat(Tx(1,:)./Rx(1,:),2,1)';`
Network resistance table data, R2	`R2Prime = repmat(Rx(2,:),2,1)';`
Network capacitance table data, C2	`C2Prime = repmat(Tx(2,:)./Rx(2,:),2,1)';`
Network resistance table data, R3	`R3Prime = repmat(Rx(3,:),2,1)';`
Network capacitance table data, C3	`C3Prime = repmat(Tx(3,:)./Rx(3,:),2,1)';`

References

[1] Ahmed, R., J. Gazzarri, R. Jackey, S. Onori, S. Habibi, et al. "Model-Based Parameter Identification of Healthy and Aged Li-ion Batteries for Electric Vehicle Applications." *SAE International Journal of Alternative Powertrains*. doi:10.4271/2015-01-0252, 4(2):2015.

[2] Gazzarri, J., N. Shrivastava, R. Jackey, and C. Borghesani. "Battery Pack Modeling, Simulation, and Deployment on a Multicore Real Time Target." *SAE International Journal of Aerospace*. doi:10.4271/2014-01-2217, 7(2):2014.

[3] Huria, T., M. Ceraolo, J. Gazzarri, and R. Jackey. "High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells." *IEEE International Electric Vehicle Conference*. March 2012, pp. 1–8.

[4] Huria, T., M. Ceraolo, J. Gazzarri, and R. Jackey. "Simplified Extended Kalman Filter Observer for SOC Estimation of Commercial Power-Oriented LFP Lithium Battery Cells." *SAE Technical Paper 2013-01-1544*. doi:10.4271/2013-01-1544, 2013.

[5] Jackey, R. "A Simple, Effective Lead-Acid Battery Modeling Process for Electrical System Component Selection." *SAE Technical Paper 2007-01-0778*. doi:10.4271/2007-01-0778, 2007.

[6] Jackey, R., G. Plett, and M. Klein. "Parameterization of a Battery Simulation Model Using Numerical Optimization Methods." *SAE Technical Paper 2009-01-1381*. doi:10.4271/2009-01-1381, 2009.

[7] Jackey, R., M. Saginaw, T. Huria, M. Ceraolo, P. Sanghvi, and J. Gazzarri. "Battery Model Parameter Estimation Using a Layered Technique: An Example Using a Lithium Iron Phosphate Cell." *SAE Technical Paper 2013-01-1547*. Warrendale, PA: SAE International, 2013.

See Also

Equivalent Circuit Battery | Estimation Equivalent Circuit Battery

Generate Optimal Current Controller Calibration Tables for Permanent Magnet Synchronous Motors

Use Model-Based Calibration Toolbox and Powertrain Blockset to generate optimized current controller and flux parameters for permanent magnet synchronous motor (PMSM) blocks. Follow these steps.

Step 1: Generate Current Controller Parameters

Use the Model-Based Calibration Toolbox to generate optimized current controller tables for flux-based motor controllers. Based on nonlinear motor flux data, the calibration tables optimize:

- Motor efficiency
- Maximum torque per ampere (MTPA)
- Flux weakening

You can use current controller parameters for the Flux-Based PM Controller block.

For the workflow, see “Generate Current Controller Parameters” on page 6-28.

Step 2: Generate Motor Parameters

Use MATLAB scripts available with Powertrain Blockset to load flux motor data, visualize the flux surface, and create plots of flux as a function of current.

You can use motor parameters for the Flux-Based PM Controller block.

For the workflow, see “Generate Feed-Forward Flux Parameters” on page 6-49.

Step 3: Generate Flux-Based PMSM Parameters

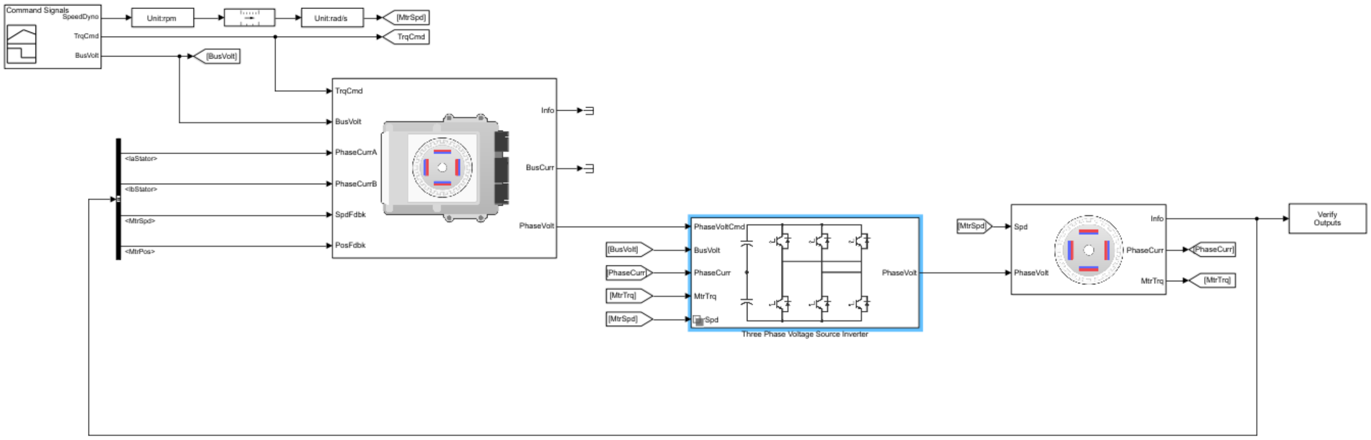
Use MATLAB scripts available with Powertrain Blockset to load flux motor data, invert the flux, and create plots of current as a function of flux.

You can use flux-based PMSM parameters for the Flux-Based PMSM block.

For the workflow, see “Generate Parameters for Flux-Based PMSM Block” on page 6-53.

Model with Optimized Parameters

To open a model with optimized parameters for the Flux-Based PM Controller and Flux-Based PMSM blocks, on the command line, type `Flux_Based_PMSM_TestBench`.



References

[1] Hu, Dakai, Yazan Alsmadi, and Longya Xu. “High fidelity nonlinear IPM modeling based on measured stator winding flux linkage.” *IEEE Transactions on Industry Applications*, Vol. 51, No. 4, July/August 2015.

[2] Chen, Xiao, Jiabin Wang, Bhaskar Sen, Panagiotis Lasari, Tianfu Sun. “A High-Fidelity and Computationally Efficient Model for Interior Permanent-Magnet Machines Considering the Magnetic Saturation, Spatial Harmonics, and Iron Loss Effect.” *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 7, July 2015.

[3] Ottosson, J., M. Alakula. “A compact field weakening controller implementation.” *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, July, 2006.

See Also

Flux-Based PM Controller | Flux-Based PMSM

Generate Current Controller Parameters

Using the Model-Based Calibration Toolbox, you can generate optimized current tables for flux-based motor controllers. Use the calibration tables for the Powertrain Blockset Flux-Based PM Controller current controller block parameters.

Based on nonlinear motor flux data, the calibration tables optimize:

- Motor efficiency
- Maximum torque per ampere (MTPA)
- Flux weakening

To generate optimized current tables, follow these workflow steps.

Workflow Steps	Description	MathWorks Tooling
"Collect and Post Process Motor Data" on page 6-29	Collect the nonlinear motor flux data from dynamometer testing or finite element analysis (FEA). For this example, file `PMSMEfficiencyData.xlsx` contains the data that you need: • Total flux, Ψ_{total}, in Wb • Allowed flux, Ψ_{max}, in Wb • *d*-axis flux, Ψ_d, in Wb • *q*-axis flux, Ψ_q, in Wb • *d*-axis current, I_d, in A • *q*-axis current, I_q, in A • Current magnitude, I_s, in A • Motor torque, T_e, in N·m • Motor speed, *n*, in rpm	N/A
"Model Motor Data" on page 6-30	Use a one-stage model to fit the data. Specifically: • Import data • Filter data • Fit model	Model-Based Calibration Toolbox

Workflow Steps	Description	MathWorks Tooling
"Generate Calibration" on page 6-34	Calibrate and optimize the data using objectives and constraints. Specifically: • Create functions. • Create tables from model. • Run an optimization. • Generate and fill optimized current controller calibration tables that are functions of motor torque and motor speed.	Model-Based Calibration Toolbox
"Set Block Parameters" on page 6-47	Use the optimized current controller calibration tables for the Flux-Based PM Controller block current controller parameters.	Powertrain Blockset

Collect and Post Process Motor Data

Collect this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA):

- *d*- and *q*- axis current
- *d*- and *q*- axis flux linkage
- Electromagnetic motor torque

Use the collected data and motor speed to calculate the total flux, maximum flux, and current magnitude:

$$\psi_{total} = \sqrt{\psi_d{}^2 + \psi_q{}^2}$$

$$i_s = \sqrt{i_d{}^2 + i_q{}^2}$$

$$n = \frac{60\omega_e}{2\pi P}$$

$$\psi_{max} = \frac{V_{dc}}{\sqrt{3}\omega_e}$$

The equations use these variables:

i_d, i_q	*d*- and *q*- axis current, respectively
i_s,	Current magnitude
Ψ_d, Ψ_q	*d*- and *q*- axis flux linkage, respectively
Ψ_{total}, Ψ_{max}	Total and allowed flux, respectively
ω_e	Electrical motor angular speed, rad/s
n	Motor speed, rpm
V_{dc}	Inverter bus voltage
P	Number of pole pairs

Finally, for each data point, create a file containing:

- Total flux, Ψ_{total}, in Wb
- Allowed flux, Ψ_{max}, in Wb
- *d*-axis flux, Ψ_d, in Wb
- *q*-axis flux, Ψ_q, in Wb
- *d*-axis current, I_d, in A
- *q*-axis current, I_q, in A
- Current magnitude, I_s, in A
- Motor torque, T_e, in N·m
- Motor speed, *n*, in rpm

For this example:

- Pole pairs, *P*, is 4
- Inverter bus voltage, V_{dc}, is 500

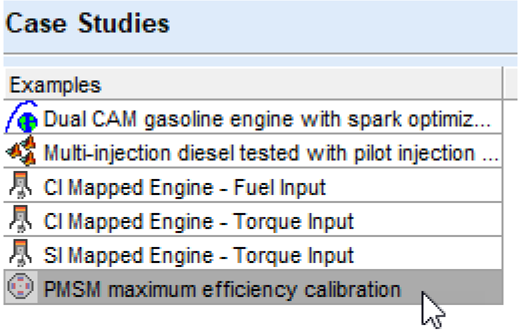
the data file `matlab\toolbox\mbc\mbctraining\PMSMEfficiencyData.xlsx` contains the motor flux data.

Model Motor Data

To model the motor data, use the **MBC Model Fitting** app to import, filter, and fit the data with a point-by-point model. For this example, the data file `PMSMEfficiencyData.xlsx` contains a large data set. You could consider using a design of experiment (DOE) to limit the data. However, the data set represents typical FEA analysis results.

Since there is a simple relationship between the *d*- and *q*-axis currents for fixed torque-speed operating points, the point-by-point model provides an accurate fit.

For comparison, the PMSM maximum efficiency calibration case study contains the model fit.


Case Studies
Examples
Dual CAM gasoline engine with spark optimiz...
Multi-injection diesel tested with pilot injection ...
CI Mapped Engine - Fuel Input
CI Mapped Engine - Torque Input
SI Mapped Engine - Torque Input
PMSM maximum efficiency calibration

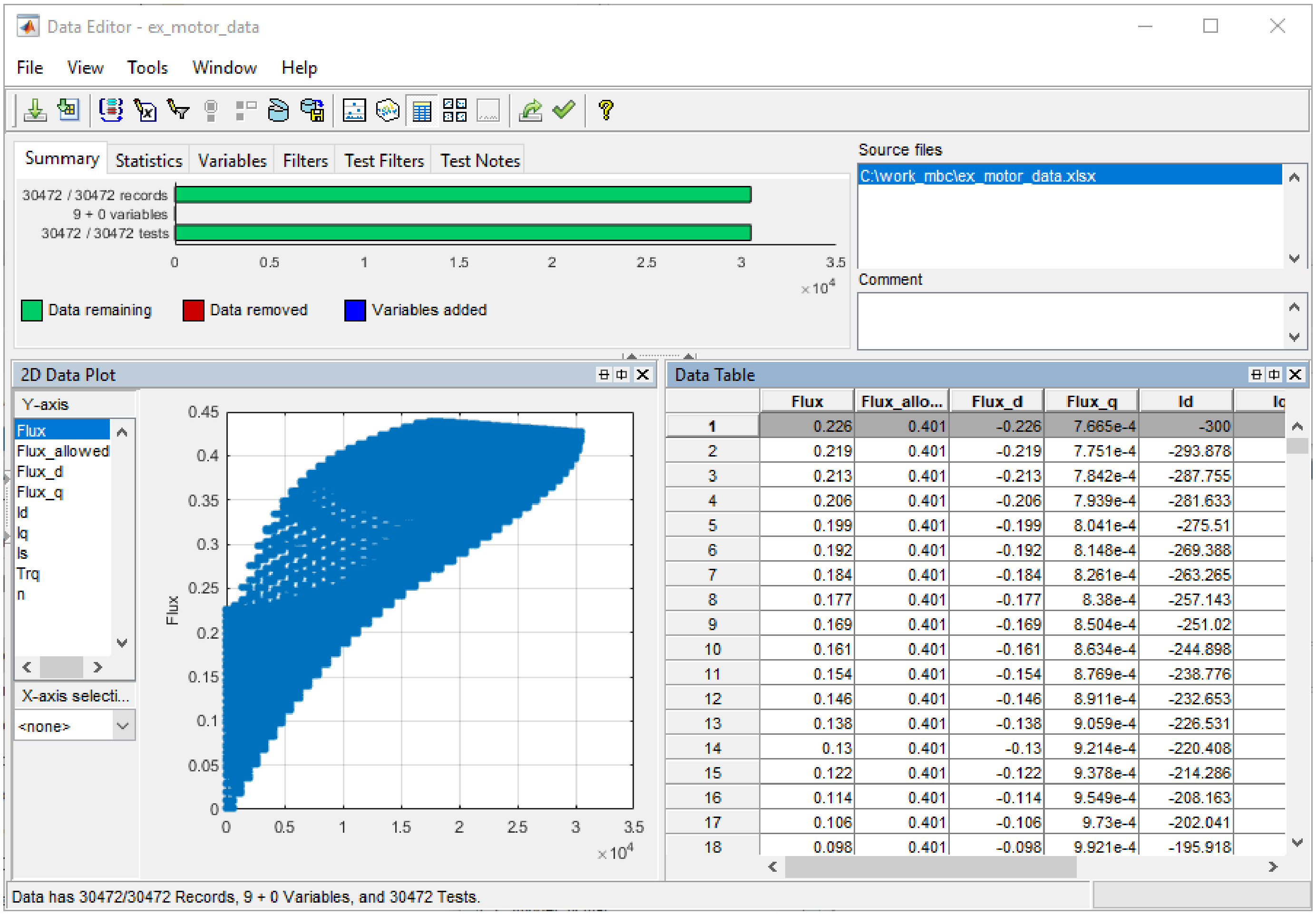
Import Data

For this example, `PMSMEfficiencyData.xlsx` contains this motor controller data:

- Total flux, Ψ_{total}, in Wb
- Allowed flux, Ψ_{max}, in Wb
- *d*-axis flux, Ψ_d, in Wb

- q-axis flux, Ψ_q, in Wb
- d-axis current, I_d, in A
- q-axis current, I_q, in A
- Current magnitude, I_s, in A
- Motor torque, T_e, in N·m
- Motor speed, n, in rpm

1. In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Model Fitting**.
2. In the Model Browser home page, click **Import Data**. Click **OK** to open a data source file.
3. Navigate to the `matlab\toolbox\mbc\mbctraining` folder. Open data file `PMSMEfficiencyData.xlsx`. The Data Editor opens with your data.



	Flux	Flux_allo...	Flux_d	Flux_q	Id	Ic
1	0.226	0.401	-0.226	7.665e-4	-300	
2	0.219	0.401	-0.219	7.751e-4	-293.878	
3	0.213	0.401	-0.213	7.842e-4	-287.755	
4	0.206	0.401	-0.206	7.939e-4	-281.633	
5	0.199	0.401	-0.199	8.041e-4	-275.51	
6	0.192	0.401	-0.192	8.148e-4	-269.388	
7	0.184	0.401	-0.184	8.261e-4	-263.265	
8	0.177	0.401	-0.177	8.38e-4	-257.143	
9	0.169	0.401	-0.169	8.504e-4	-251.02	
10	0.161	0.401	-0.161	8.634e-4	-244.898	
11	0.154	0.401	-0.154	8.769e-4	-238.776	
12	0.146	0.401	-0.146	8.911e-4	-232.653	
13	0.138	0.401	-0.138	9.059e-4	-226.531	
14	0.13	0.401	-0.13	9.214e-4	-220.408	
15	0.122	0.401	-0.122	9.378e-4	-214.286	
16	0.114	0.401	-0.114	9.549e-4	-208.163	
17	0.106	0.401	-0.106	9.73e-4	-202.041	
18	0.098	0.401	-0.098	9.921e-4	-195.918	

Data has 30472/30472 Records, 9 + 0 Variables, and 30472 Tests.

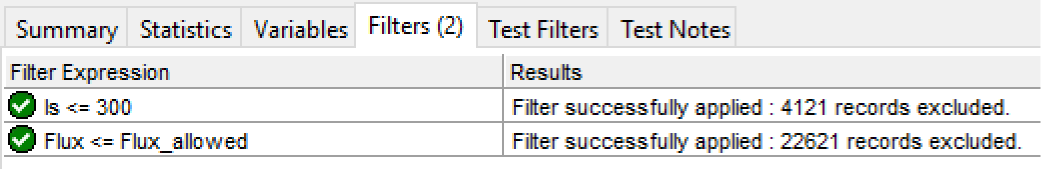
Filter Data

You can filter data to exclude records from the model fit. In this example, set up a filter to include only flux and current magnitudes that are less than a specified threshold. Specifically:

- Current magnitude, I_s, less than or equal to `300` A.
- Total flux, Ψ_{total}, less than or equal to allowed flux Ψ_{max}

1 In the Data Editor, select **Tools** > **Filters** to open the **Filter Editor**. Create these filters:

- `Is <= 300`
- `Flux <= Flux_allowed`



Filter Expression	Results
✓ Is <= 300	Filter successfully applied : 4121 records excluded.
✓ Flux <= Flux_allowed	Filter successfully applied : 22621 records excluded.

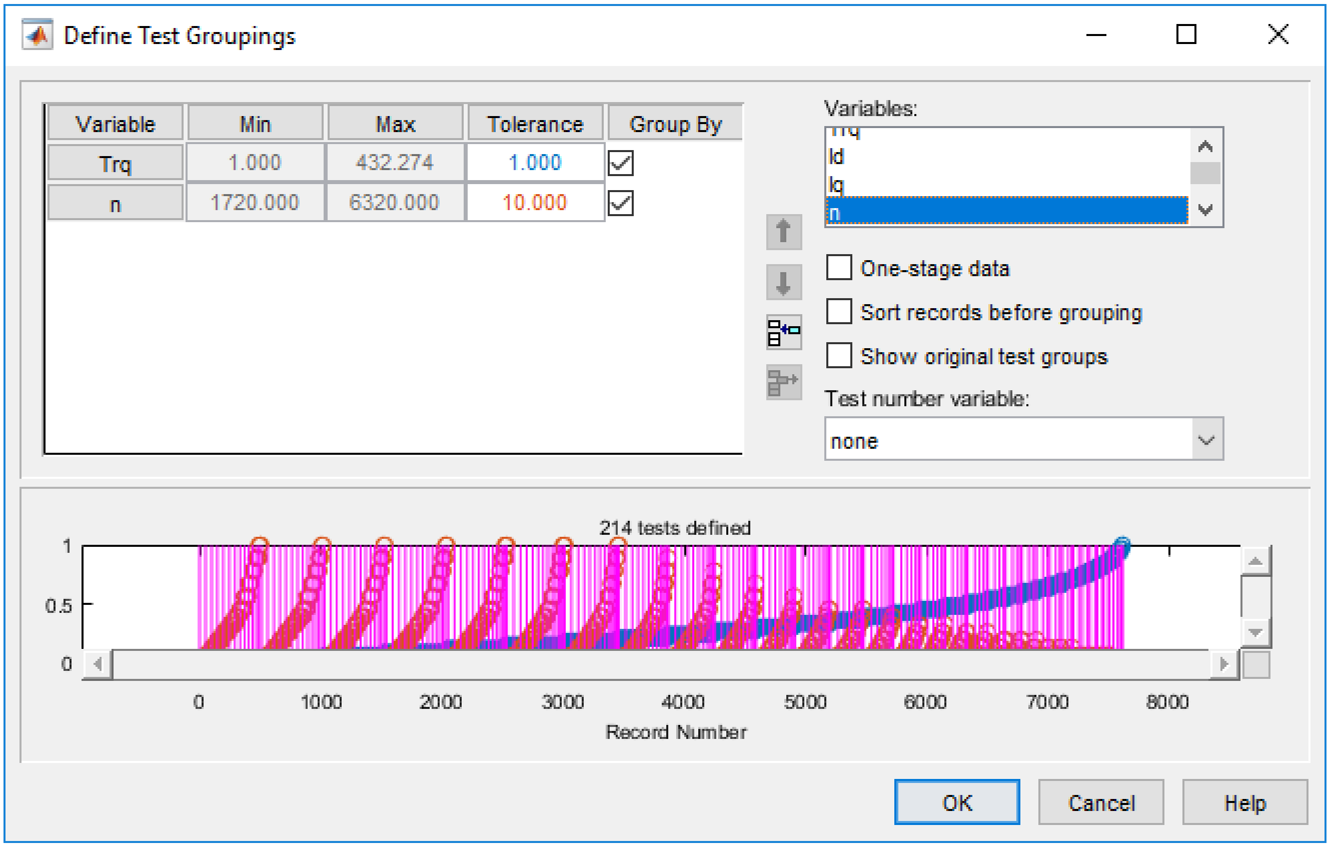
Define Test Groupings

For point-by-point models, you need to define test groups. In the example, define groups for motor torque and speed. Set the tolerances to so that Model-Based Calibration Toolbox groups small variations in torque and speed at the same operating point.

1 In the Data Editor, select **Tools** > **Test Groups** to open the **Define Test Groupings** dialog box. Create groups for the motor torque and speed.

2 Set these tolerances:

- Motor torque, Trq, to `1.000`
- Motor speed, n, to `10.000`



Define Test Groupings

Variable	Min	Max	Tolerance	Group By
Trq	1.000	432.274	1.000	☑
n	1720.000	6320.000	10.000	☑

Variables:
Trq
Id
Iq
n

□ One-stage data
□ Sort records before grouping
□ Show original test groups

Test number variable: none

3 In the Data Editor, select **File** > **Save & Close**. Accept the changes to the data.

Fit Model

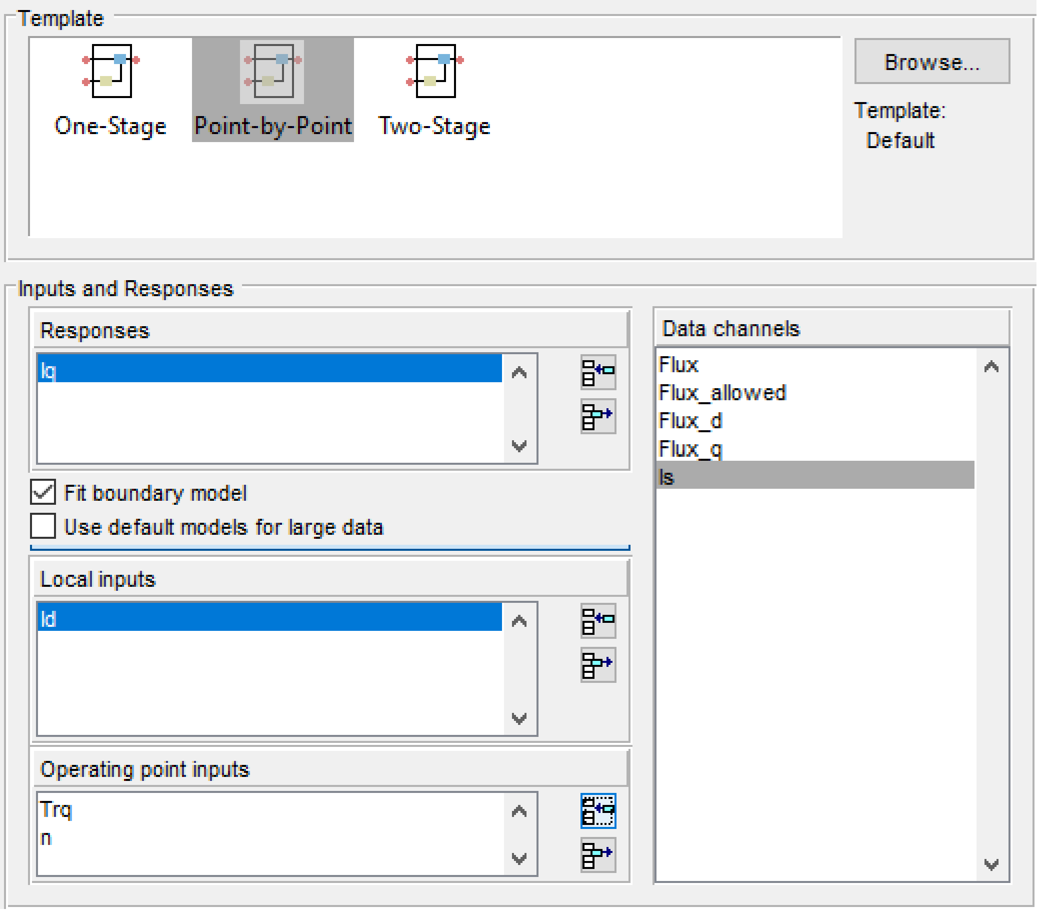
Fit the data to a point-by-point model with these responses, local inputs, and operating points:

- Responses
  - *q*-axis current, I_q, in A
- Local inputs
  - *d*-axis current, I_d, in A
- Operating points
  - Motor speed, *n*, in rpm
  - Electromagnetic motor torque, T_e, in N·m

1 In the Model Browser, select **Fit Models**.

2 In **Fit Models**, configure a Point-by-Point model with these responses and inputs.

Responses	Local Inputs	Operating Points
`Iq`	`Id`	`Trq` `n`



Template

One-Stage

Point-by-Point

Two-Stage

Browse...

Template: Default

Inputs and Responses

Responses

Iq

☑ Fit boundary model

□ Use default models for large data

Local inputs

Id

Operating point inputs

Trq

n

Data channels

Flux

Flux_allowed

Flux_d

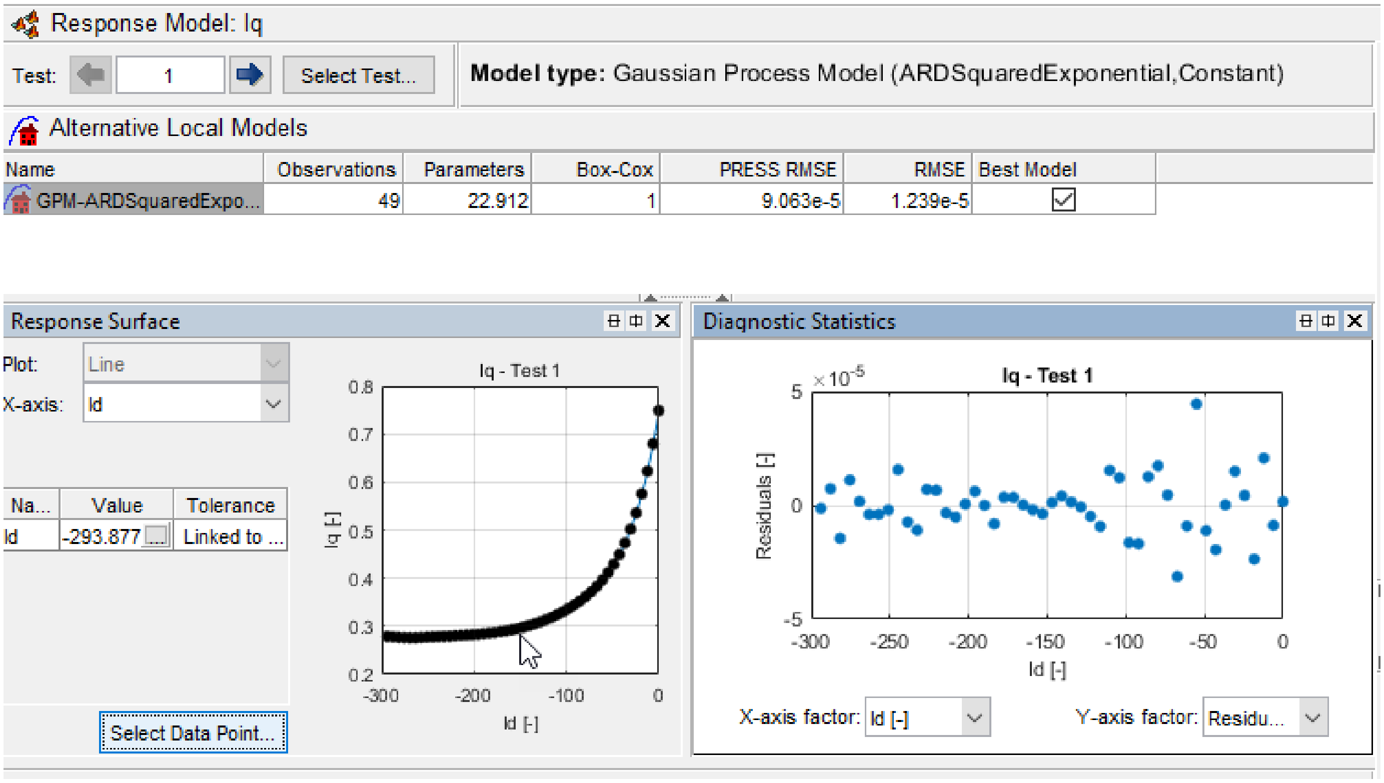
Flux_q

Is

3 To fit the model, select **OK**. If prompted, accept changes to data. By default, the fit uses a Gaussian Process Model (GPM) to fit the data.

4 After the fit completes, examine the response models for I_q. The Model Browser displays information that you can use to determine the accuracy of the model fit.

- In the Model Browser, select `Iq`. Examine the response surface and diagnostic statistics. These results indicate a reasonably accurate fit. You can browse through each test to examine the response for each torque-speed operating point.



Response Model: Iq

Test: 1 Select Test...

Model type: Gaussian Process Model (ARDSquaredExponential,Constant)

Alternative Local Models

Name	Observations	Parameters	Box-Cox	PRESS RMSE	RMSE	Best Model
GPM-ARDSquaredExpo...	49	22.912	1	9.063e-5	1.239e-5	☑

Response Surface

Plot: Line

X-axis: Id

Na...	Value	Tolerance
Id	-293.877	Linked to ...

Select Data Point...

Diagnostic Statistics

X-axis factor: Id [-]

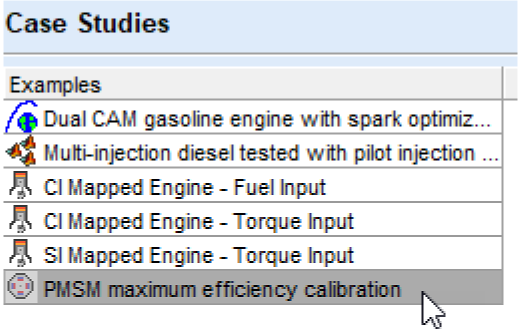
Y-axis factor: Residu...

5 Save your project. For example, select **Files** > **Save Project**. Save `gs_example.mat` to the `work` folder.

Generate Calibration

After you fit the model, create functions and tables, run the optimization, and fill the calibration tables.

For comparison, the PMSM maximum efficiency calibration case study contains the calibration results.



Case Studies

Examples

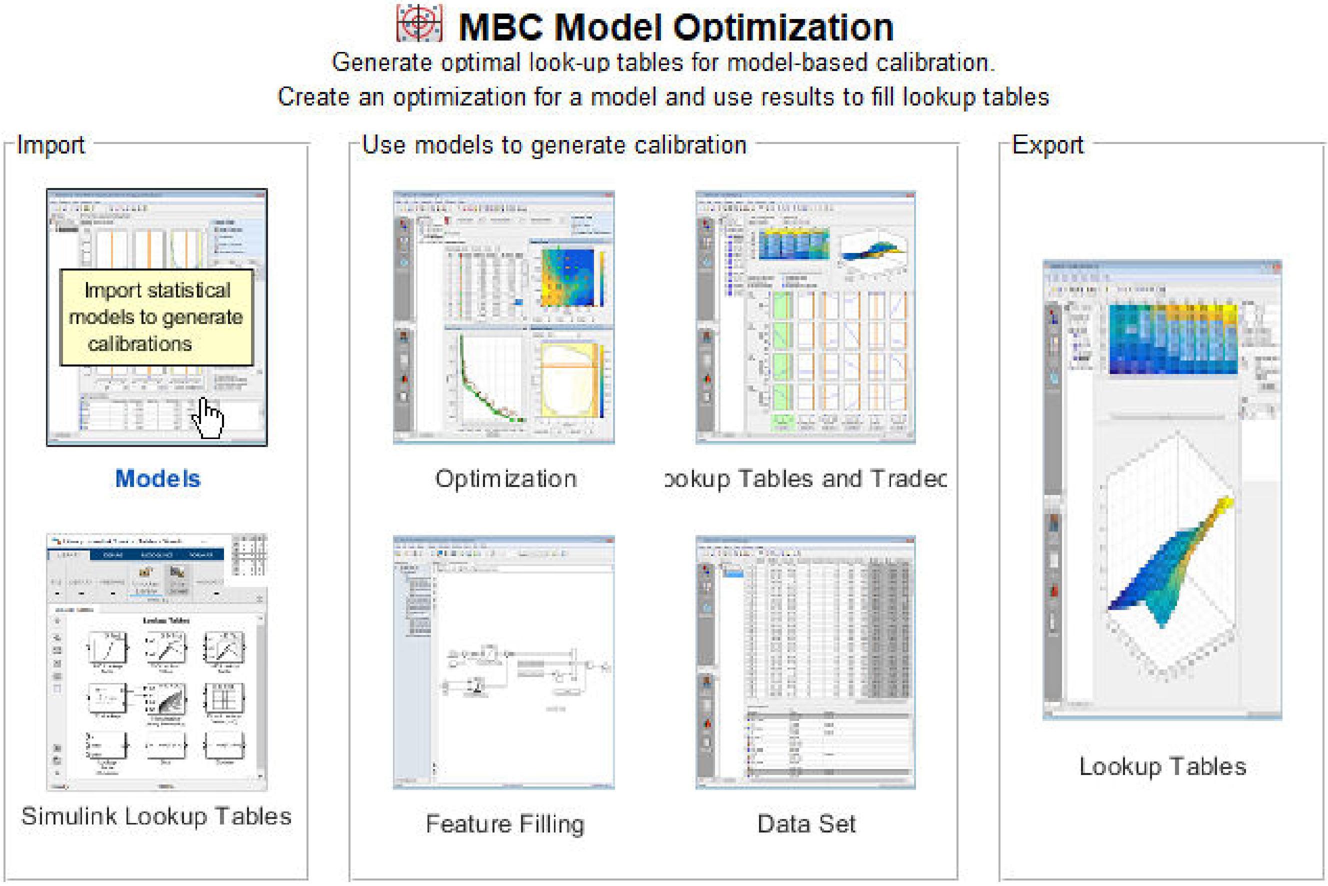
- Dual CAM gasoline engine with spark optimiz...
- Multi-injection diesel tested with pilot injection ...
- CI Mapped Engine - Fuel Input
- CI Mapped Engine - Torque Input
- SI Mapped Engine - Torque Input
- PMSM maximum efficiency calibration

Import Models and Create Functions

Import models and create the functions to use when you optimize the calibration. In this example, set up functions for:

- Current magnitude, I_s
- Torque per amp, *TPA*

1. In MATLAB, on the **Apps** tab, in the **Automotive** group, click **MBC Optimization**.
2. In the Cage Browser, select **Models**. If it is not already opened, in the MBC Model Fitting browser, open the `gs_example.mat` project.

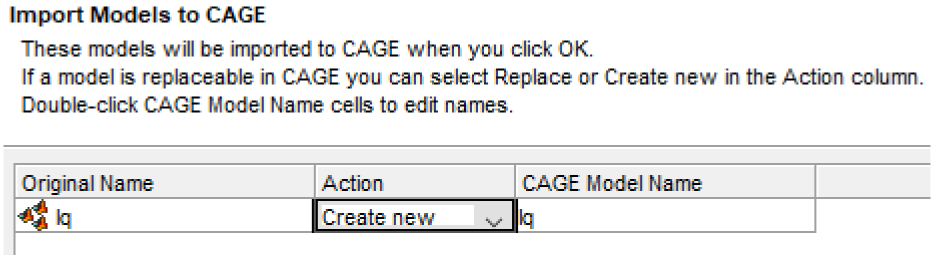


MBC Model Optimization

Generate optimal look-up tables for model-based calibration.

Create an optimization for a model and use results to fill lookup tables

3 In Import Models, click **OK**. Close the CAGE Import Tool.



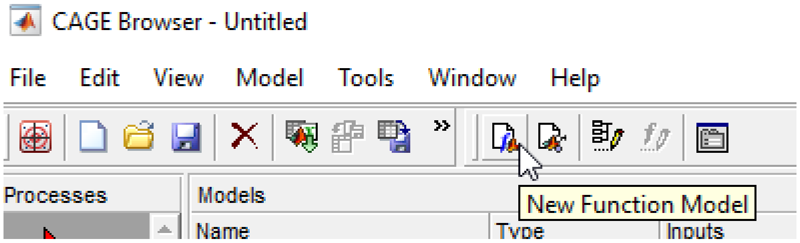
Import Models to CAGE

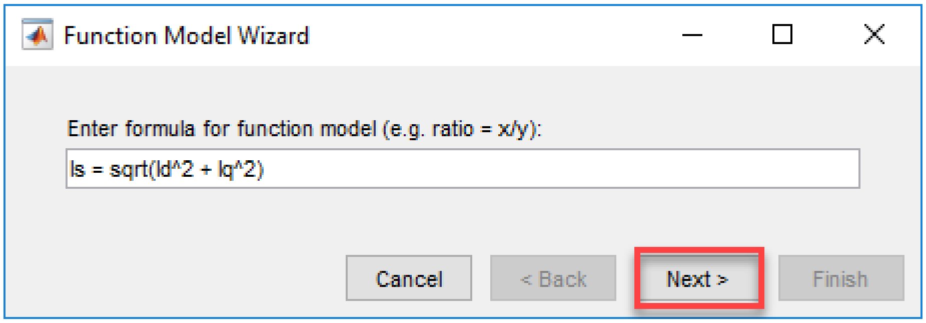
These models will be imported to CAGE when you click OK.
If a model is replaceable in CAGE you can select Replace or Create new in the Action column.
Double-click CAGE Model Name cells to edit names.

Original Name	Action	CAGE Model Name
Iq	Create new	Iq

4 In the Cage Browser toolbar, use **New Function Model** wizard to create these functions:

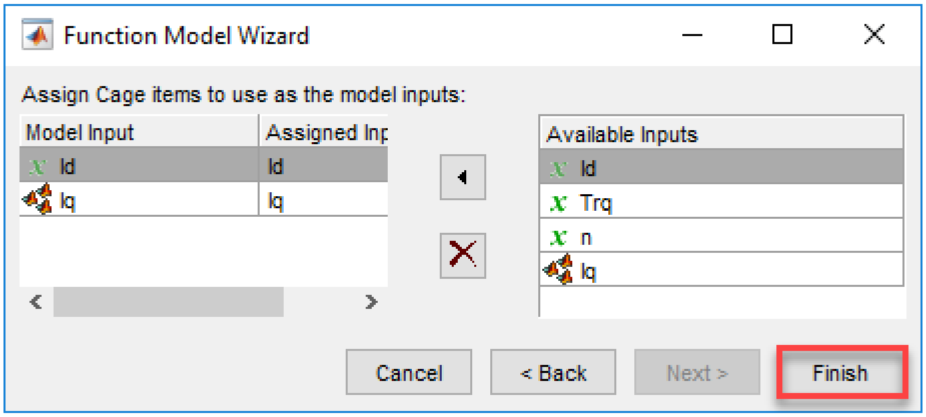
- `Is = sqrt(Id^2 + Iq^2)`
- `TPA = Trq/Is`





Enter formula for function model (e.g. ratio = x/y):

Is = sqrt(Id^2 + Iq^2)

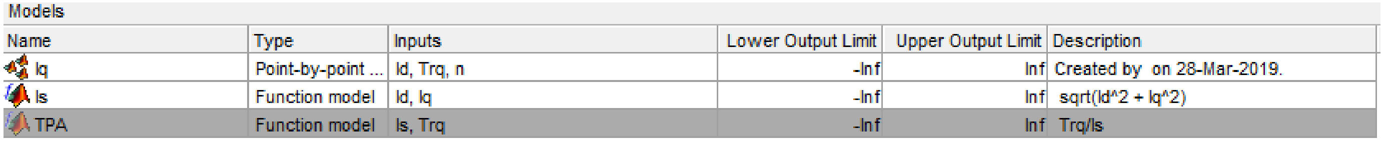


Assign Cage items to use as the model inputs:

Model Input	Assigned Inp
Id	Id
Iq	Iq

Available Inputs
Id
Trq
n
Iq

5 In the Cage Browser, verify that the function models for `Is` and `TPA` have these descriptions.



Models

Name	Type	Inputs	Lower Output Limit	Upper Output Limit	Description
Iq	Point-by-point ...	Id, Trq, n	-Inf	Inf	Created by on 28-Mar-2019.
Is	Function model	Id, Iq	-Inf	Inf	sqrt(Id^2 + Iq^2)
TPA	Function model	Is, Trq	-Inf	Inf	Trq/Is

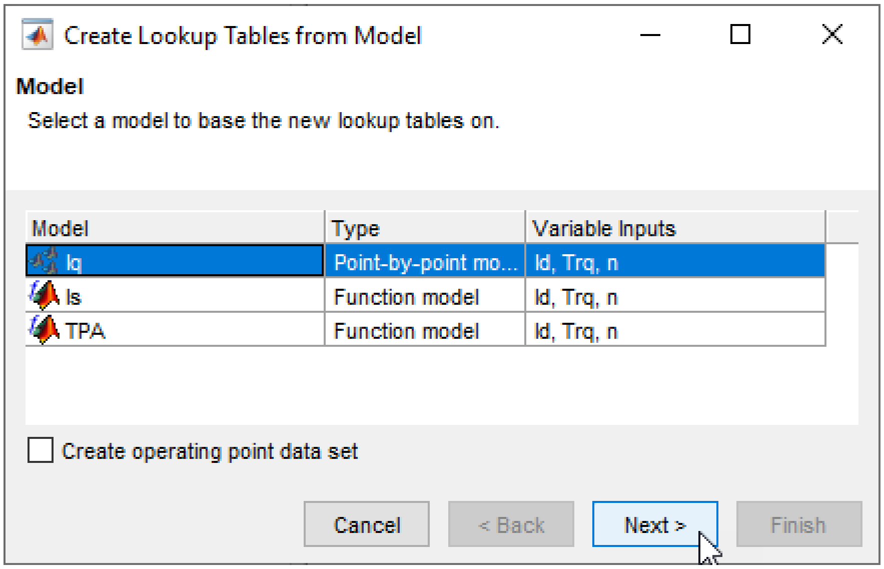
6 Select **File** > **Save Project**. Save `gs_example.cag` to the `work` folder.

Create Lookup Tables from Model

Create tables that the Model-Based Calibration Toolbox optimizers uses to store the optimized parameters. For this example, the tables are:

- *d*-axis current, I_d, as a function of motor torque, `Trq`, and motor speed, `n`.
- *q*-axis current, I_q, as a function of motor torque, `Trq`, and motor speed, `n`.

1 In the Cage Browser, select **Lookup Tables and Tradeoff**. In Create Lookup Tables from Model, select `Iq`. Click **Next**.



Model

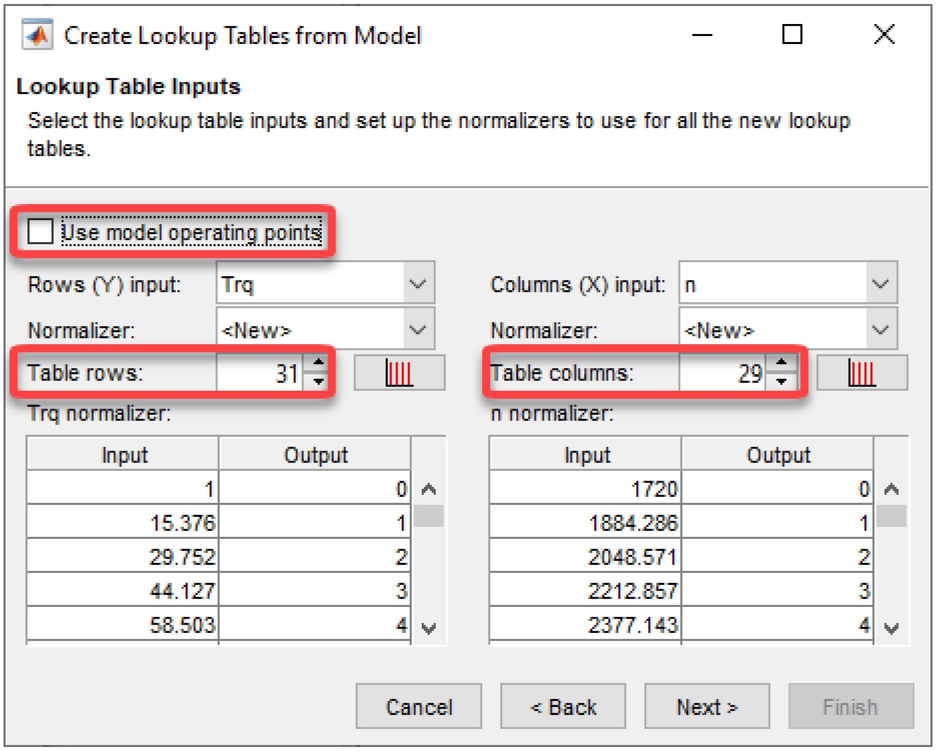
Select a model to base the new lookup tables on.

Model	Type	Variable Inputs
Iq	Point-by-point mo...	Id, Trq, n
Is	Function model	Id, Trq, n
TPA	Function model	Id, Trq, n

□ Create operating point data set

2 In the Create Lookup Tables from Model wizard:

- Clear **Use model operating points**.
- Set **Table rows** to 31.
- Set **Table columns** to 29.
- Click **Next**.



Create Lookup Tables from Model

Lookup Table Inputs

Select the lookup table inputs and set up the normalizers to use for all the new lookup tables.

□ Use model operating points

Rows (Y) input: Trq

Normalizer: <New>

Table rows: 31

Trq normalizer:

Input	Output
1	0
15.376	1
29.752	2
44.127	3
58.503	4

Columns (X) input: n

Normalizer: <New>

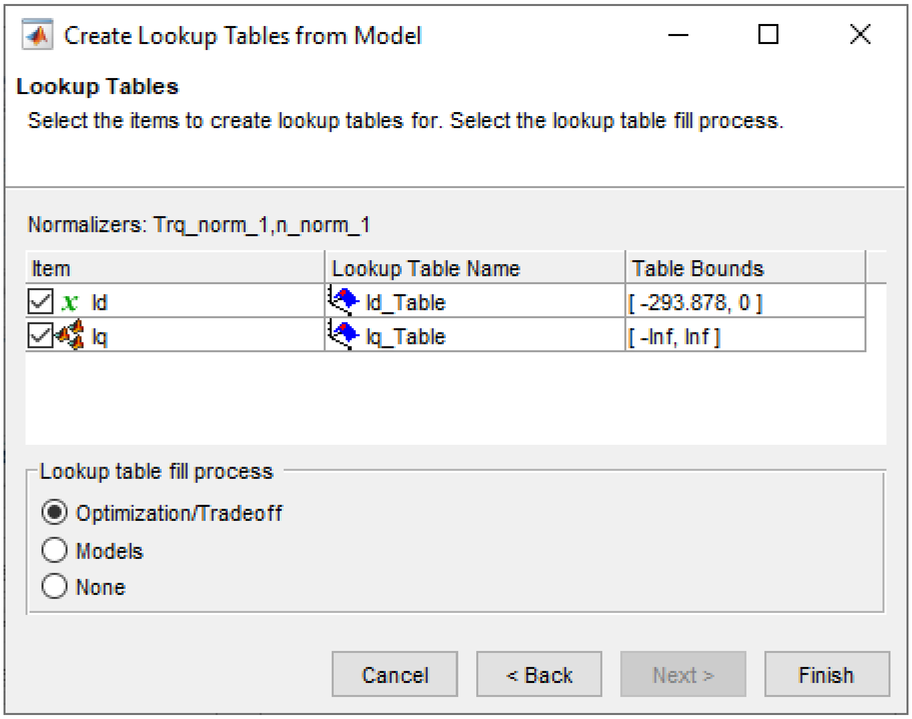
Table columns: 29

n normalizer:

Input	Output
1720	0
1884.286	1
2048.571	2
2212.857	3
2377.143	4

3 In Create Lookup Tables from Model:

- Select `Id` and `Iq`.
- Click **Finish**.



Create Lookup Tables from Model

Lookup Tables

Select the items to create lookup tables for. Select the lookup table fill process.

Normalizers: Trq_norm_1,n_norm_1

Item	Lookup Table Name	Table Bounds
☑ Id	Id_Table	[-293.878, 0]
☑ Iq	Iq_Table	[-Inf, Inf]

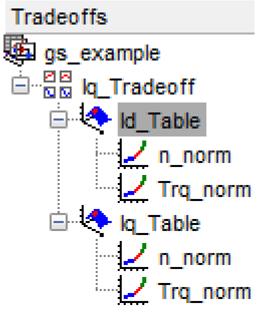
Lookup table fill process

● Optimization/Tradeoff

○ Models

○ None

4 In the CAGE Browser, examine the tables.

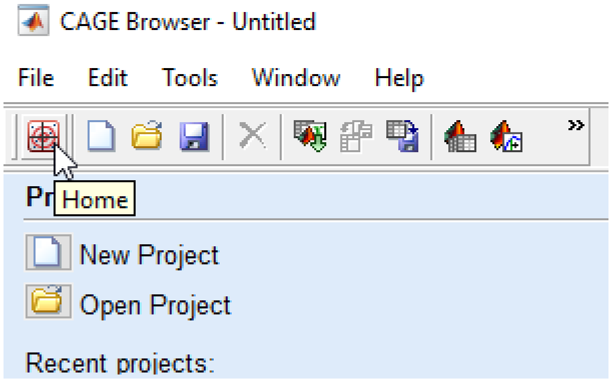


Run Optimization

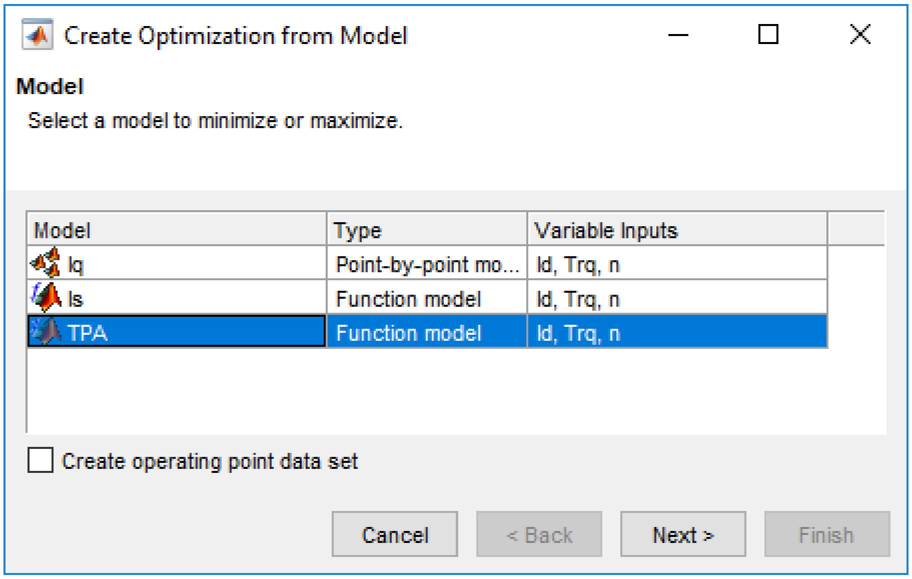
In this example, run an optimization with these specifications:

- Current magnitude, I_s, less than or equal to `300` A.
- Maximizes torque per ampere, *TPA*.

1 On the Cage Browser home, select **Optimization**.



2 In Create Optimization from Model, select `TPA` and **Next**.



Create Optimization from Model

Model

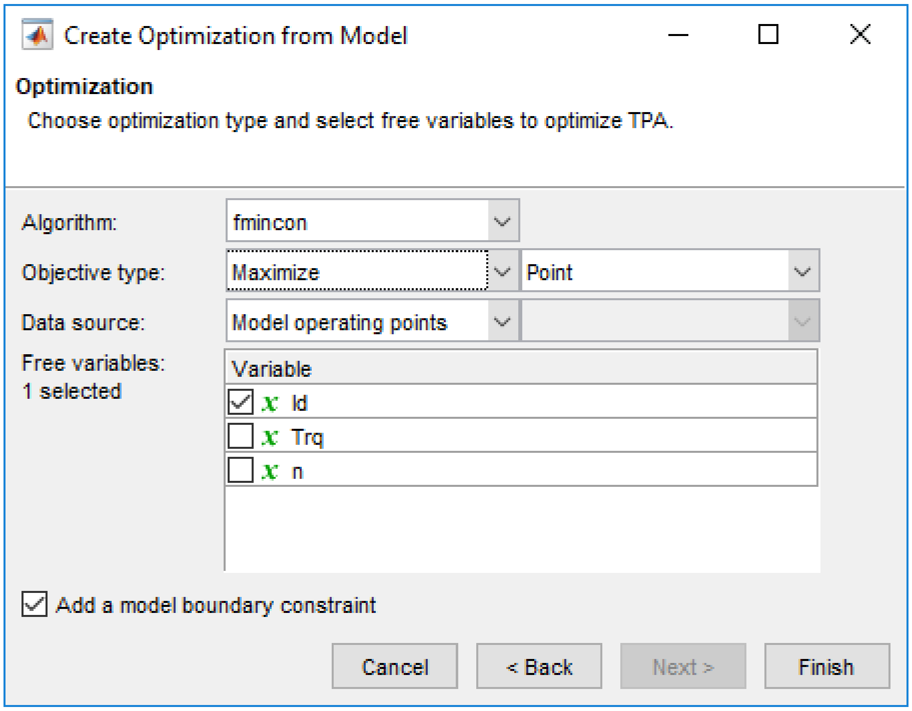
Select a model to minimize or maximize.

Model	Type	Variable Inputs
Iq	Point-by-point mo...	Id, Trq, n
Is	Function model	Id, Trq, n
TPA	Function model	Id, Trq, n

□ Create operating point data set

3 In Create Optimization from Model:

- Select `Id`.
- Set **Objective type** to `Maximize`.
- Click **Finish**.



Create Optimization from Model

Optimization

Choose optimization type and select free variables to optimize TPA.

Algorithm: fmincon

Objective type: Maximize | Point

Data source: Model operating points

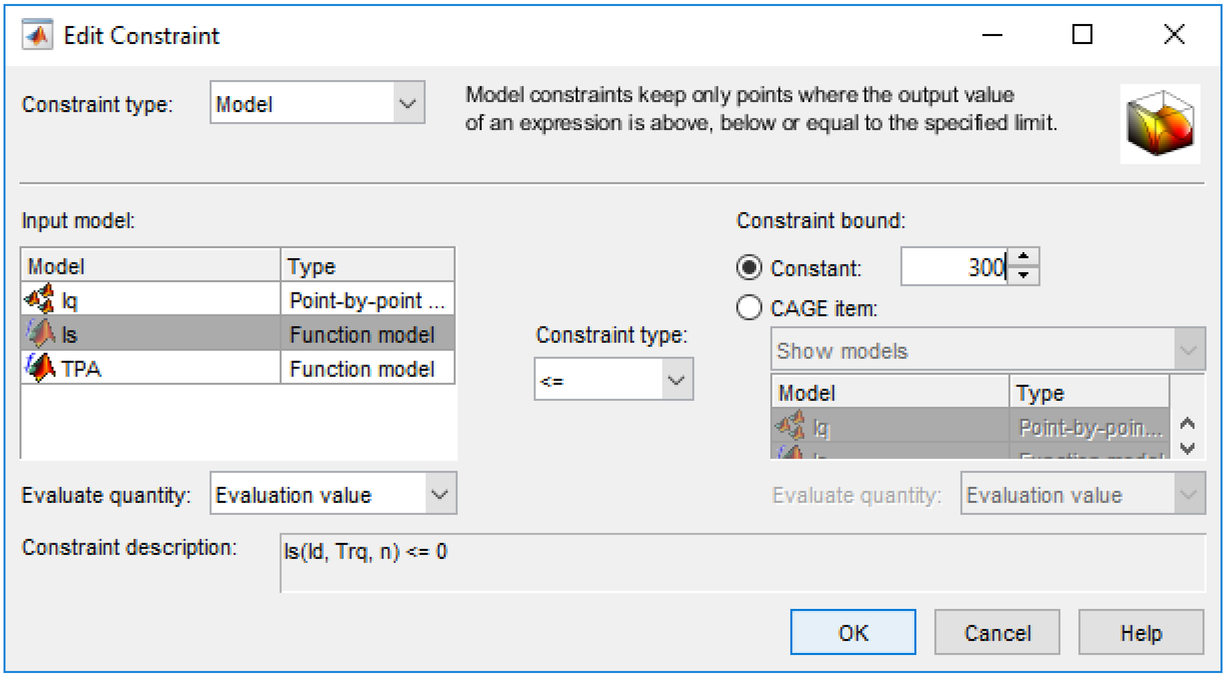
Free variables:
1 selected

Variable
☑ x Id
□ x Trq
□ x n

☑ Add a model boundary constraint

4 Add the optimization constraint for the current magnitude, I_s. In the CAGE Browser, select **Optimization** > **Constraints** > **Add Constraints** to open Edit Constraint. Use the dialog box to create a constraint on the current.

- `Is <= 300`



Constraint type: Model

Model constraints keep only points where the output value of an expression is above, below or equal to the specified limit.

Input model:

Model	Type
Iq	Point-by-point ...
Is	Function model
TPA	Function model

Constraint type: <=

Constraint bound:

● Constant: 300

○ CAGE item:

Show models

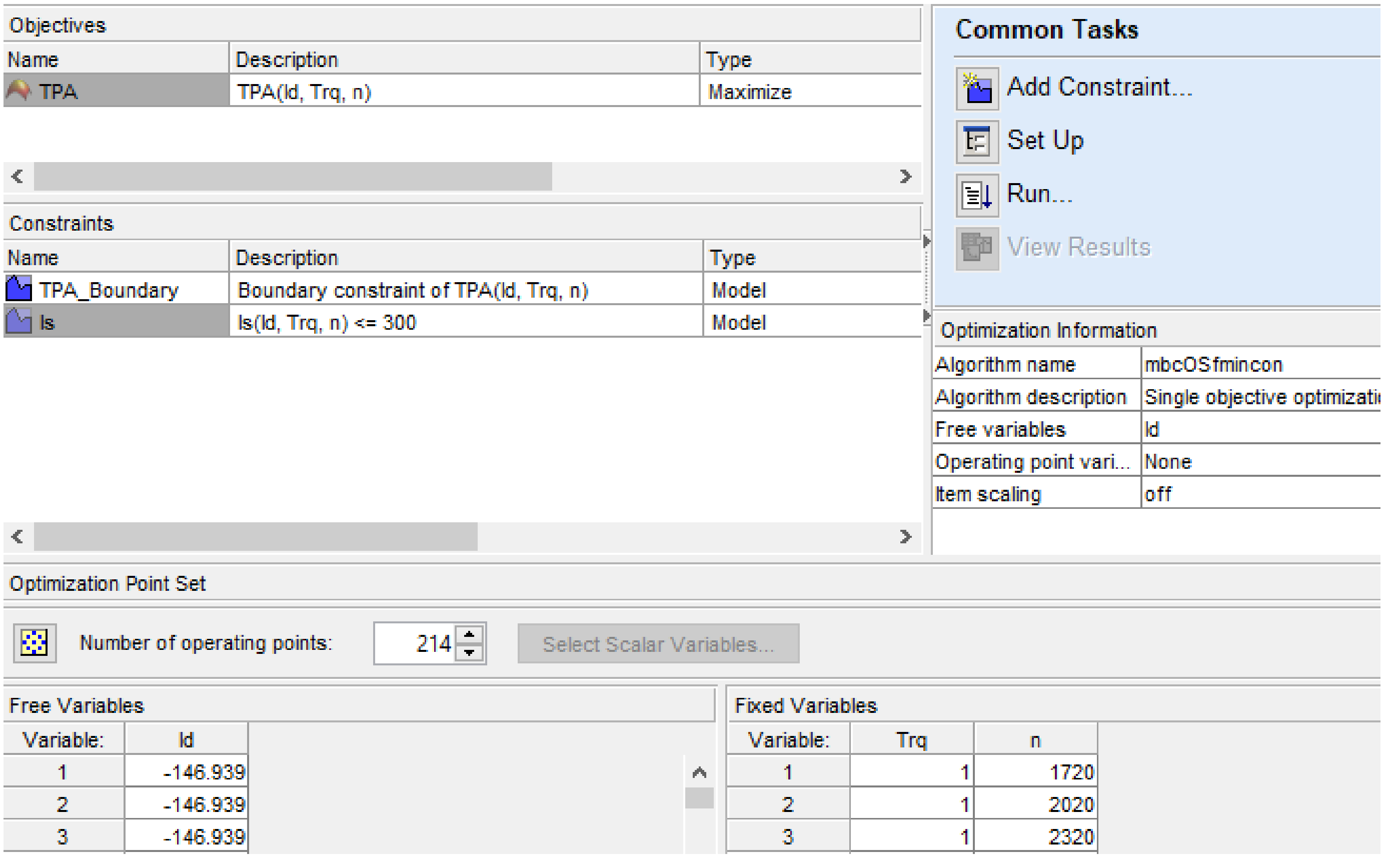
Model	Type
Iq	Point-by-poin...

Evaluate quantity: Evaluation value

Evaluate quantity: Evaluation value

Constraint description: Is(Id, Trq, n) <= 0

5 In the Cage Browser, *carefully* verify the Objectives and Constraints.



Objectives

Name	Description	Type
TPA	TPA(Id, Trq, n)	Maximize

Constraints

Name	Description	Type
TPA_Boundary	Boundary constraint of TPA(Id, Trq, n)	Model
Is	Is(Id, Trq, n) <= 300	Model

Optimization Information

Algorithm name	mbcOSfmincon
Algorithm description	Single objective optimizati
Free variables	Id
Operating point vari...	None
Item scaling	off

Optimization Point Set

Number of operating points: 214

Free Variables

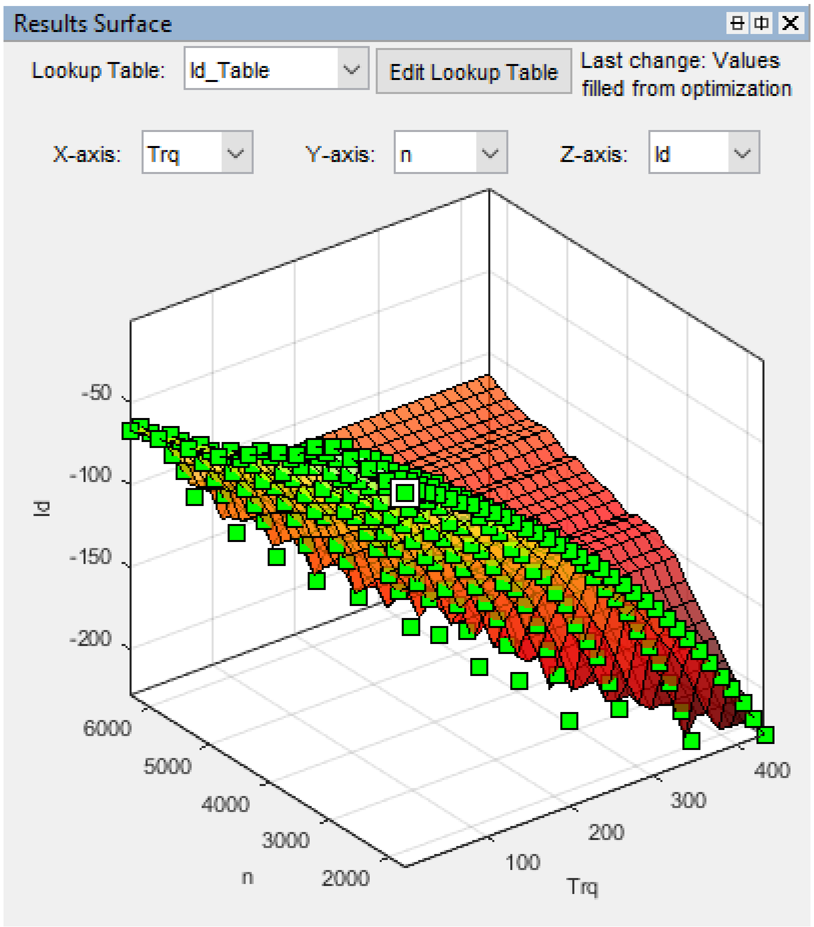
Variable:	Id
1	-146.939
2	-146.939
3	-146.939

Fixed Variables

Variable:	Trq	n
1	1	1720
2	1	2020
3	1	2320

6 In the Cage Browser, select **Run**.

The optimization results are similar to these.



Results Surface

Lookup Table: Id_Table

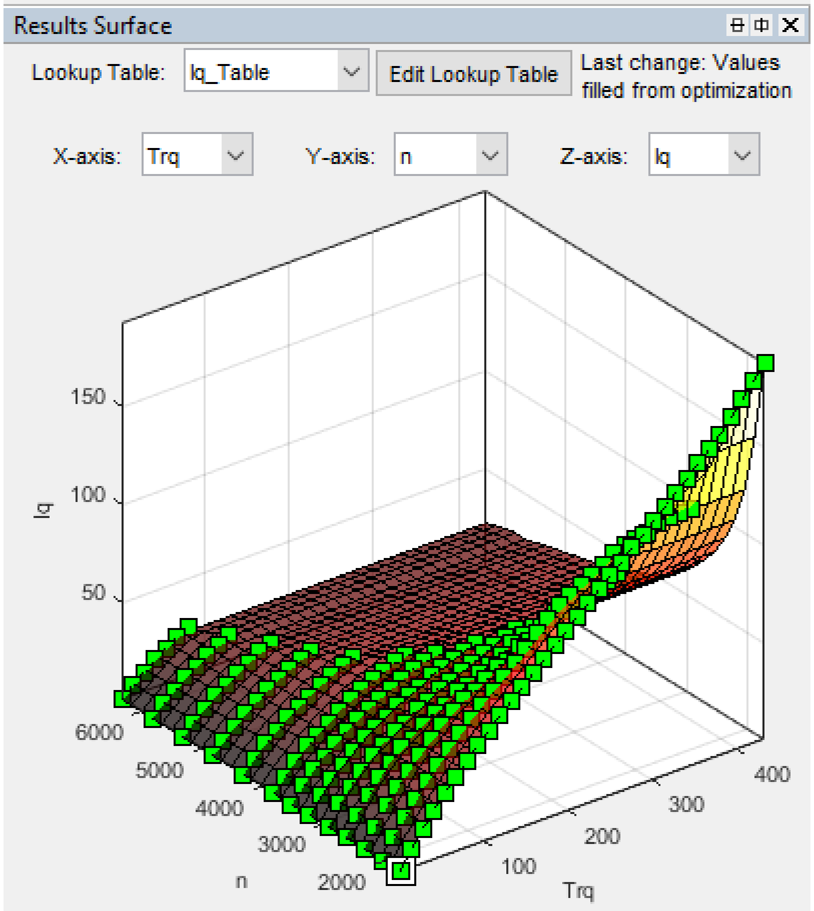
Edit Lookup Table

Last change: Values filled from optimization

X-axis: Trq

Y-axis: n

Z-axis: Id



Results Surface

Lookup Table: Iq_Table

Edit Lookup Table

Last change: Values filled from optimization

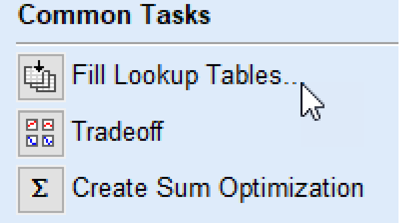
X-axis: Trq

Y-axis: n

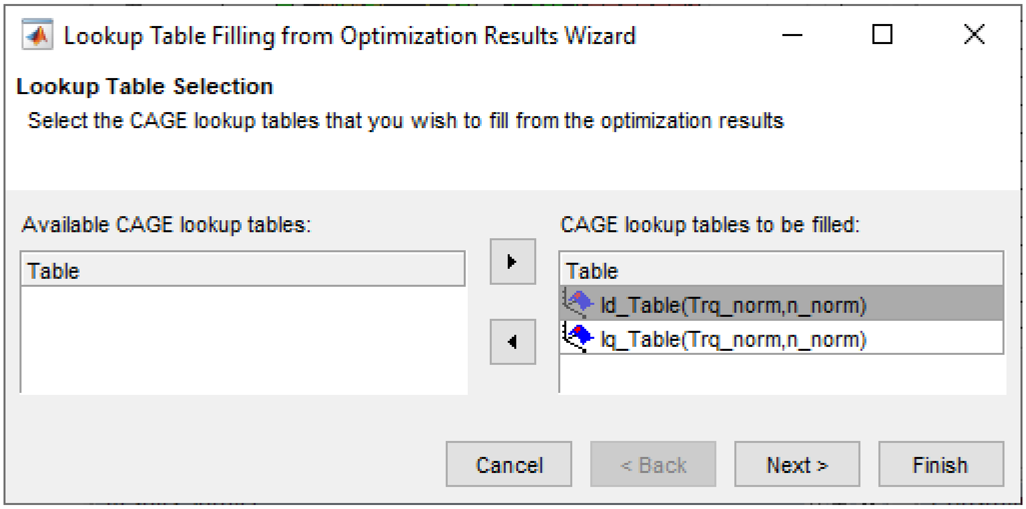
Z-axis: Iq

Fill Lookup Tables

1 In the CAGE Browser, select **Fill Lookup Tables**.



2 Use the Lookup Table Filling from Optimization Results Wizard to fill the `Id_Table` and `Iq_Table` tables.



Lookup Table Selection

Select the CAGE lookup tables that you wish to fill from the optimization results

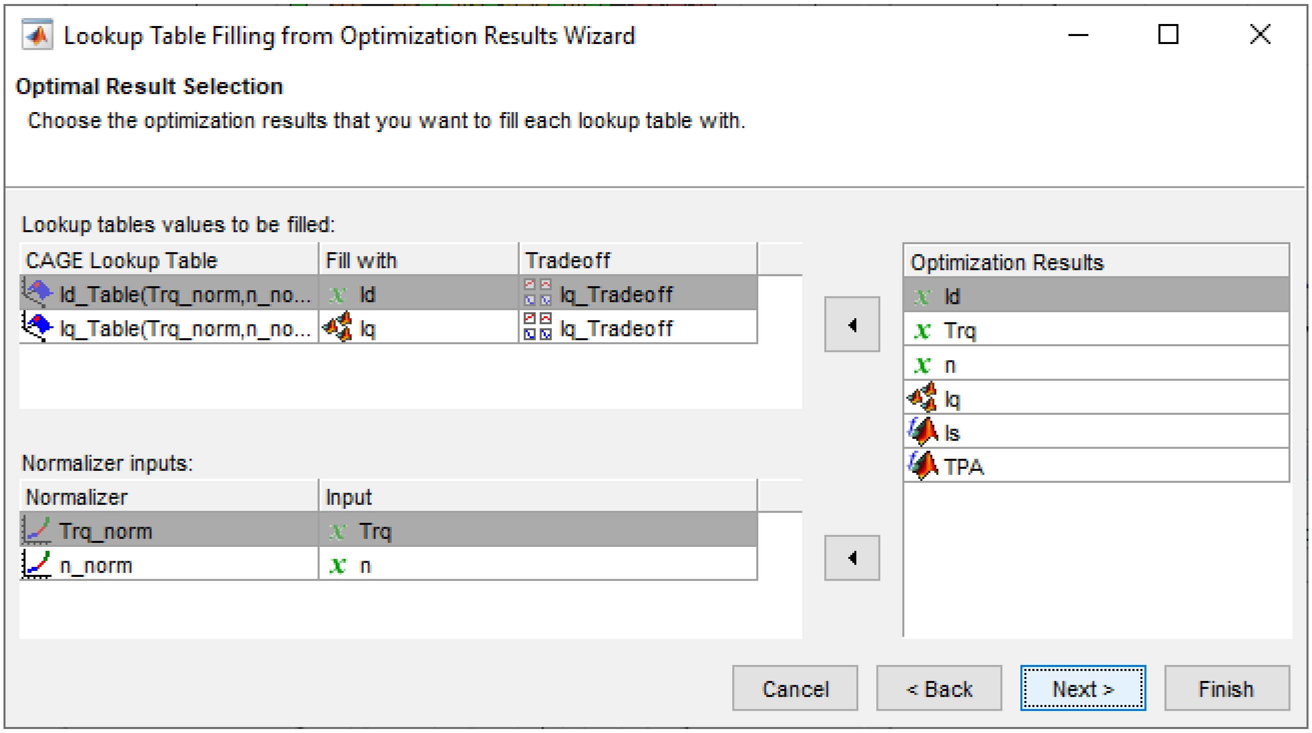
Available CAGE lookup tables:

Table

CAGE lookup tables to be filled:

Table
Id_Table(Trq_norm,n_norm)
Iq_Table(Trq_norm,n_norm)

- For the `Id_Table`, fill with `Id`.
- For the `Iq_Table`, fill with `Iq`.



Lookup Table Filling from Optimization Results Wizard

Optimal Result Selection

Choose the optimization results that you want to fill each lookup table with.

Lookup tables values to be filled:

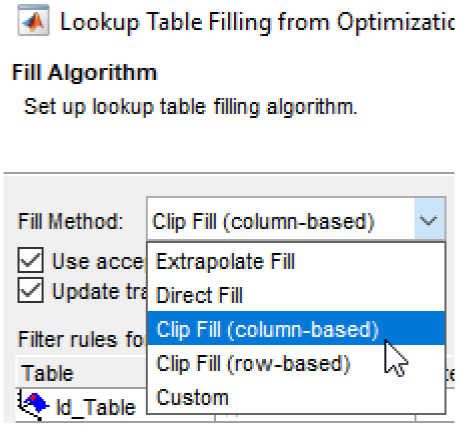
CAGE Lookup Table	Fill with	Tradeoff
Id_Table(Trq_norm,n_no...	Id	Iq_Tradeoff
Iq_Table(Trq_norm,n_no...	Iq	Iq_Tradeoff

Normalizer inputs:

Normalizer	Input
Trq_norm	Trq
n_norm	n

Optimization Results
Id
Trq
n
Iq
Is
TPA

Click **Next**. For the **Fill Method**, select `Clip Fill (column-based)`.



Lookup Table Filling from Optimizatic

Fill Algorithm

Set up lookup table filling algorithm.

Fill Method: Clip Fill (column-based)

☑ Use acce

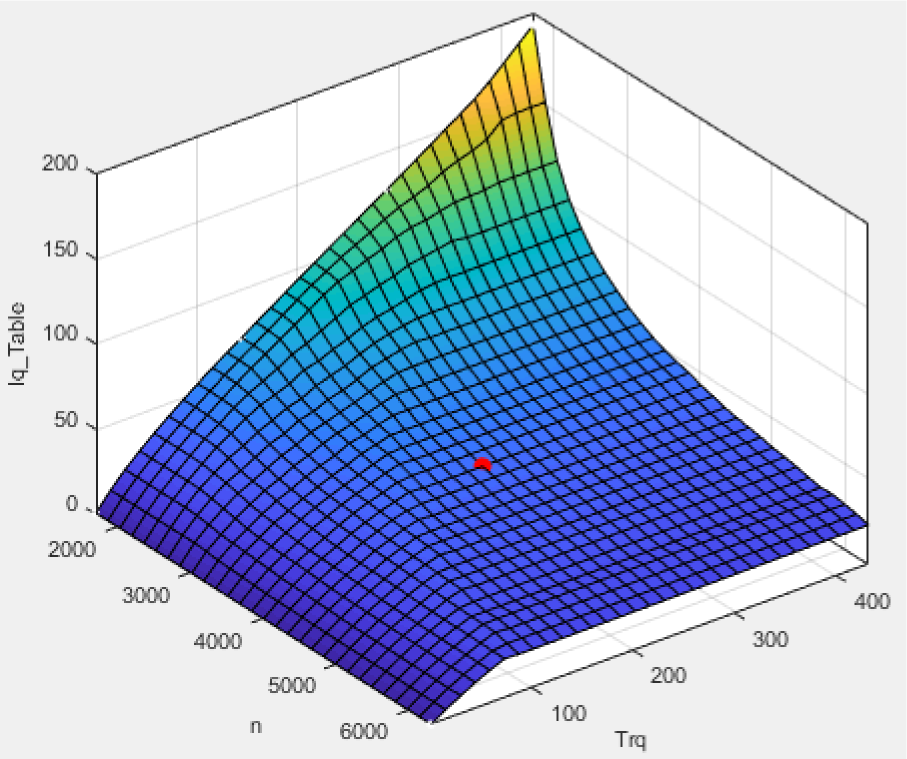
☑ Update tra

Filter rules fo

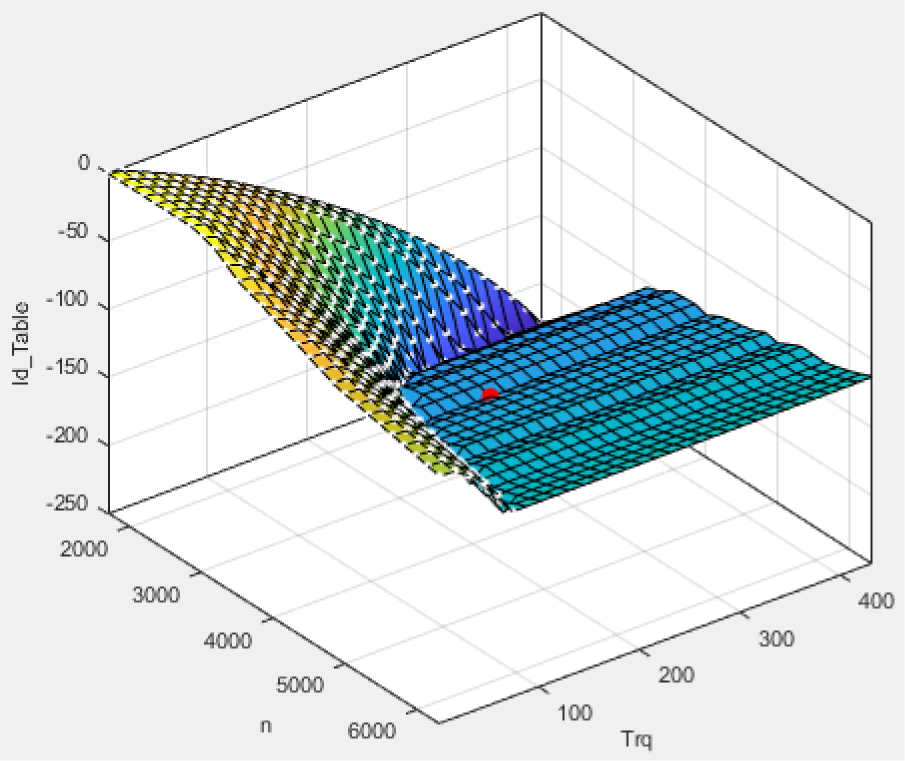
Table
Id_Table

Click **Finish**.

3 Review results for `Iq_Table`. The results are similar to these.



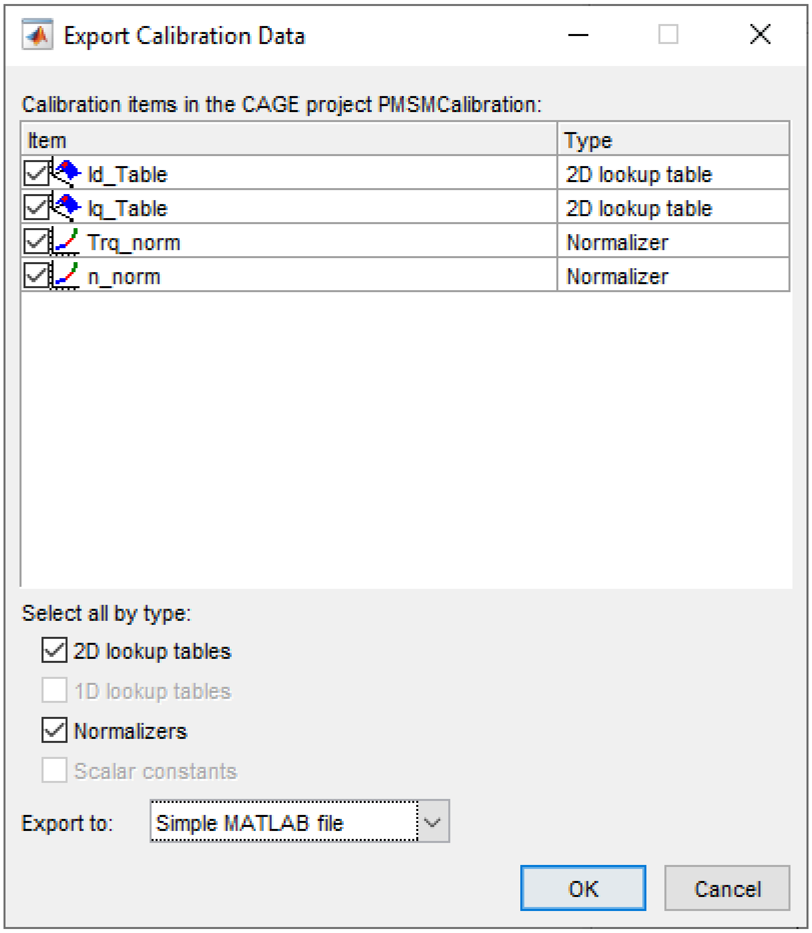
4 Review results for `Id_Table`. The results are similar to these.



5 Select **File** > **Save Project**. Save `gs_example.cag` to `work` folder.

Export Results

1. Select **File** > **Export** > **Calibration**.
2. Use Export Calibration Data to select the items to export and format. For example, export the Id and Iq tables and breakpoints to MATLAB file `gs_example.m`.



Calibration items in the CAGE project PMSMCalibration:

Item	Type
☑ Id_Table	2D lookup table
☑ Iq_Table	2D lookup table
☑ Trq_norm	Normalizer
☑ n_norm	Normalizer

Select all by type:

☑ 2D lookup tables

□ 1D lookup tables

☑ Normalizers

□ Scalar constants

Export to: Simple MATLAB file

Set Block Parameters

The optimized current controller calibration tables are functions of motor torque and motor speed. Use the tables for these Flux-Based PM Controller block parameters:

- **Corresponding d-axis current reference, id_ref**
- **Corresponding q-axis current reference, iq_ref**
- **Vector of speed breakpoints, wbp**
- **Vector of torque breakpoints, tbp**

To set the block parameters:

1. Run the .m file that contains the Model-Based Calibration Toolbox calibration results for the current controller. For example, in the MATLAB command line, run `gs_example.m`:

```
% Access data from MBC current controller calibration
gs_example
```

2 Assign the breakpoint parameters to the data contained in the .m file. In this example, the speed data is in rpm. To use the calibration data for the block parameters, convert the speed breakpoints from rpm to rad/s.

Parameter	MATLAB Commands
Vector of speed breakpoints, wbp	`tbp=Trq_norm.X;`

Vector of speed breakpoints, wbp	`% MBC data for speed is in rpm.`
	`% For the block parameter, use rad/s`
	`nbp=n_norm.X;`
	`conversion=(2*pi/60.);`
	`wbp=conversion.*nbp;`

Corresponding d-axis current reference, id_ref	`id_table=Id_Table.Z;`
	`id_ref=id_table';`

Corresponding q-axis current reference, iq_ref	`iq_table=Iq_Table.Z;`
	`iq_ref=iq_table';`

Generate Feed-Forward Flux Parameters

Using MathWorks tools, you can create lookup tables for an interior permanent magnet synchronous motor (PMSM) controller that characterizes the *d*-axis and *q*-axis flux as a function of *d*-axis and *q*-axis currents.

To generate the flux parameters for the Flux-Based PM Controller block, follow these workflow steps. The steps use example script `VisualizeFluxSurface.m`.

Workflow	Description
"Step 1: Load and Preprocess Data" on page 6-49	Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA): • *d*- and *q*- axis current • *d*- and *q*- axis flux • Electromagnetic motor torque
"Step 2: Generate Evenly Spaced Data" on page 6-49	Use spline interpolation to generate evenly spaced data. Visualize the flux surface plots.
"Step 3: Set Block Parameters" on page 6-51	Set workspace variables that you can use for the Flux-Based PM Controller block parameters.

Step 1: Load and Preprocess Data

Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA):

- *d*- and *q*- axis current
- *d*- and *q*- axis flux
- Electromagnetic motor torque

1. Open the example script `VisualizeFluxSurface.m`.
2. Load and preprocess the data.

```
%
% Load the data from a |mat| file captured from a dynamometer or
% another CAE tool.
load FEAdata.mat;

% Load the data matrix.
lambda_d = FEAdata.flux.d;
lambda_q = FEAdata.flux.q;
id = FEAdata.current.d;
iq = FEAdata.current.q;
```

Step 2: Generate Evenly Spaced Data

The flux tables and can have different step sizes for the currents. Evenly spacing the rows and columns helps improve interpolation accuracy. This script uses spline interpolation.

1. Set the spacing for the table rows and columns.

   ```
   % Set the spacing for the table rows and columns
   flux_d_size = 50;
   flux_q_size = 50;
   ```

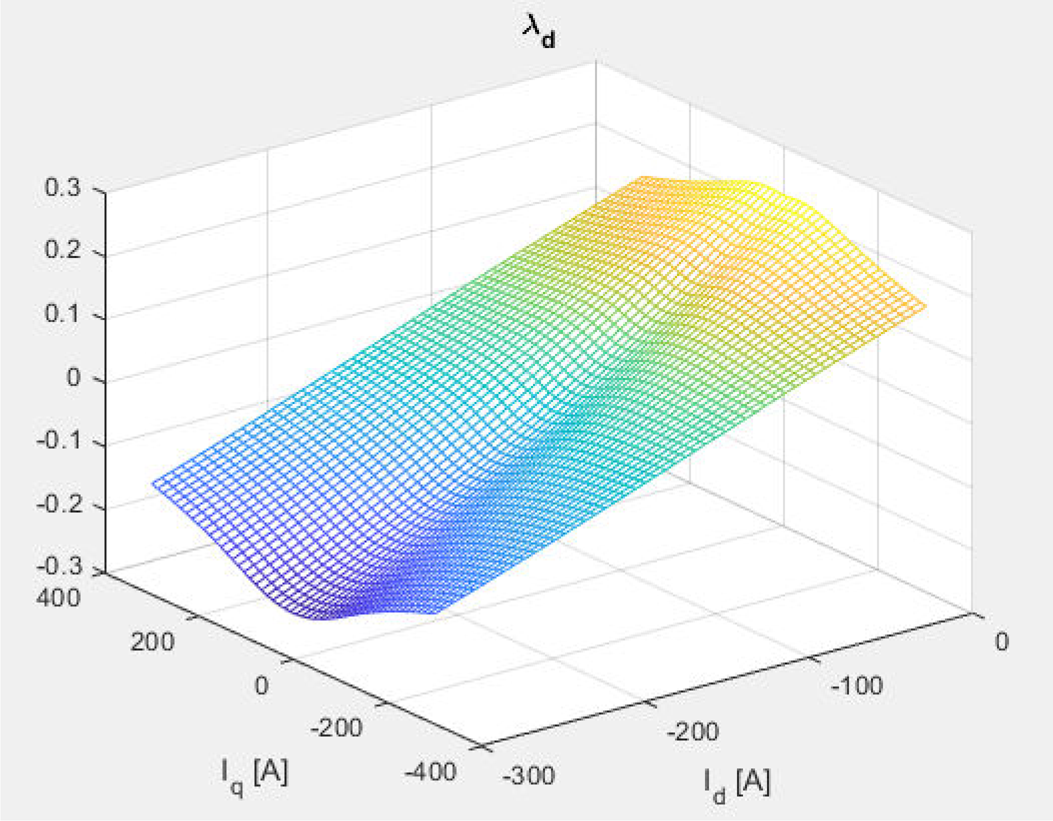
2. Use spline interpolation to get higher resolution.

   ```
   % Use spline interpolation to get higher resolution
   id_new = linspace(min(id),max(id),flux_d_size);
   iq_new = linspace(min(iq),max(iq),flux_q_size);
   lambda_d_new = interp2(id',iq,lambda_d,id_new',iq_new,'spline');
   lambda_q_new = interp2(id',iq,lambda_q,id_new',iq_new,'spline');
   ```

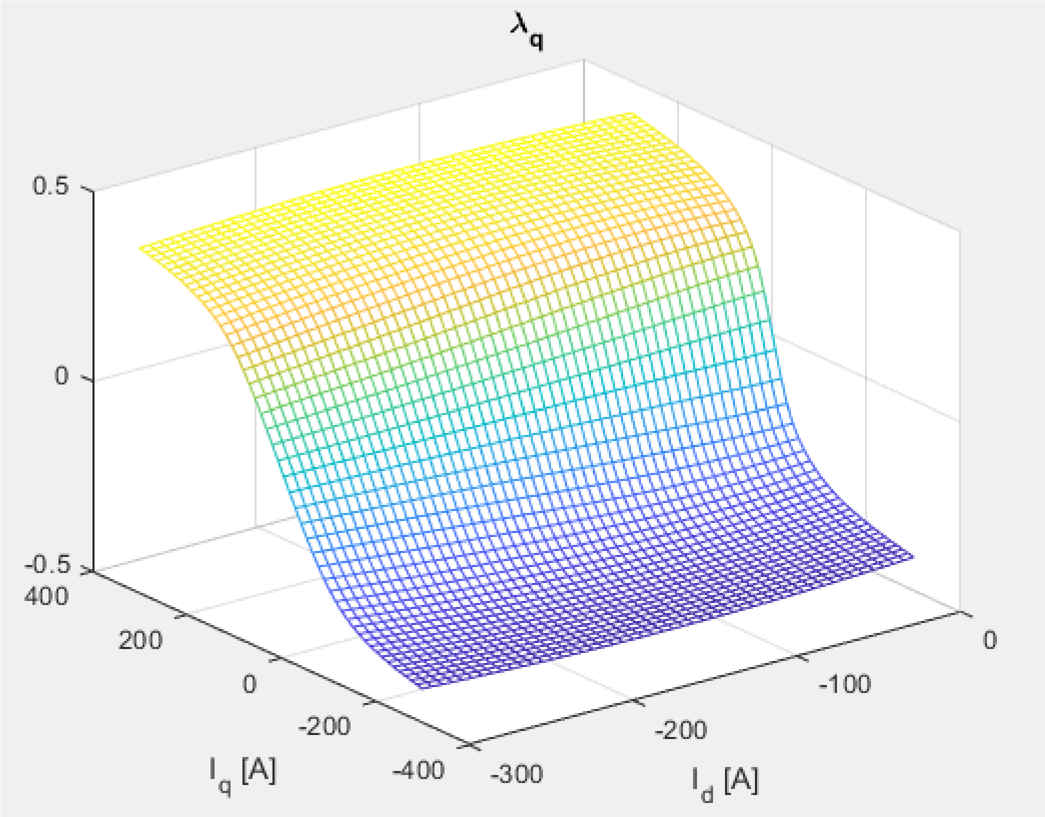
3. Visualize the flux surfaces.

   ```
   % Visualize the flux surface
   figure;
   mesh(id_new,iq_new,lambda_d_new);
   xlabel('I_d [A]')
   ylabel('I_q [A]')
   title('\lambda_d'); grid on;

   figure;
   mesh(id_new,iq_new,lambda_q_new);
   xlabel('I_d [A]')
   ylabel('I_q [A]')
   title('\lambda_q'); grid on;
   ```

   - d-axis flux, λ_d, as a function of d-axis current, I_d, and q-axis current, I_q.



- q-axis flux, λ_q, as a function of d-axis current, I_d, and q-axis current, I_q.



Step 3: Set Block Parameters

Set the block parameters to these values assigned in the example script.

Parameter	MATLAB Commands
Vector of d-axis current breakpoints, id_index	`id_index=id_new;`
Vector of q-axis current breakpoints, iq_index	`iq_index=iq_new;`
Corresponding d-axis flux, lambda_d	`lambda_d=lambda_d_new;`
Corresponding q-axis flux, lambda_q	`lambda_q=lambda_q_new;`

References

[1] Hu, Dakai, Yazan Alsmadi, and Longya Xu. “High fidelity nonlinear IPM modeling based on measured stator winding flux linkage.” *IEEE Transactions on Industry Applications*, Vol. 51, No. 4, July/August 2015.

[2] Chen, Xiao, Jiabin Wang, Bhaskar Sen, Panagiotis Lasari, Tianfu Sun. “A High-Fidelity and Computationally Efficient Model for Interior Permanent-Magnet Machines Considering the Magnetic Saturation, Spatial Harmonics, and Iron Loss Effect.” *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 7, July 2015.

[3] Ottosson, J., M. Alakula. “A compact field weakening controller implementation.” *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, July, 2006.

See Also

Flux-Based PM Controller | Flux-Based PMSM

Generate Parameters for Flux-Based PMSM Block

Using MathWorks tools, you can create lookup tables for an interior permanent magnet synchronous motor (PMSM) controller that characterizes the *d*-axis and *q*-axis current as a function of *d*-axis and *q*-axis flux.

To generate the flux parameters for the Flux-Based PMSM block, follow these workflow steps. Example script `CreatingIdqTable.m` calls `gridfit` to model the current surface using scattered or semi-scattered flux data.

Workflow	Description
"Step 1: Load and Preprocess Data" on page 6-53	Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA): • *d*- and *q*- axis current • *d*- and *q*- axis flux • Electromagnetic motor torque
"Step 2: Generate Evenly Spaced Table Data From Scattered Data" on page 6-54	Use the `gridfit` function to generate evenly spaced data. Visualize the flux surface plots.
"Step 3: Set Block Parameters" on page 6-56	Set workspace variables that you can use for the Flux-Based PM Controller block parameters.

Step 1: Load and Preprocess Data

Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA):

- *d*- and *q*- axis current
- *d*- and *q*- axis flux
- Electromagnetic motor torque

1. Open the example script `CreatingIdqTable.m`.
2. Load and preprocess the data.

   ```
   % Load the data from a |mat| file captured from a dynamometer or
   % another CAE tool.
   load FEAdata.mat;
   ```
3. Determine the minimum and maximum flux values.

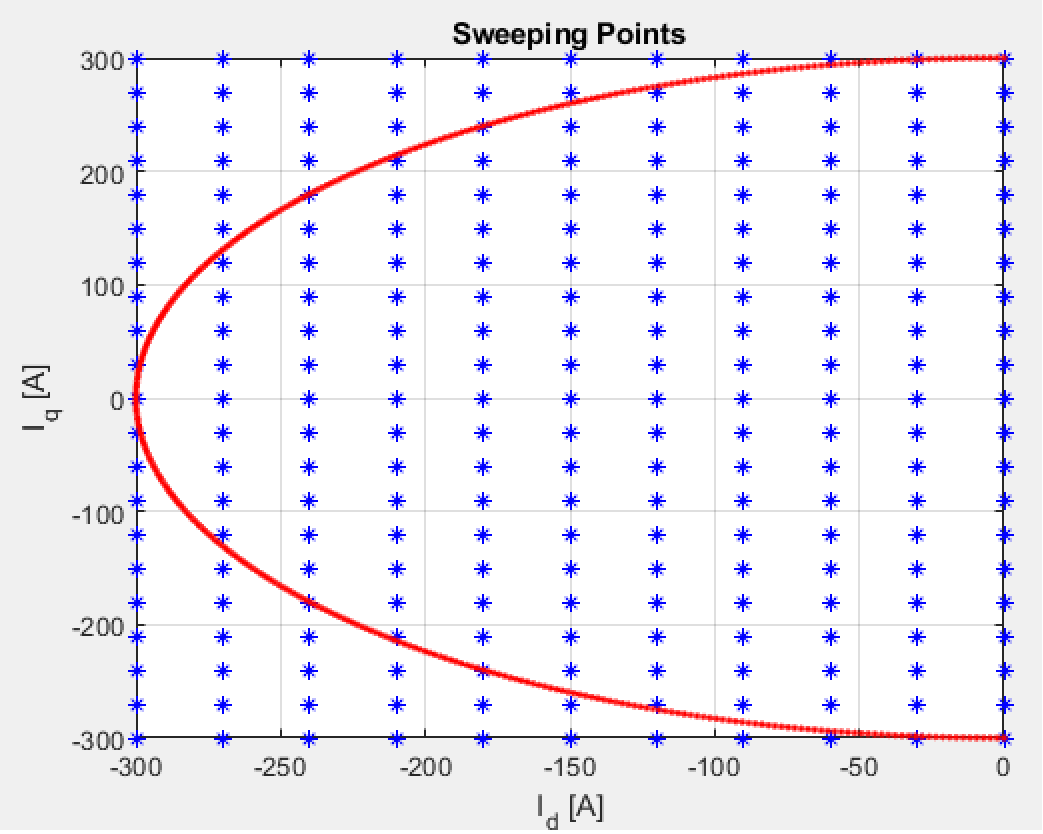
   ```
   flux_d_min = min(min(FEAdata.flux.d)) ;
   flux_d_max = max(max(FEAdata.flux.d)) ;
   flux_q_min = min(min(FEAdata.flux.q)) ;
   flux_q_max = max(max(FEAdata.flux.q)) ;
   ```
4. Plot the sweeping current points used to collect the data.

   ```
   for i = 1:length(FEAdata.current.d)
       for j = 1:1:length(FEAdata.current.q)
       plot(FEAdata.current.d(i),FEAdata.current.q(j),'b*');
       hold on
   ```

```
        end
    end
```

5 Plot the current limit sweeping points and circle.

```
for angle_theta = pi/2:(pi/2/200):(3*pi/2)
    plot(300*cos(angle_theta),300*sin(angle_theta),'r.');
    hold on
end
xlabel('I_d [A]')
ylabel('I_q [A]')
title('Sweeping Points'); grid on;
xlim([-300,0]);
ylim([-300,300]);
hold off
```



Step 2: Generate Evenly Spaced Table Data From Scattered Data

The flux tables and can have different step sizes for the currents. Evenly spacing the rows and columns helps improve interpolation accuracy. This script uses spline interpolation.

1 Set the spacing for the table rows and columns.

```
% Set the spacing for the table rows and columns
flux_d_size = 50;
flux_q_size = 50;
```

2 Generate linear spaced vectors for the breakpoints.

```
% Generate linear spaced vectors for the breakpoints
ParamFluxDIndex = linspace(flux_d_min,flux_d_max,flux_d_size);
ParamFluxQIndex = linspace(flux_q_min,flux_q_max,flux_q_size);
```

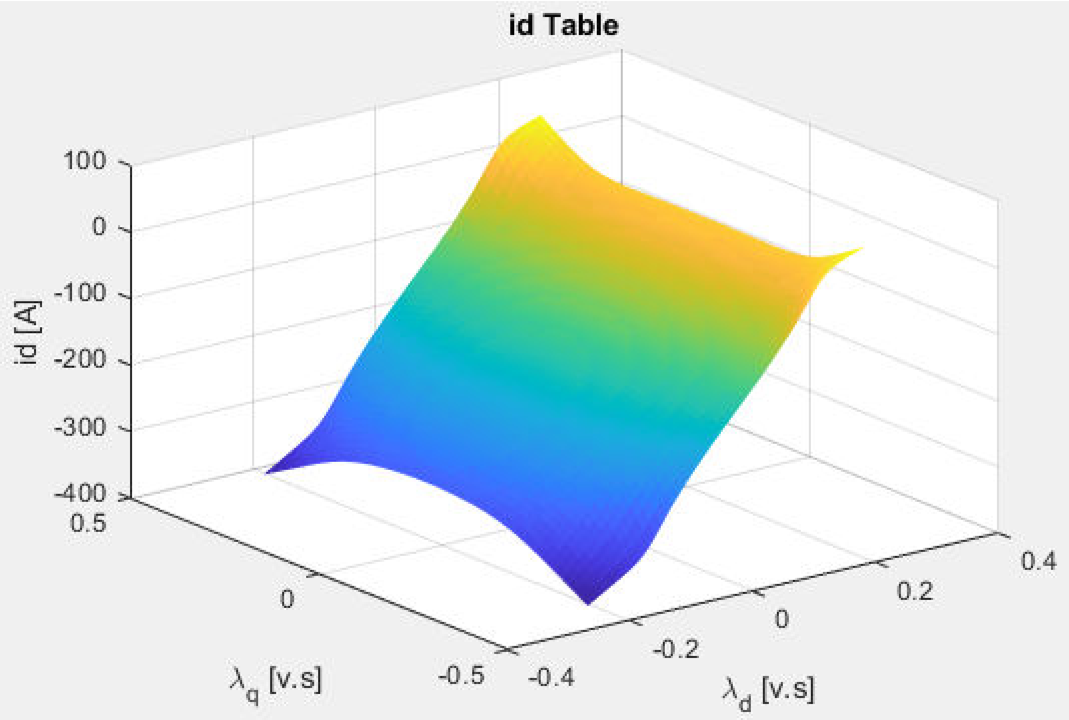
3 Create 2-D grid coordinates based on the *d*-axis and *q*-axis currents.

```
% Create 2-D grid coordinates based on the d-axis and q-axis currents
[id_m,iq_m] = meshgrid(FEAdata.current.d,FEAdata.current.q);
```

4 Create the table for the *d*-axis current.

```
% Create the table for the d-axis current
id_fit = gridfit(FEAdata.flux.d,FEAdata.flux.q,id_m,ParamFluxDIndex,ParamFluxQIndex);
ParamIdLookupTable = id_fit';
figure;
surf(ParamFluxDIndex,ParamFluxQIndex,ParamIdLookupTable');
xlabel('\lambda_d [v.s]');ylabel('\lambda_q [v.s]');zlabel('id [A]');title('id Table'); grid on;
shading flat;
```

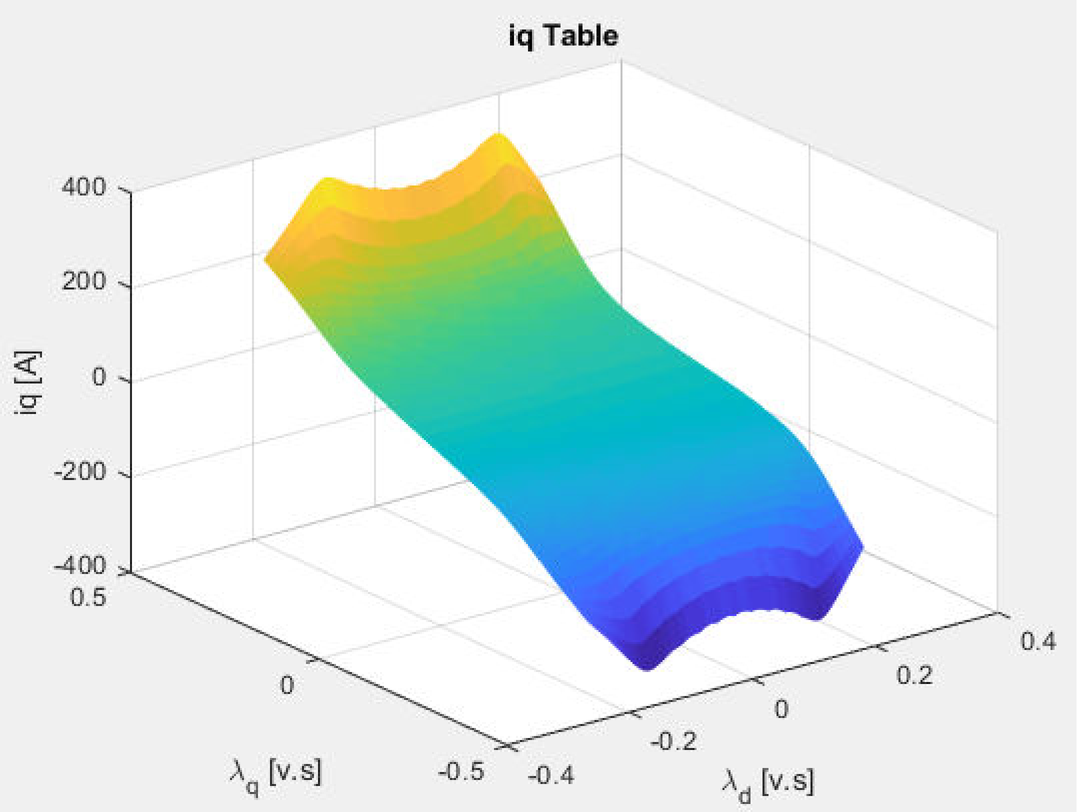
d-axis current, I_d, as a function of q-axis flux, λ_q, and d-axis flux, λ_d.



5 Create the table for the *q*-axis current.

```
% Create the table for the q-axis current
iq_fit = gridfit(FEAdata.flux.d,FEAdata.flux.q,iq_m,ParamFluxDIndex,ParamFluxQIndex);
ParamIqLookupTable = iq_fit';
figure;
surf(ParamFluxDIndex,ParamFluxQIndex,ParamIqLookupTable');
xlabel('\lambda_d [v.s]');ylabel('\lambda_q [v.s]');zlabel('iq [A]'); title('iq Table'); grid on;
shading flat;
```

q-axis current, I_q, as a function of q-axis flux, λ_q, and d-axis flux, λ_d.



Step 3: Set Block Parameters

Set the block parameters to these values assigned in the example script.

Parameter	MATLAB Commands
Vector of d-axis flux, flux_d	`flux_d=ParamFluxDIndex;`
Vector of q-axis flux, flux_q	`flux_q=ParamFluxQIndex;`
Corresponding d-axis current, id	`id=ParamIdLookupTable;`
Corresponding q-axis current, iq	`iq=ParamIqLookupTable;`

References

[1] Hu, Dakai, Yazan Alsmadi, and Longya Xu. “High fidelity nonlinear IPM modeling based on measured stator winding flux linkage.” *IEEE Transactions on Industry Applications*, Vol. 51, No. 4, July/August 2015.

[2] Chen, Xiao, Jiabin Wang, Bhaskar Sen, Panagiotis Lasari, Tianfu Sun. “A High-Fidelity and Computationally Efficient Model for Interior Permanent-Magnet Machines Considering the Magnetic Saturation, Spatial Harmonics, and Iron Loss Effect.” *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 7, July 2015.

[3] Ottosson, J., M. Alakula. “A compact field weakening controller implementation.” *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, July, 2006.

See Also

Flux-Based PM Controller | Flux-Based PMSM

External Websites

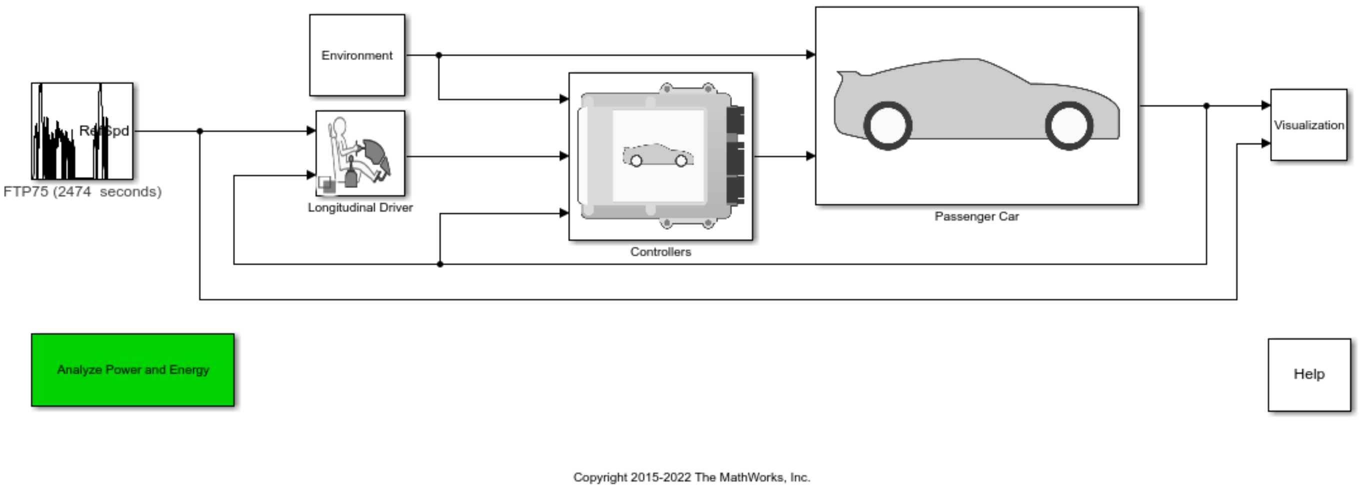
- Surface Fitting using gridfit

Powertrain Blockset Examples

Conventional Vehicle Reference Application

The conventional vehicle reference application represents a full vehicle model with an internal combustion engine, transmission, and associated powertrain control algorithms. Use the reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing.

For more information, see “Build a Conventional Vehicle Model” on page 3-5.



See Also

Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Conventional Vehicle Spark-Ignition Engine Fuel Economy and Emissions” on page 1-10
- “Conventional Vehicle Powertrain Efficiency” on page 1-15
- “Generate Deep Learning SI Engine Model” on page 3-117

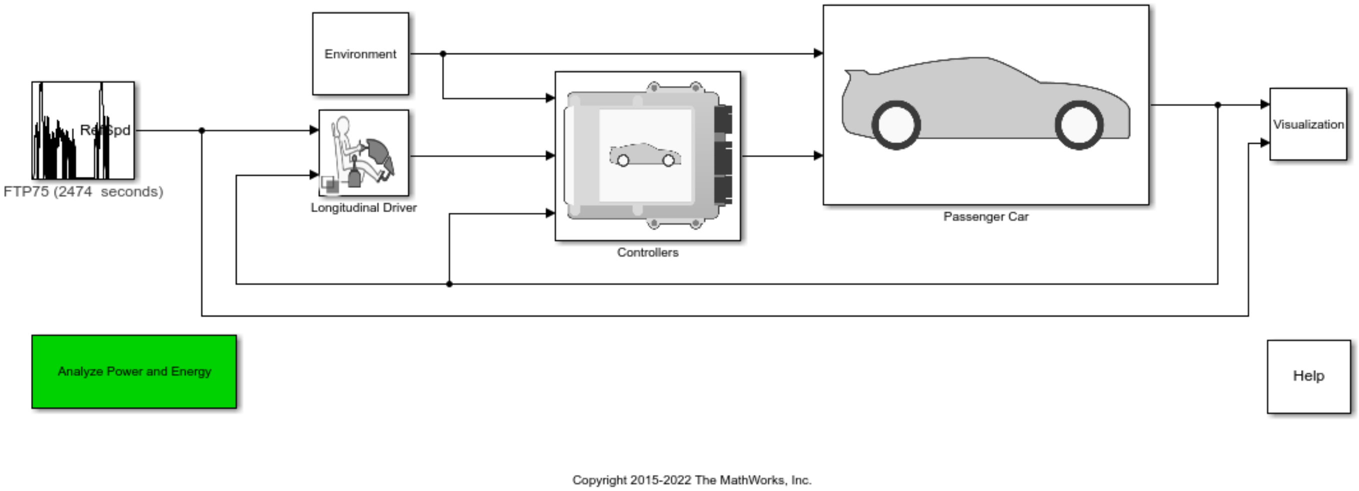
More About

- “Analyze Power and Energy” on page 3-129
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128
- “Variant Systems”

HEV Multimode Reference Application

The hybrid electric vehicle (HEV) multimode reference application represents a full multimode HEV model with an internal combustion engine, transmission, battery, motor, generator, and associated powertrain control algorithms. Use the reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing.

For more information, see “Build Hybrid Electric Vehicle Multimode Model” on page 3-19.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle P2 Model” on page 3-65
- “Build Full Electric Vehicle Model” on page 3-26

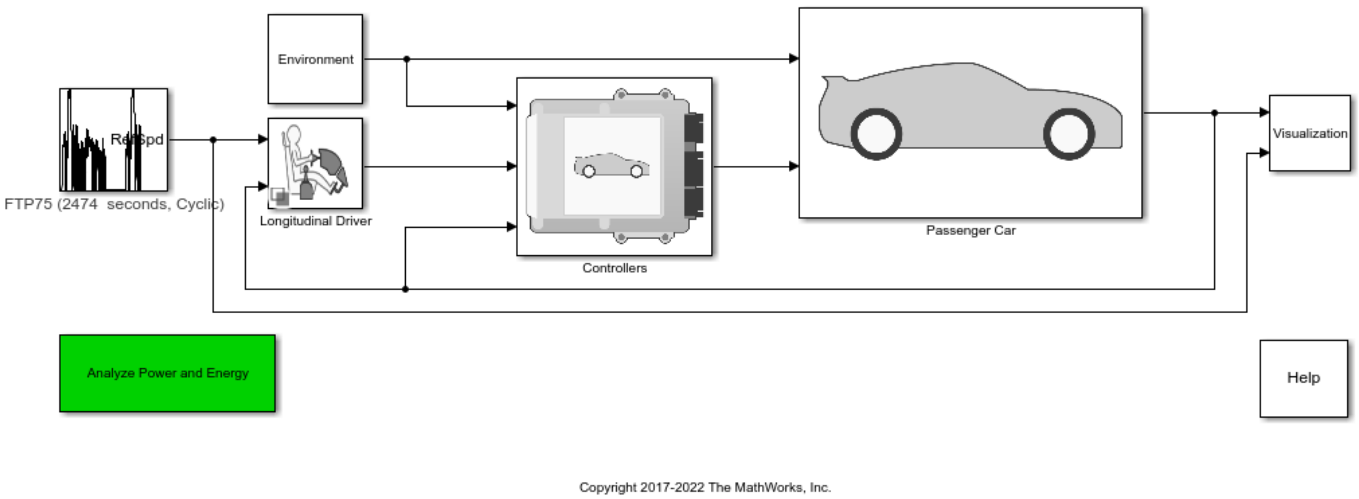
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV Input Power-Split Reference Application

The hybrid electric vehicle (HEV) input power-split reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, generator, and associated powertrain control algorithms. Use the HEV input power-split reference application for HIL testing, tradeoff analysis, and control parameter optimization of a power-split hybrid like the Toyota® Prius®.

For more information, see “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Hybrid Electric Vehicle P2 Model” on page 3-65
- “Build Full Electric Vehicle Model” on page 3-26

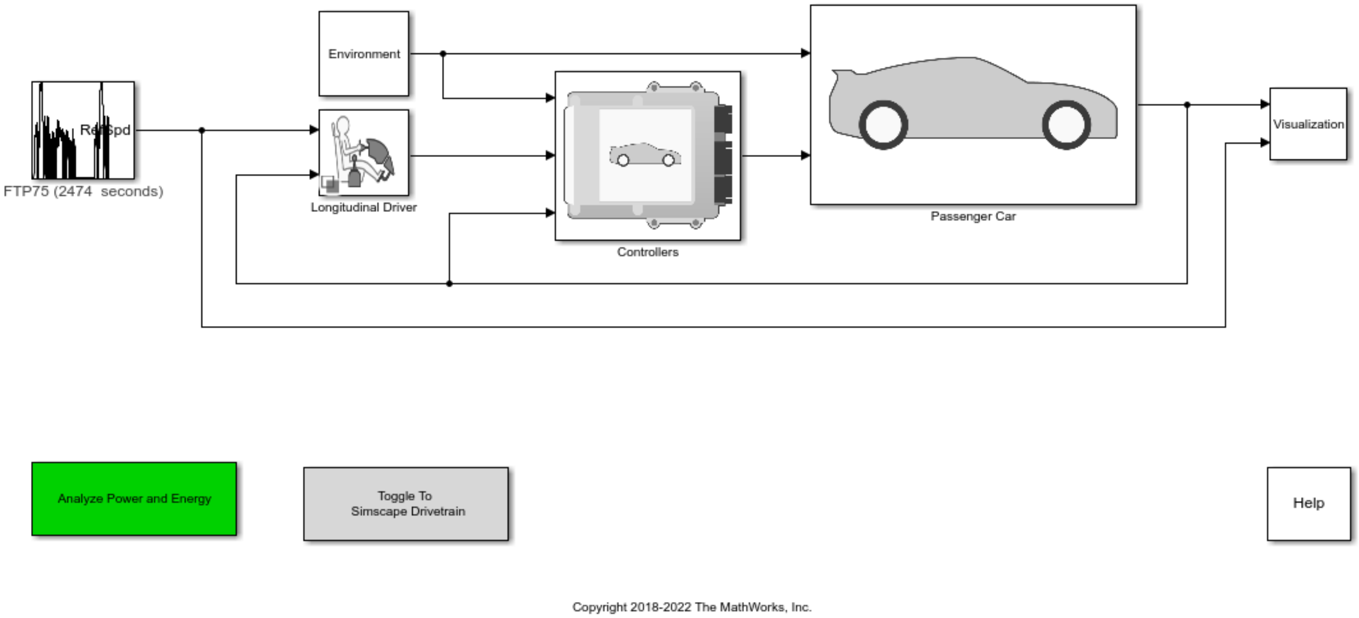
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV P0 Reference Application

The hybrid electric vehicle (HEV) P0 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P0 hybrid.

For more information, see “Build Hybrid Electric Vehicle P0 Model” on page 3-51.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Full Electric Vehicle Model” on page 3-26

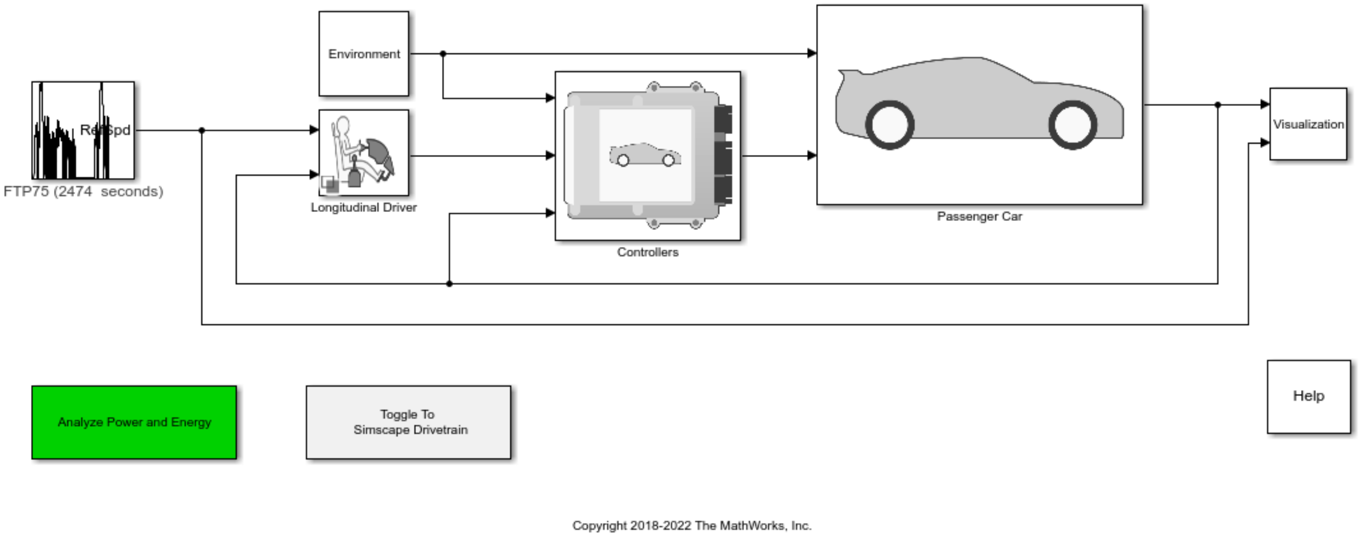
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV P1 Reference Application

The hybrid electric vehicle (HEV) P1 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P1 hybrid.

For more information, see “Build Hybrid Electric Vehicle P1 Model” on page 3-58.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Full Electric Vehicle Model” on page 3-26

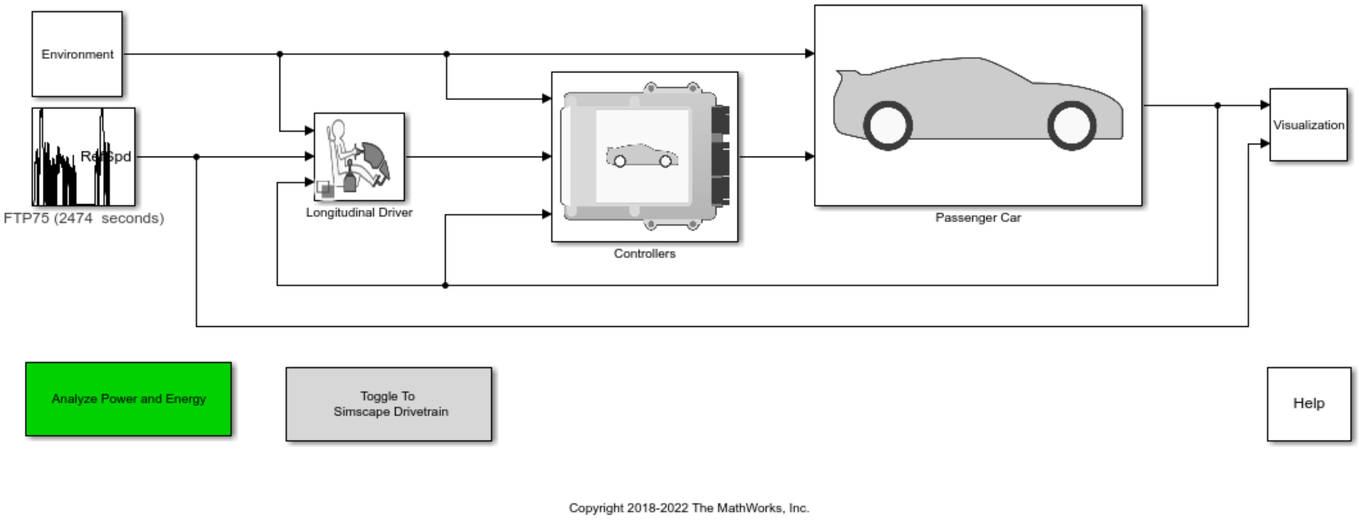
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV P2 Reference Application

The hybrid electric vehicle (HEV) P2 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P2 hybrid.

For more information, see “Build Hybrid Electric Vehicle P2 Model” on page 3-65.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Full Electric Vehicle Model” on page 3-26

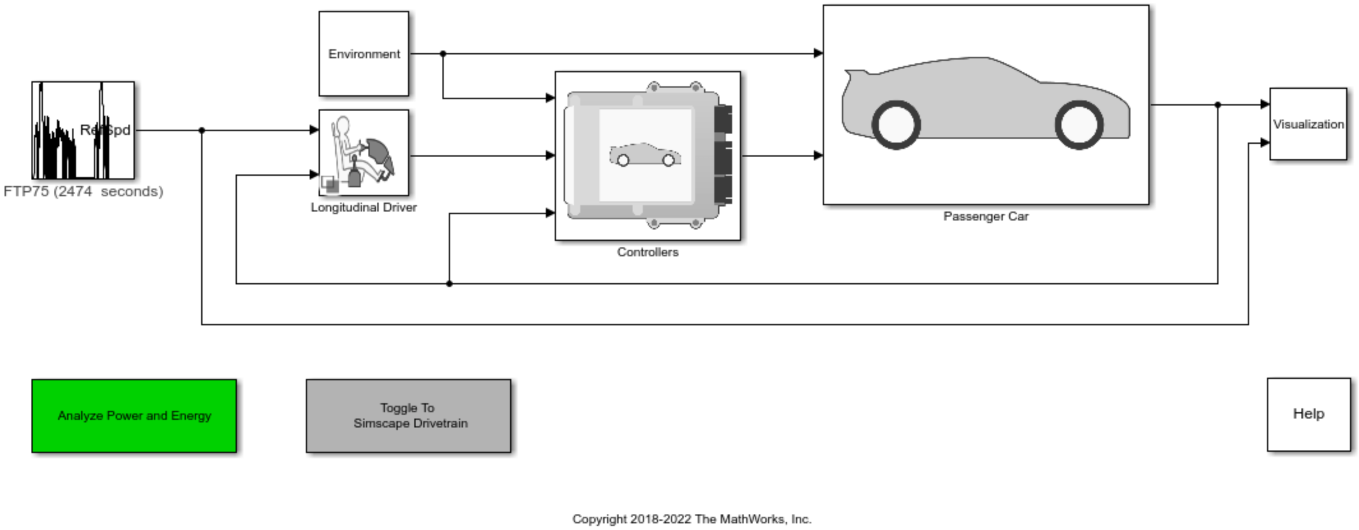
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV P3 Reference Application

The hybrid electric vehicle (HEV) P3 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P3 hybrid.

For more information, see “Build Hybrid Electric Vehicle P3 Model” on page 3-75.



See Also

Interior PMSM | Interior PM Controller | Drive Cycle Source | Longitudinal Driver | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Controller

Related Examples

- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Full Electric Vehicle Model” on page 3-26

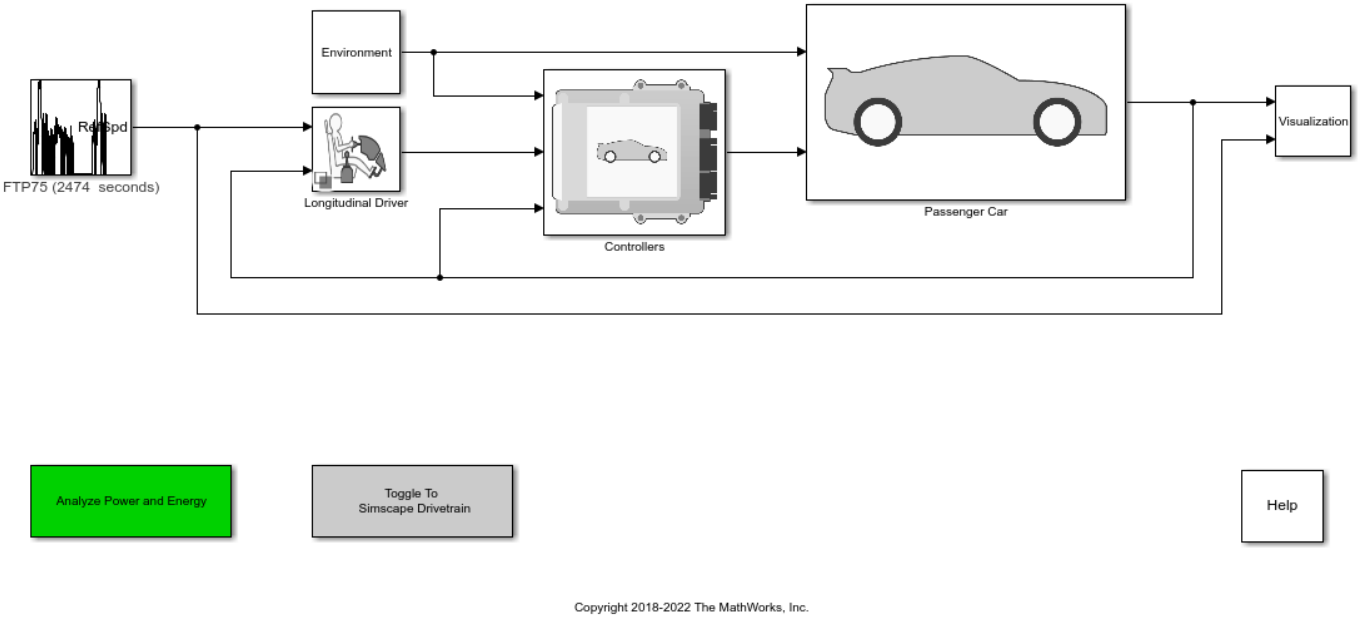
More About

- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

HEV P4 Reference Application

The hybrid electric vehicle (HEV) P4 reference application represents a full HEV model with an internal combustion engine, transmission, battery, motor, and associated powertrain control algorithms. Use the reference application for hardware-in-the-loop (HIL) testing, tradeoff analysis, and control parameter optimization of a HEV P4 hybrid.

For more information, see "Build Hybrid Electric Vehicle P4 Model" on page 3-82.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | SI Core Engine | Mapped SI Engine | SI Controller | Mapped CI Engine | CI Core Engine | CI Controller

Related Examples

- "Build Hybrid Electric Vehicle Input Power-Split Model" on page 3-42
- "Build Hybrid Electric Vehicle Multimode Model" on page 3-19
- "Build Full Electric Vehicle Model" on page 3-26

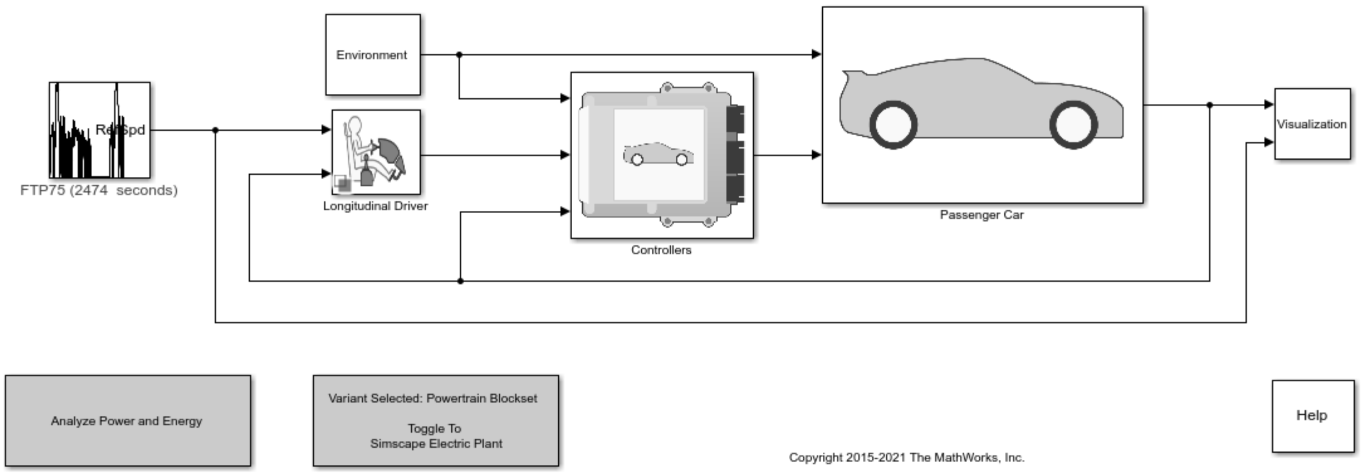
More About

- "Analyze Power and Energy" on page 3-129
- "Variant Systems"

EV Reference Application

The electric vehicle (EV) reference application represents a full electric vehicle model with a motor-generator, battery, direct-drive transmission, and associated powertrain control algorithms. Use the electric vehicle reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing.

For more information, see “Build Full Electric Vehicle Model” on page 3-26.



See Also

Interior PMSM | Interior PM Controller | Datasheet Battery | Drive Cycle Source | Longitudinal Driver | Mapped Motor

Related Examples

- “Build Hybrid Electric Vehicle Multimode Model” on page 3-19
- “Build Hybrid Electric Vehicle Input Power-Split Model” on page 3-42
- “Build Hybrid Electric Vehicle P2 Model” on page 3-65

More About

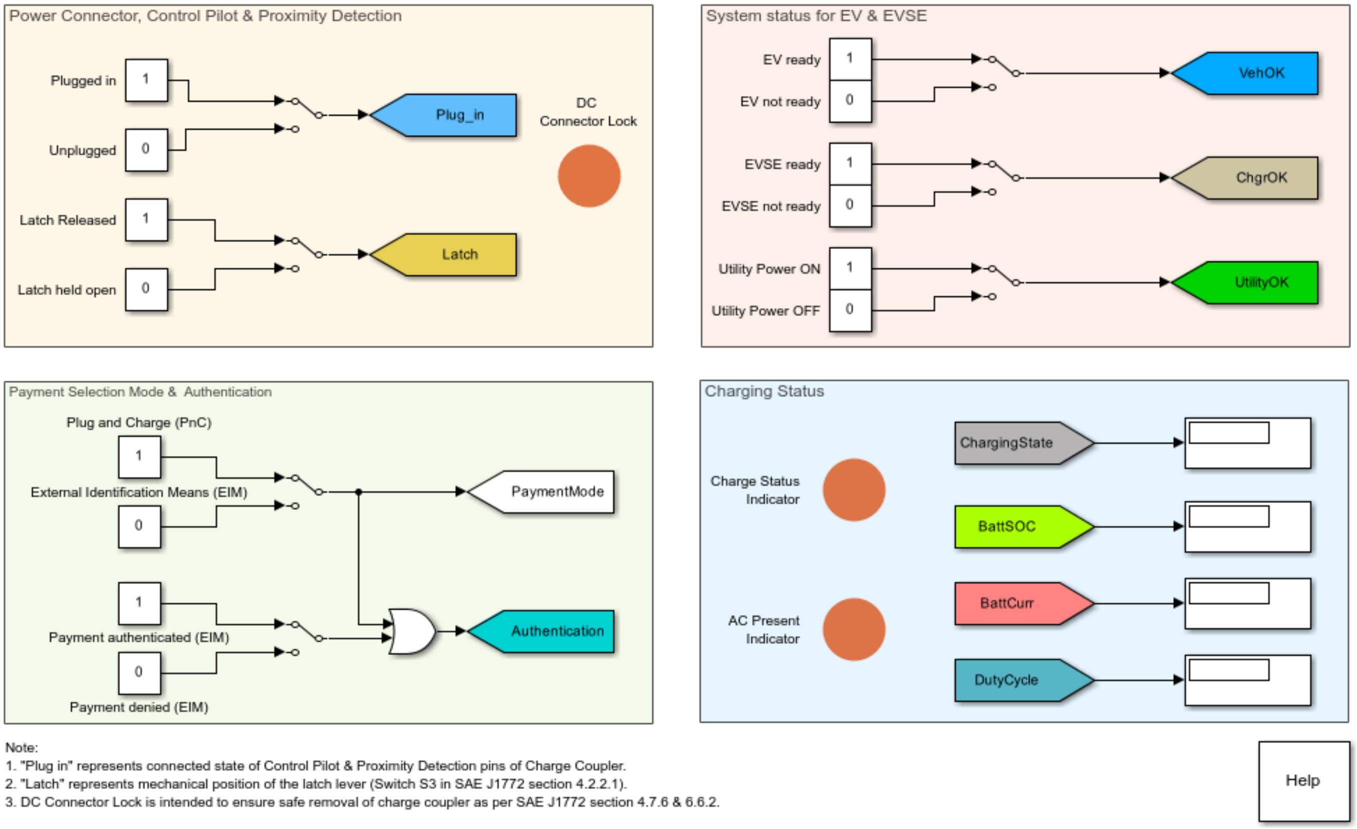
- “Analyze Power and Energy” on page 3-129
- “Variant Systems”

EV Charging Reference Application

The electric vehicle (EV) charging reference application models DC charging of an EV battery from an electric vehicle supply equipment (EVSE) unit, including digital communication between them. The application uses SAE J1772 protocol for basic charging and ISO15118 protocol for high level communication (HLC). The application incorporates a mapped battery block without thermal management.

During the charging process, a dashboard displays the system states and the charging progress.

For more information, see “Charge an Electric Vehicle” on page 3-32.



Note:
1. "Plug in" represents connected state of Control Pilot & Proximity Detection pins of Charge Coupler.
2. "Latch" represents mechanical position of the latch lever (Switch S3 in SAE J1772 section 4.2.2.1).
3. DC Connector Lock is intended to ensure safe removal of charge coupler as per SAE J1772 section 4.7.6 & 6.6.2.

See Also

Interior PMSM | Drive Cycle Source | Mapped Motor

Related Examples

- “Build Full Electric Vehicle Model” on page 3-26

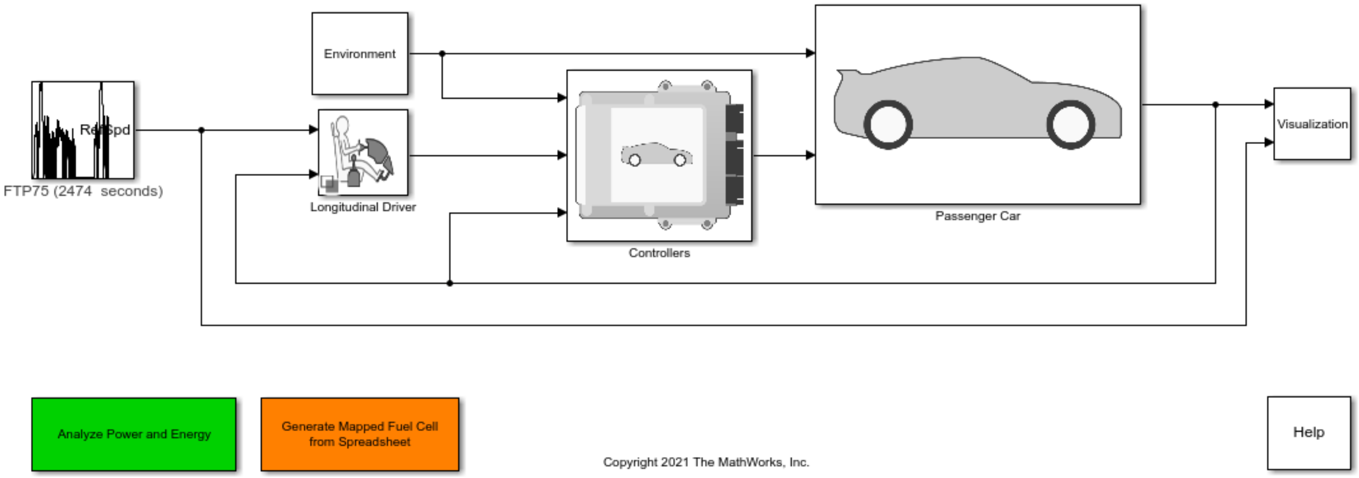
More About

- “Variant Systems”

FCEV Reference Application

The fuel cell electric vehicle (FCEV) reference application represents a fuel cell electric vehicle model with a motor-generator, battery, direct-drive transmission, and associated powertrain control algorithms. Use the fuel cell electric vehicle reference application for powertrain matching analysis and component selection, control and diagnostic algorithm design, and hardware-in-the-loop (HIL) testing.

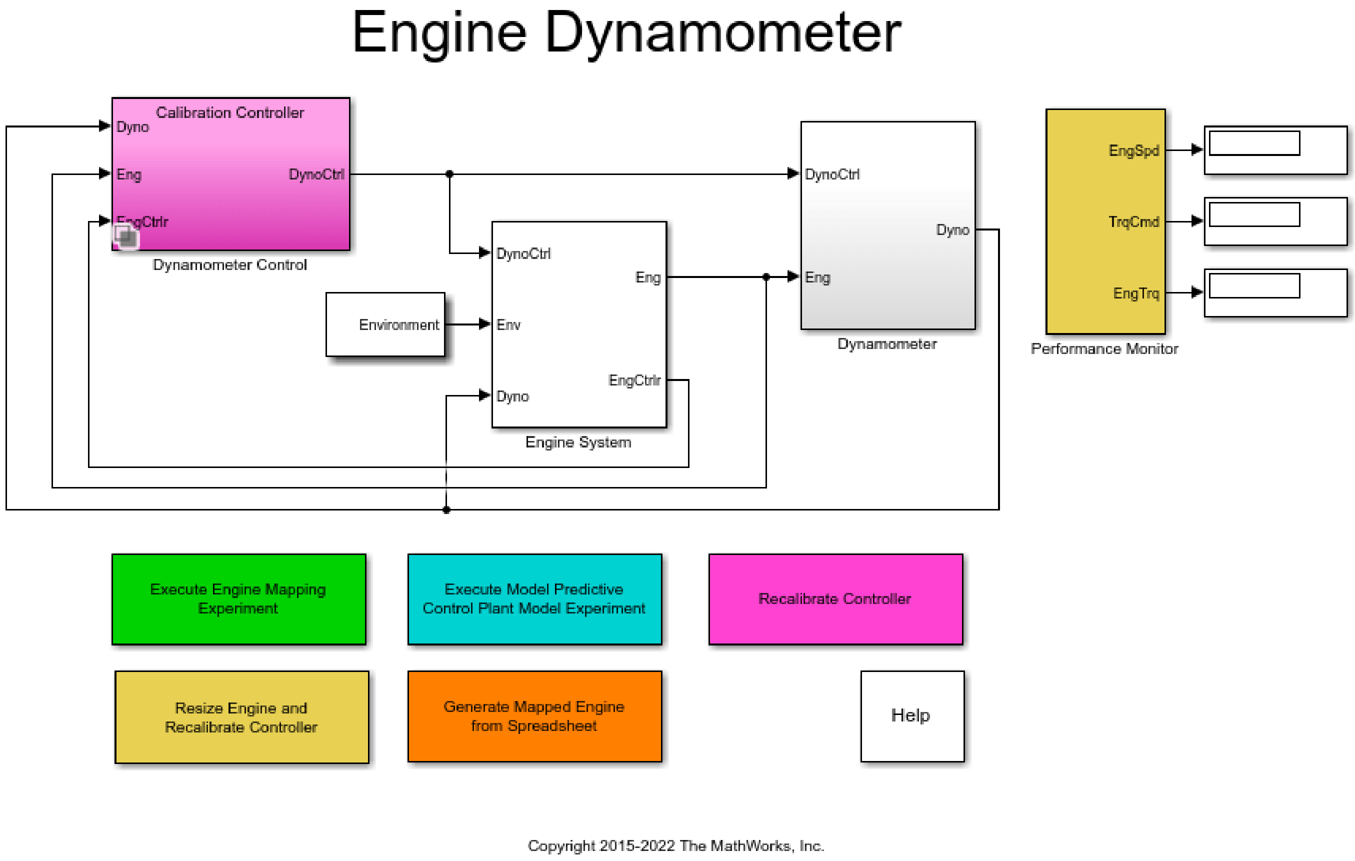
For more information, see “Build Fuel Cell Electric Vehicle” on page 3-37.



CI Engine Dynamometer Reference Application

The compression-ignition (CI) engine dynamometer reference application represents a CI engine plant and controller connected to a dynamometer with a tailpipe emission analyzer. Using the reference application, you can calibrate, validate, and optimize the engine controller and plant model parameters before integrating the engine with the vehicle model.

For more information, see “Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness” on page 3-11.



See Also

Mapped CI Engine | CI Core Engine | CI Controller

More About

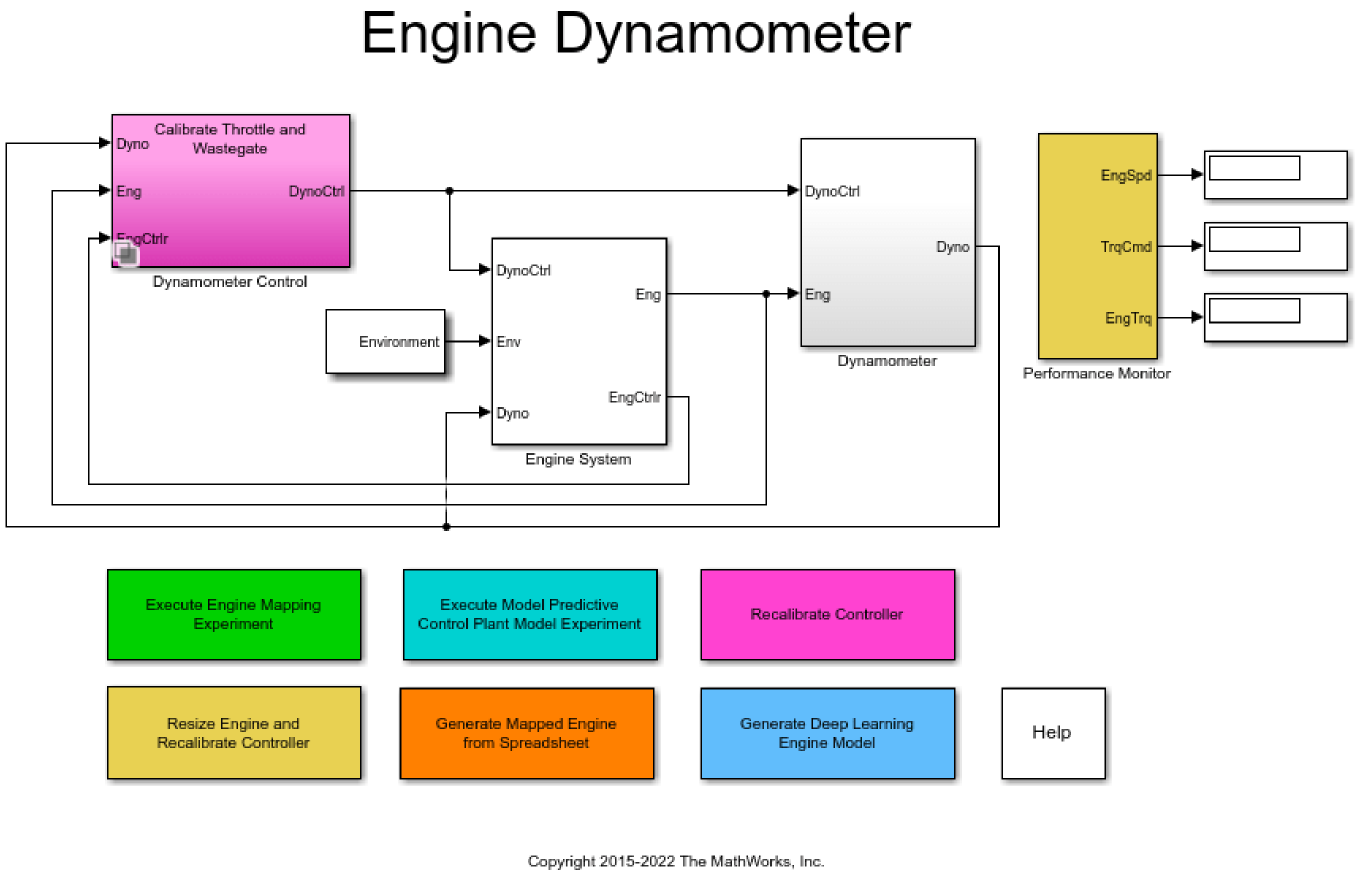
- “CI Engine Project Template” on page 4-2
- “Generate Mapped CI Engine from a Spreadsheet” on page 3-108
- “Resize the CI Engine” on page 3-94
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128

- “Variant Systems”

SI Engine Dynamometer Reference Application

The spark-ignition (SI) engine dynamometer reference application represents an SI engine plant and controller connected to a dynamometer with a tailpipe emission analyzer. Using the reference application, you can calibrate, validate, and optimize the engine controller and plant model parameters before integrating the engine with the vehicle model.

For more information, see “Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15.


Engine Dynamometer
Calibrate Throttle and Wastegate
Dyno
Eng
DynoCtrl
EngCtrlr
Dynamometer Control
Environment
DynoCtrl
Eng
Env
EngCtrlr
Dyno
Engine System
DynoCtrl
Dyno
Eng
Dynamometer
EngSpd
TrqCmd
EngTrq
Performance Monitor
Execute Engine Mapping Experiment
Execute Model Predictive Control Plant Model Experiment
Recalibrate Controller
Resize Engine and Recalibrate Controller
Generate Mapped Engine from Spreadsheet
Generate Deep Learning Engine Model
Help
Copyright 2015-2022 The MathWorks, Inc.

See Also

SI Core Engine | Mapped SI Engine | SI Controller

More About

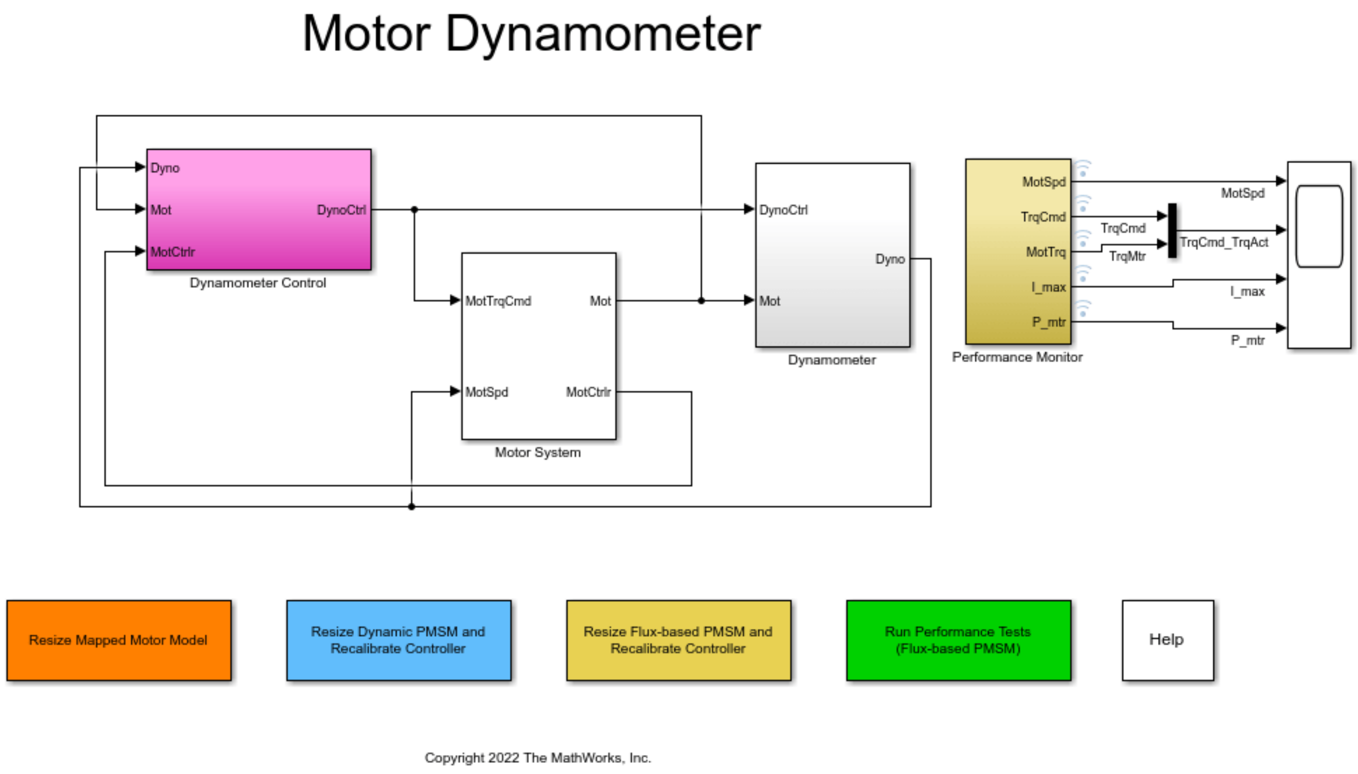
- “Generate Mapped SI Engine from a Spreadsheet” on page 3-113
- “Generate Deep Learning SI Engine Model” on page 3-117
- “Resize the SI Engine” on page 3-101
- “Internal Combustion Mapped and Dynamic Engine Models” on page 3-128

- “Variant Systems”

Motor Dynamometer Reference Application

The motor dynamometer reference application represents a motor controller and plant connected to a dynamometer. Using the reference application, you can calibrate, validate, and optimize the motor controller and plant model parameters before integrating the motor with the vehicle model.

For more information, see “Develop, Resize, and Calibrate Motors with Dynamometer Test Harness” on page 3-89.



See Also

Mapped Motor | Interior PM Controller | Interior PMSM | Flux-Based PM Controller | Flux-Based PMSM

More About

- “Resize Motors” on page 3-92
- “Variant Systems”
- “Estimate Motor Parameters Using Motor Control Blockset Parameter Estimation Tool” (Motor Control Blockset)

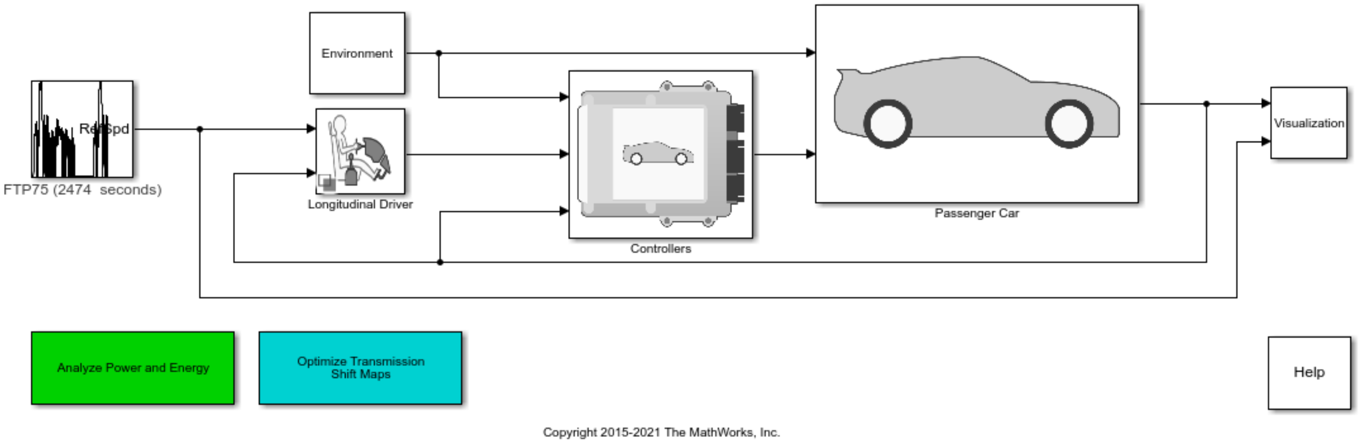
Optimize Transmission Control Module Shift Schedules

This example shows how to use the conventional vehicle reference application to optimize the transmission control module (TCM) shift schedules. Use the optimized shift schedules to:

- Design control algorithms.
- Assess the impact of powertrain changes, such as an engine or gear ratio, on performance, fuel economy, and emissions.

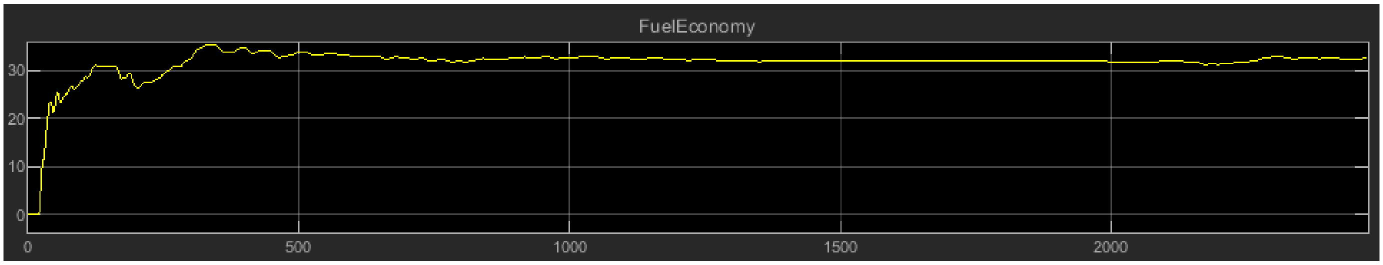
This example uses the Global Optimization Toolbox, Simulink® Design Optimization™, and Stateflow®. To increase optimization performance, consider using the Parallel Computing Toolbox™.

For more information about the reference application, see “Build a Conventional Vehicle Model” on page 3-5.



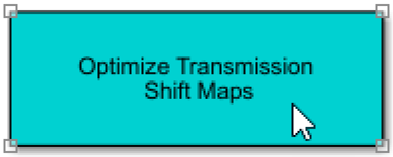
Run Conventional Vehicle Reference Application

Click **Run** to simulate the conventional vehicle reference application with the default settings. The results indicate that the conventional vehicle has a fuel economy of approximately 31 mpg.



Optimize Transmission Shift Maps

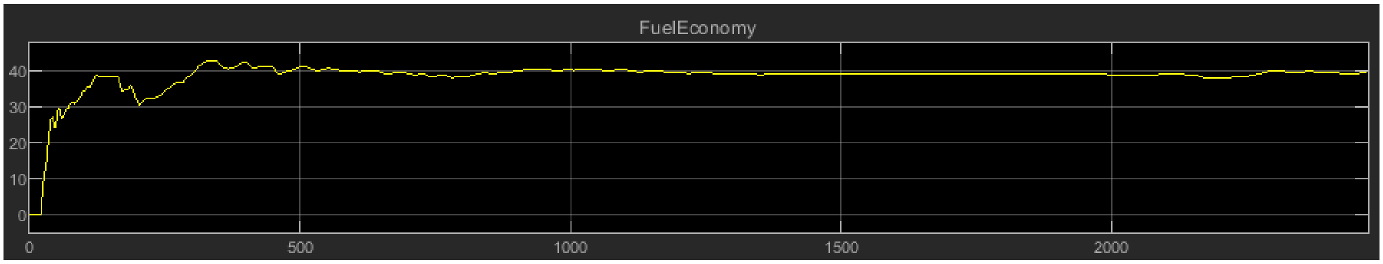
Click **Optimize Transmission Shift Maps**. Optimizing the shift schedules can take time to run. If you have the Parallel Computing Toolbox, the optimization uses parallel workers by default. View the optimization in the MATLAB® window.



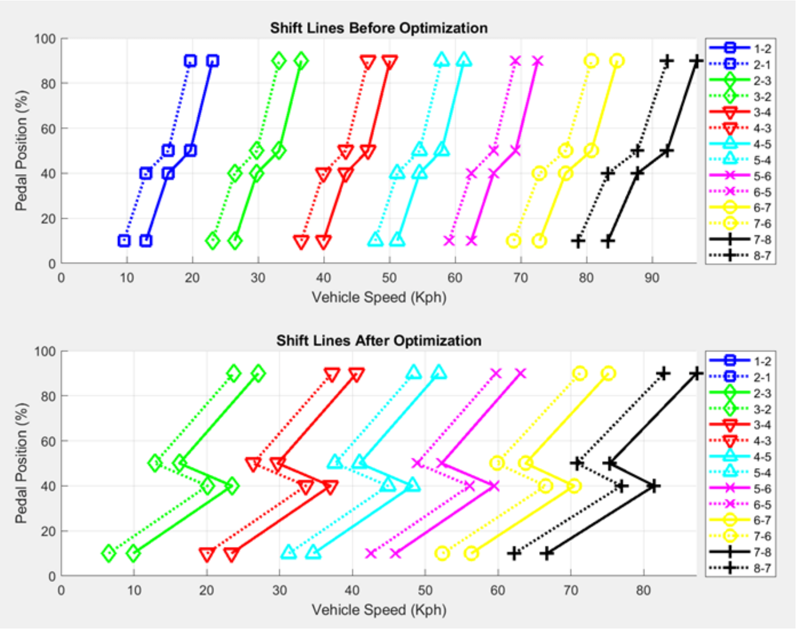
View Results

After you optimize the shift schedule, view the results.

The performance scope indicates that the conventional vehicle with an optimized TCM shift schedule has a fuel economy of approximately 40 mpg.



The figure shows the transmission shift schedule upshift and downshift calibration lines before and after optimization.


Shift Lines Before Optimization

Shift Lines After Optimization

Open Reference Application From Command Line

Use this command to open a version of the conventional vehicle reference application that includes the option to optimize transmission shift maps.

```
autoblkConVehShftOptStart
```

See Also

More About

- “Build a Conventional Vehicle Model” on page 3-5
- “Global Optimization Toolbox”
- “Simulink Design Optimization”
- “Stateflow”

Calibrate ECMS Block for Objective Hybrid Vehicle Fuel Economy Assessment

When you compare fuel economy changes in a hybrid vehicle drive cycle, you must account for the amount of energy stored in the vehicle battery. Otherwise, net battery charging or depletion during the drive cycle can artificially inflate or deflate the fuel economy estimate for the drive cycle. Powertrain Blockset allows you to simulate the fuel economy, performance, and emissions of hybrid electric vehicles with standard hybrid powertrain architectures P0, P1, P2, P3, and P4.

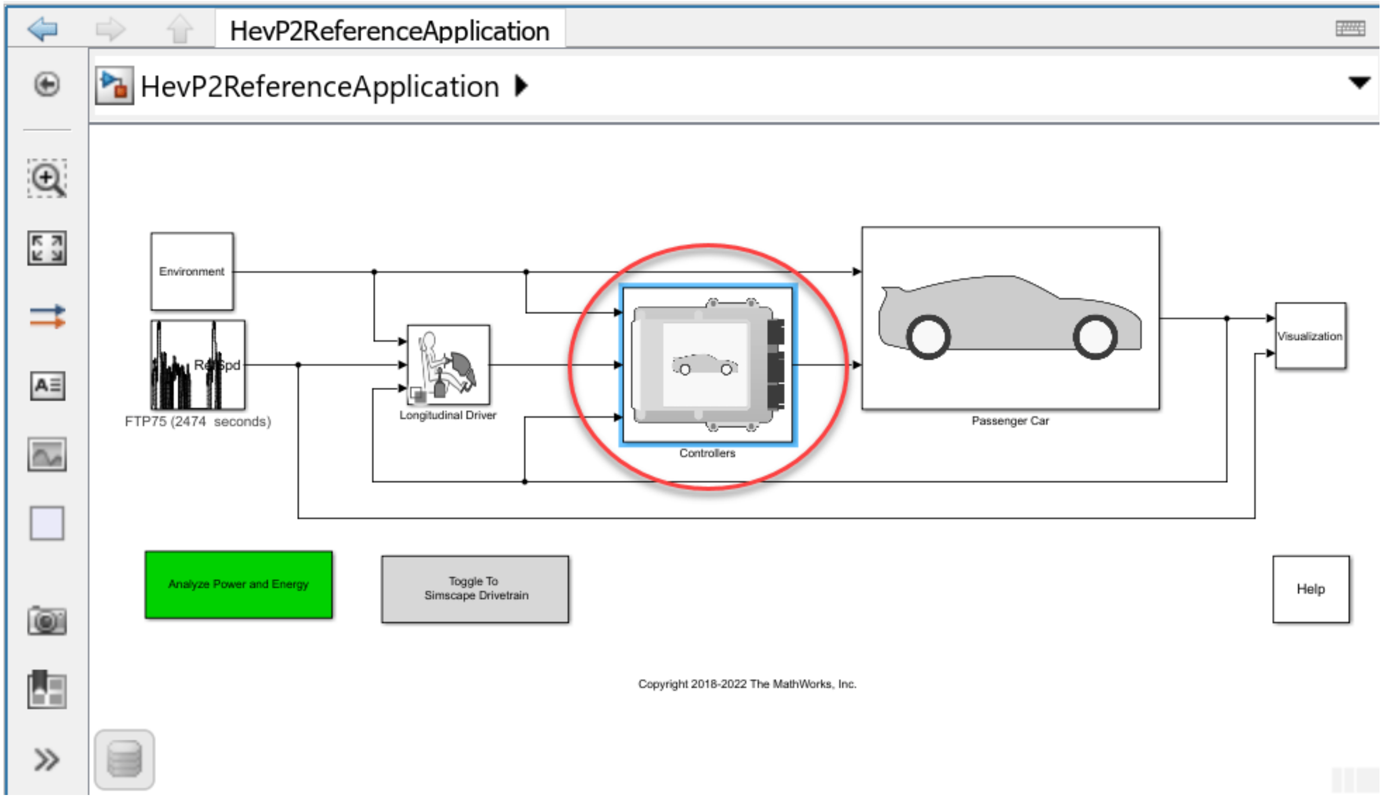
Powertrain Blockset uses an industry-standard equivalent consumption minimization strategy (ECMS) to optimize the fuel economy of hybrid reference applications by optimally trading off internal combustion engine (ICE) use versus electric motor use during a given drive-cycle.

An ECMS block in the hybrid controller of each hybrid reference application optimizes trading off engine and motor use during the cycle. The ECMS block has a parameter in its block mask called **ECMS_s** that you can set so that the hybrid battery state of charge (SOC) at the end of a given drive cycle is the same as the SOC at the beginning of the drive cycle.

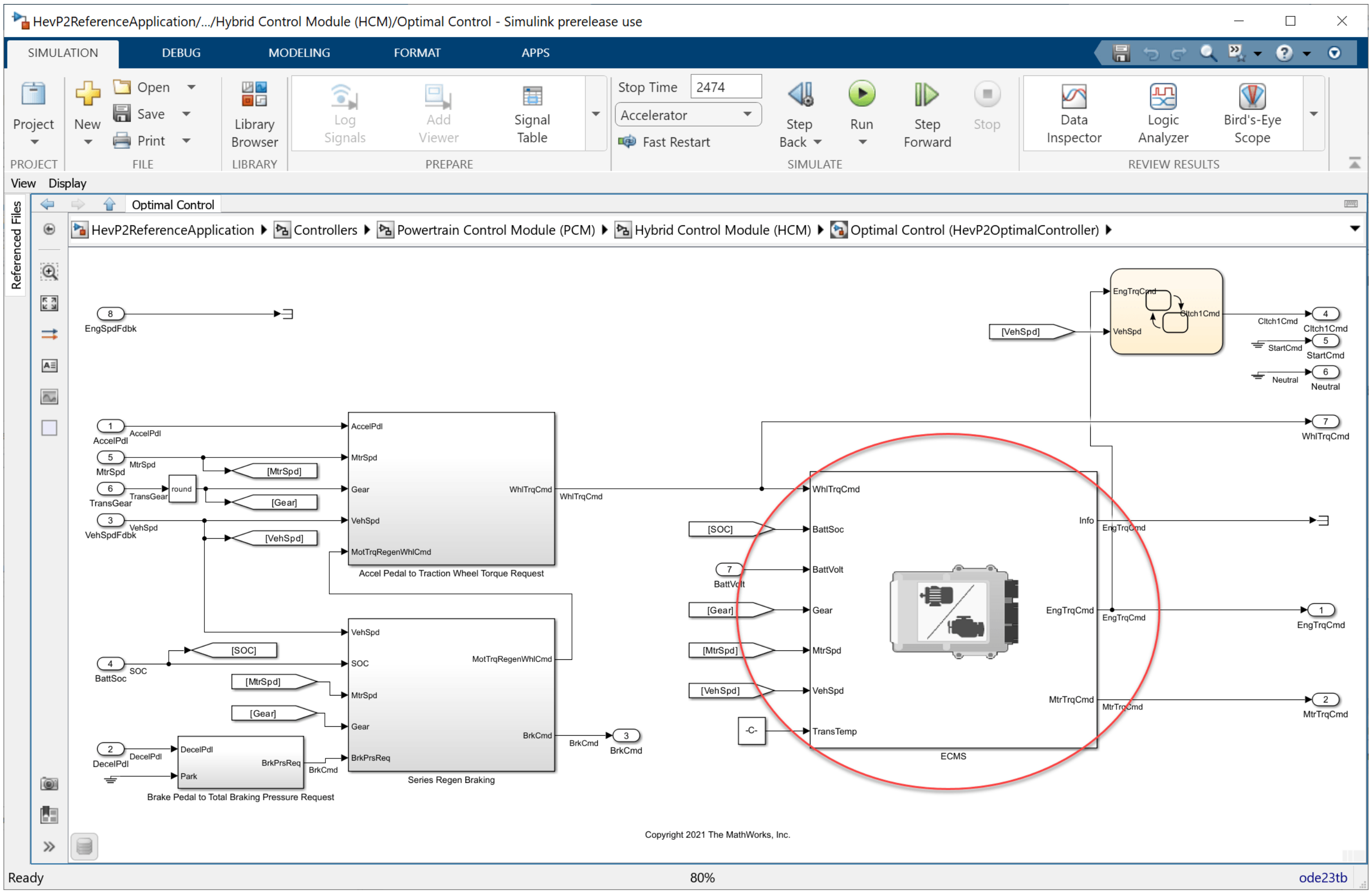
This script optimizes the **ECMS_s** parameter automatically. The script runs a drive-cycle simulation, measures starting and ending SOC, and repeats the process until the difference between starting and ending SOC is minimized.

During the optimization process, an optimization plot indicates the progress of the optimizer toward net zero change in SOC over the given driven cycle for the chosen vehicle configuration.

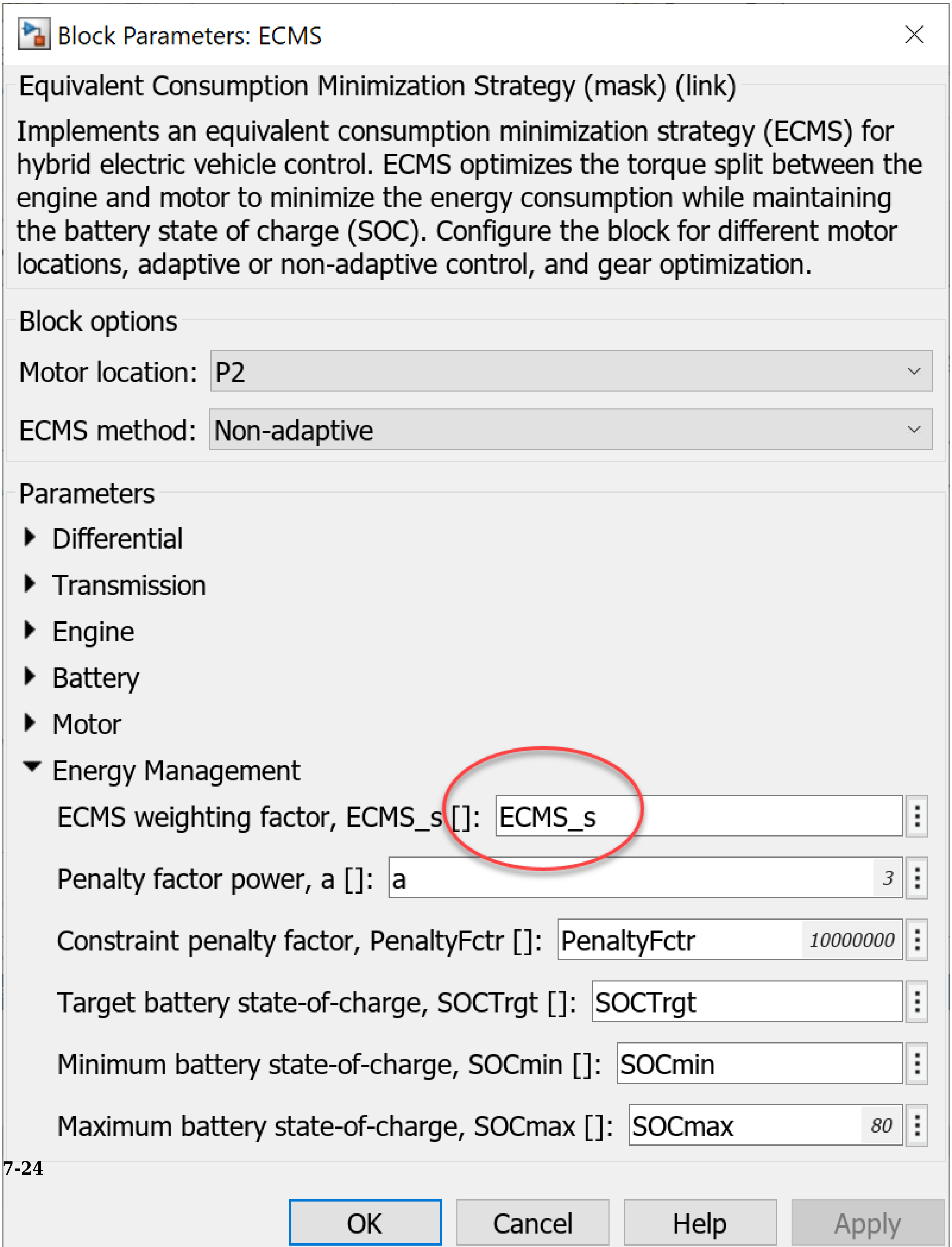
Note that the optimization process may take more than one hour to complete. After optimization completes, the **ECMS_s** parameter for net zero change in cycle SOC is stored in the model workspace of the Optimal Control Model Reference.


HevP2ReferenceApplication
HevP2ReferenceApplication
Environment
RefSpd
FTP75 (2474 seconds)
Longitudinal Driver
Controllers
Passenger Car
Visualization
Analyze Power and Energy
Toggle To Simscape Drivetrain
Help
Copyright 2018-2022 The MathWorks, Inc.

ECMS is in HEV Reference Application Controllers



ECMS Block Helps Achieve Optimal Fuel Economy Results



Block Parameters: ECMS

Equivalent Consumption Minimization Strategy (mask) (link)

Implements an equivalent consumption minimization strategy (ECMS) for hybrid electric vehicle control. ECMS optimizes the torque split between the engine and motor to minimize the energy consumption while maintaining the battery state of charge (SOC). Configure the block for different motor locations, adaptive or non-adaptive control, and gear optimization.

Block options

Motor location: P2

ECMS method: Non-adaptive

Parameters

- Differential
- Transmission
- Engine
- Battery
- Motor
- Energy Management

ECMS weighting factor, ECMS_s []: ECMS_s

Penalty factor power, a []: a 3

Constraint penalty factor, PenaltyFctr []: PenaltyFctr 10000000

Target battery state-of-charge, SOCTrgt []: SOCTrgt

Minimum battery state-of-charge, SOCmin []: SOCmin

Maximum battery state-of-charge, SOCmax []: SOCmax 80

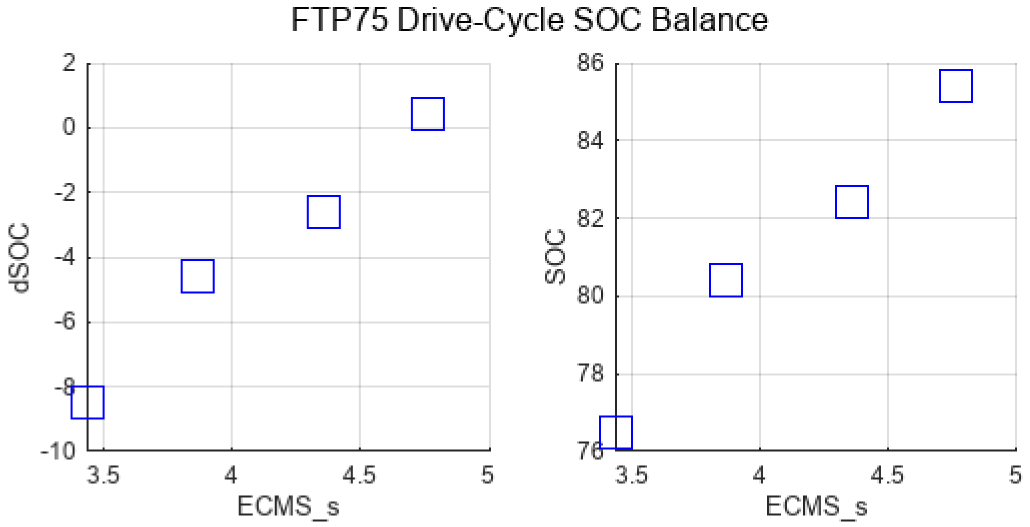
ECMS Block Parameter ECMS_s Ensures Starting and Ending Battery SOC Are Equal

ECMS Calibration Script

Use this code to calibrate the **ECMS_s** block parameter for net zero change in battery SOC for any of the Powertrain Blockset Hybrid Reference Applications P0, P1, P2, P3, and P4.

Choose HEV Reference Application and Run Optimization

```
% Open a given reference application
pxName="P2"; % HEV powertrain architecture configuration: P0,P1,P2,P3,P4
eval("autoblkHev"+pxName+"Start"); %Start a Px Reference Application
SOCinit = 0.85; % initial SOC, unit is in [0, 1]

ECMS_s=Calibrate_ECMS_s(pxName,SOCinit); %Run ECMS_s optimization and return optimal ECMS_s for
```



```
Elapsed time is 420.625345 seconds.
Search converged. ECMS_s parameter updated in model.
ECMS_s = 4.760604, SOCEndTrg = 85.000000, SOC_end = 85.413778
```

Execute ECMS_s Calibration Function

```
function ECMS_s=Calibrate_ECMS_s(pxName,SOCinit)

    % calibration_ecms_script
    battMdl = "BattHev"+pxName; % battery model - for SOC sweep
    mdlName="Hev"+pxName+"ReferenceApplication"; % Hev model
    ctrlMdl="Hev"+pxName+"OptimalController"; % Controller model
    SOCEndTrg = SOCinit*100;
    maxIter=4;

    % Set up optimization progress plot window size and position
    x0=600;
    y0=1040;
    width=600;
    height=280;

    tic
```

```
load_system(mdlName);
load_system(ctrlMdl);
load_system(battMdl);

blk = mdlName + "/" + "Drive Cycle Source";
m = get_param(blk, 'MaskObject');
ECMS_CurrentCycle = m.Parameters(1,1).Value; % Current cycle setting.

mdlWks = get_param(ctrlMdl,'ModelWorkspace');  % find model workspace
% get current ECMS_s value
ECMS_obj = getVariable(mdlWks,'ECMS_s');
if  isnumeric(ECMS_obj); ECMS_CurrentValue = ECMS_obj; end
if ~isnumeric(ECMS_obj); ECMS_CurrentValue = ECMS_obj.Value; end

% extract initial value/name of ECMS tuning parameter
p = Simulink.Mask.get(ctrlMdl+"/ECMS");
baseParamName=p.getParameter("ECMS_s").Value;
battChrgMaxValue_obj = getVariable(get_param(battMdl,'modelworkspace'),'BattChargeMax');
if  isnumeric(battChrgMaxValue_obj); battChrgMaxValue = battChrgMaxValue_obj; end
if ~isnumeric(battChrgMaxValue_obj); battChrgMaxValue = battChrgMaxValue_obj.Value; end

% set to a temp name
set_param(ctrlMdl+"/ECMS",'ECMS_s',"ECMS_s_tune")
save_system(mdlName,[],'SaveDirtyReferencedModels','on');
save_system(ctrlMdl,[],'SaveDirtyReferencedModels','on');

% Enable SOC recording
x1_handles = get_param(mdlName+"/Visualization/Rate Transition1",'PortHandles');
x1 = x1_handles.Outport(1);
Simulink.sdi.markSignalForStreaming(x1,'on');

% arrays used to store ECMS_s and SOC_end values
xc = zeros (3,1);
yc = zeros (3,1);

% plot showing the ECMS_s and SOC_end
subplot (1,2,1);
set(gcf,'position',[x0,y0,width,height])
xlabel({'ECMS\_s',' '})
ylabel('dSOC')
hold on;
subplot (1,2,2);
xlabel('ECMS\_s')
ylabel('SOC')
hold on;
sgtitle([ECMS_CurrentCycle ' Drive-Cycle SOC Balance']);
% set model workspace variables
in = ModelSetVariable (mdlName, battChrgMaxValue, SOCinit,battMdl, SOCEndTrg, ctrlMdl);
% get initial 3 ECMS_s points and SOC_end values
[x, y, ECMS_s, SOC_end, solution_found] = dSOC_generate_starting_points (in, ECMS_CurrentValu

if ~solution_found

    for i = 1 : maxIter

        % solve second order equation to get z = ECMS_s corresponds to y = SOCEndTrg
```

$SOCEndTrg = SOC_{init} = ax^2 + bx + c,$ where a, b, c satisfies $y_i = ax_i^2 + bx_i + c,\ \ 1 \le i \le 3$. with

$x_1 = \mathrm{ECMS}_s^{(1)}, y_1 = SOC_{end}(\mathrm{ECMS}_s^{(1)}),\ \ x_2 = \mathrm{ECMS}_s^{(2)},\ \ y_2 = SOC_{end}(\mathrm{ECMS}_s^{(2)}),\ \ x_3 = \mathrm{ECMS}_s^{(3)},\ \ y_3 = SOC_{end}(\mathrm{E}$

```
[z] = Second_order_roots (x, y, SOCEndTrg);
in = in.setVariable("ECMS_s_tune", z);
[SOC_end, dSOC] =dSOCsim_v1(in);
subplot (1,2,1);
plot(z,dSOC,'bs','MarkerSize',15)
grid on
hold on

subplot (1,2,2);
plot(z,SOC_end,'bs','MarkerSize',15)
grid on
hold on

if (abs(SOC_end - SOCEndTrg) <= 1) % check if a solution is found
    ECMS_s = z;
    x(1) = ECMS_s;
    y(1) = SOC_end;
    solution_found = 1;
    break;
end

xc(1:3) = x (1:3); yc(1:3) = y (1:3);
x(1) = z; y(1) = SOC_end; % since z and SOC_end are new points, we use them and put i
[yb, II] = sort(yc,'ascend'); % sort the array for processing
xb = xc(II);

% begin the process to pick other two points from the original 3 point
% xb(1:3), yb(1:3)

if (y(1) > SOCEndTrg) % y(1)> SOCEndTrg, we need to pick other points < SOCEndTrg
    if ((yb(2) < SOCEndTrg) && (SOCEndTrg < yb(3)))  % yb(2) < SOCEndTrg, pick yb(2)
        x(2) = xb(2); y(2) = yb(2);
        if (abs(yb(1)-SOCEndTrg) < abs(yb(3)-SOCEndTrg)) % pick last point from xb(1
            x(3) = xb(1); y(3) = yb(1);
        else
            x(3) = xb(3); y(3) = yb(3);
        end
    end
    if ((yb(1) < SOCEndTrg) && (SOCEndTrg < yb(2))) % yb(1) < SOCEndTrg, pick yb(1)
        x(2) = xb(1); y(2) = yb(1);
        if (abs(yb(2)-SOCEndTrg) < abs(yb(3)-SOCEndTrg)) % pick last point from xb(2
            x(3) = xb(2); y(3) = yb(2);
        else
            x(3) = xb(3); y(3) = yb(3);
        end
    end
end
if (y(1) < SOCEndTrg) % y(1)< SOCEndTrg, we need to pick other points > SOCEndTrg
    if ((yb(2) < SOCEndTrg) && (SOCEndTrg < yb(3))) % yb(3) > SOCEndTrg, pick yb(3)
        x(2) = xb(3); y(2) = yb(3);
        if (abs(yb(1)-SOCEndTrg) < abs(yb(2)-SOCEndTrg)) % pick last point from xb(1
            x(3) = xb(1); y(3) = yb(1);
        else
```

```
                        x(3) = xb(2); y(3) = yb(2);
                    end
                end
                if ((yb(1) < SOCEndTrg) && (SOCEndTrg < yb(2))) % yb(2) > SOCEndTrg, pick this po
                    x(2) = xb(2); y(2) = yb(2);
                    if (abs(yb(1)-SOCEndTrg) < abs(yb(3)-SOCEndTrg)) % pick last point from xb(1
                        x(3) = xb(1); y(3) = yb(1);
                    else
                        x(3) = xb(3); y(3) = yb(3);
                    end
                end
            end
        end

        y_abs = abs(y-SOCEndTrg);
        [yb, II] = sort(y_abs,'ascend');
        xb = x(II);
        ECMS_s = xb (1);

    end

    hold off;

    toc;

    if solution_found
        fprintf ('Search converged. ECMS_s parameter updated in model. \n');
        fprintf ('ECMS_s = %f, SOCEndTrg = %f, SOC_end = %f \n',ECMS_s, SOCEndTrg, SOC_end);
    else
        fprintf ('Search failed to converge. An approximate ECMS_s is updated in the model. \n')
        fprintf ('Refer to the Troubleshooting section of the example page for recommendations. \
    end

    ECMS_s_tune = ECMS_s;
    % reset model
    Simulink.sdi.markSignalForStreaming(x1,'off');
    load_system(ctrlMdl)
    set_param(ctrlMdl+"/ECMS",'ECMS_s',baseParamName)
    save_system(mdlName,[],'SaveDirtyReferencedModels','on');

    % update model and sim
    hws = get_param(ctrlMdl, 'modelworkspace');% get the workspace
    hws.assignin('ECMS_s',ECMS_s);
    save_system(ctrlMdl,[],'SaveDirtyReferencedModels','on');
    open_system(mdlName);
    load_system(battMdl);
    battChrgMaxValue_obj = getVariable(get_param(battMdl,'modelworkspace'),'BattChargeMax');
    if  isnumeric(battChrgMaxValue_obj); battChrgMaxValue = battChrgMaxValue_obj; end
    if ~isnumeric(battChrgMaxValue_obj); battChrgMaxValue = battChrgMaxValue_obj.Value; end
    in = in.setVariable('BattCapInit', battChrgMaxValue*SOCinit,'Workspace',battMdl);
    in = in.setVariable('SOCTrgt', SOCEndTrg,'Workspace',ctrlMdl);
    in = in.setVariable('SOCmin', max(SOCEndTrg-20,20.5),'Workspace',ctrlMdl);
    in = in.setVariable('SOCmax', min(SOCEndTrg+20,100),'Workspace',ctrlMdl);
    set_param(ctrlMdl+"/ECMS",'ECMS_s',"ECMS_s");
    save_system(battMdl);
    save_system(ctrlMdl);

end
```

```
function in = ModelSetVariable (mdlName, battChrgMaxValue, SOCinit, battMdl, SOCEndTrg, ctrlMdl)
in = Simulink.SimulationInput(mdlName);
in = in.setVariable('BattCapInit', battChrgMaxValue*SOCinit,'Workspace',battMdl);
in = in.setVariable('SOCTrgt', SOCEndTrg,'Workspace',ctrlMdl);
in = in.setVariable('SOCmin', max(SOCEndTrg-20,20.5),'Workspace',ctrlMdl);
in = in.setVariable('SOCmax', min(SOCEndTrg+20,100),'Workspace',ctrlMdl);
end

%%
function [x_out, y_out, ECMS_s, SOC_end, solution_found] = dSOC_generate_starting_points (in, ECM

x_out = zeros (3,1);
y_out = zeros (3,1);
x = zeros (4,1);
y = zeros (4,1);

solution_found = false;
ECMS_s = ECMS_CurrentValue;
alpha = 0.8;

tol = 1;

x1 = ECMS_CurrentValue;  % use current ECMS_s value as the starting point
in = in.setVariable("ECMS_s_tune", x1);
[SOC_end,  dSOC] =dSOCsim_v1(in);  %evaluate x1 results
y1 = SOC_end;
subplot (1,2,1);
plot(x1,dSOC,'bs','MarkerSize',15)
grid on
hold on

subplot (1,2,2);
plot(x1,y1,'bs','MarkerSize',15)
grid on
hold on

if (abs(y1 - SOCEndTrg) <= tol)  % check if a solution found
    ECMS_s  = x1;
    SOC_end = y1;
    solution_found = true;
end

if ((y1 > SOCEndTrg) && ~solution_found)
    x2 = x1 - ECMS_S_dis_up (x1, y1, SOCEndTrg); % use a decreased ECMS_s value as x2
    in = in.setVariable("ECMS_s_tune", x2);
    [SOC_end, dSOC] =dSOCsim_v1(in);
    y2 = SOC_end;

    subplot (1,2,1);
    plot(x2,dSOC,'bs','MarkerSize',15)
    grid on
    hold on

    subplot (1,2,2);
    plot(x2,y2,'bs','MarkerSize',15)
```

```
grid on
hold on


if (abs(y2 - SOCEndTrg) <= tol)  % check if a solution found
    ECMS_s  = x2;
    SOC_end = y2;
    solution_found = true;
end
if ((y2 > SOCEndTrg) && ~solution_found) % y2 is still too high
    y3_try = SOCEndTrg - 2; % set a lower SOC end trial to it will be easier to get y2 < SOC_
    x3 = x1 + alpha*(y3_try - y1)*(x2-x1)/(y2-y1); % use x1, y1, x2, y2 to fit a linear func
```

$$x = x_1 + \alpha \frac{y - y_1}{y_2 - y_1} \times (x_2 - x_1) \text{ such that } x = x_1 \text{ at } y = y_1, \text{ and } x = \alpha x_2 + (1 - \alpha) x_1 \text{ at } y = y_2$$

```
    dx = ECMS_S_dis_up (x2, y2, SOCEndTrg); % get a pre-defined decrease value
    x3 = min (x3, x2 -   dx);  % upper limit for x3
    x3 = max (x3, x2 - 2*dx);  % lower limit for x3
    in = in.setVariable("ECMS_s_tune", x3); % set the x3 value as ECMS_s
    [SOC_end,  dSOC] =dSOCsim_v1(in);       % evaluate
    y3 = SOC_end;
    subplot (1,2,1);
    plot(x3,dSOC,'bs','MarkerSize',15)
    grid on
    hold on

    subplot (1,2,2);
    plot(x3,y3,'bs','MarkerSize',15)
    grid on
    hold on

    if (abs(y3 - SOCEndTrg) <= tol) % check if a solution is found
    ECMS_s  = x3;
    SOC_end = y3;
    solution_found = true;
    end
    if ~solution_found
        x_out(1) = x1; x_out(2) = x2; x_out(3) = x3; y_out(1) = y1; y_out(2) = y2; y_out(3) =
    end
end
if ~solution_found
    if ((y2 > SOCEndTrg) && (y3 >= SOCEndTrg)) % if y3 is still greater than SOCEndTrg, lowe
        for it_again = 1 : 10
            [x(4)] = Second_order_roots (x_out, y_out, SOCEndTrg - 2);  % decrease again
            dx = ECMS_S_dis_up (x_out (3), y_out (3), SOCEndTrg);   % standard decrease
            x(4) = max (x(4), x_out (3) -   dx);             % make it higher than the standa
            x(4) = min (x(4), x_out (3) - 2*dx);             % limit the increase to 2 times
            in = in.setVariable("ECMS_s_tune", x(4));
            [SOC_end, ~] = dSOCsim_v1(in);
            ECMS_s = x(4);
            y(4) = SOC_end;
            x_out (1) = x_out (2);   y_out (1) = y_out (2);
            x_out (2) = x_out (3);   y_out (2) = y_out (3);
            x_out (3) = x(4);        y_out (3) = SOC_end;
            if (abs(y(4) - SOCEndTrg) <= tol); break; end
            if (SOC_end <= SOCEndTrg); break; end
        end
```

```
                    if (abs(y(4) - SOCEndTrg) <= tol)
                        ECMS_s  = x(4);
                        SOC_end = y(4);
                        solution_found = true;
                    end
                end
            end

            if ((y2 < SOCEndTrg) && ~solution_found) % y2 is lower than SOCEndTrg while y1 is greater tha
                x_min = min (x1, x2); x_max = max (x1, x2); y_min = min(y1, y2); y_max = max (y1, y2); %
                w_min = abs (y_min - SOCEndTrg) / (abs(y_min - SOCEndTrg) + abs(y_max - SOCEndTrg)); % we
                x3 = (1-w_min) * x_min + w_min * x_max; % x3 is a weighted interpolation between x1 and x
                in = in.setVariable("ECMS_s_tune", x3);
                [SOC_end,  dSOC] =dSOCsim_v1(in);
                y3 = SOC_end;

                subplot (1,2,1);
                plot(x3,dSOC,'bs','MarkerSize',15)
                grid on
                hold on

                subplot (1,2,2);
                plot(x3,y3,'bs','MarkerSize',15)
                grid on
                hold on

                if (abs(y3 - SOCEndTrg) <= tol)
                ECMS_s  = x3;
                SOC_end = y3;
                solution_found = true;
                end
            end
            if ~solution_found
                x_out(1) = x1; x_out(2) = x2; x_out(3) = x3; y_out(1) = y1; y_out(2) = y2; y_out(3) = y3
            end
        end

        if ((y1 < SOCEndTrg) && ~solution_found) % y1 < SOCEndTrg
            x2 = x1 + ECMS_S_dis_up (x1, y1, SOCEndTrg); % use a higher ECMS_s value as x2
            in = in.setVariable("ECMS_s_tune", x2);
            [SOC_end, dSOC] =dSOCsim_v1(in);
            y2 = SOC_end;
            subplot (1,2,1);
            plot(x2,dSOC,'bs','MarkerSize',15)
            grid on

            subplot (1,2,2);
            plot(x2,y2,'bs','MarkerSize',15)
            grid on
            hold on

            if (abs(y2 - SOCEndTrg) <= tol)
                ECMS_s  = x2;
                SOC_end = y2;
                solution_found = true;
            end
            if ((y2 < SOCEndTrg) && ~solution_found) % x2 is not high enough
            y3_try = SOCEndTrg + 2;                    % set a higher SOC target
```

$$x = x_1 + \alpha\frac{y - y_1}{y_2 - y_1}(x_2 - x_1),\ \ x = x_1,\ \text{ at }\ y = y_1, x = \alpha x_2 + (1 - \alpha)x_1\ \text{ at }\ y = y_2,$$

```
            x3 = x1 + alpha*(y3_try - y1)*(x2-x1)/(y2-y1); % use x1, y1, x2, y2 as a linear prdeictio
            dx = ECMS_S_dis_up (x2, y2, SOCEndTrg);    % standard increase
            x3 = max (x3, x2 +   dx);                  % lower limit for x3
            x3 = min (x3, x2 + 2*dx);                  % upper limit for x3
            in = in.setVariable("ECMS_s_tune", x3);
            [SOC_end, dSOC] =dSOCsim_v1(in);
            y3 = SOC_end;
            subplot (1,2,1);
            plot(x3,dSOC,'bs','MarkerSize',15)
            grid on

            subplot (1,2,2);
            plot(x3,y3,'bs','MarkerSize',15)
            grid on

            if (abs(y3 - SOCEndTrg) <= tol)
                ECMS_s  = x3;
                SOC_end = y3;
                solution_found = true;
            end

            if ~solution_found
                x_out(1) = x1; x_out(2) = x2; x_out(3) = x3; y_out(1) = y1; y_out(2) = y2; y_out(3) =
            end
        end
        if ~solution_found
            if ((y2 < SOCEndTrg) && (y3 <= SOCEndTrg))  % y3 is still not high enough
                for it_again = 1 : 10
                    [x(4)] = Second_order_roots (x_out, y_out, SOCEndTrg + 2);  % increase again
                    dx = ECMS_S_dis_up (x_out (3), y_out (3), SOCEndTrg);   % standard increase
                    x(4) = max (x(4), x_out (3) +   dx);             % make it higher than the standa
                    x(4) = min (x(4), x_out (3) + 2*dx);             % limit the increase to 2 times
                    in = in.setVariable("ECMS_s_tune", x(4));
                    [SOC_end, dSOC] = dSOCsim_v1(in);
                    ECMS_s = x(4);
                    y(4) = SOC_end;
                    x_out (1) = x_out (2);   y_out (1) = y_out (2);
                    x_out (2) = x_out (3);   y_out (2) = y_out (3);
                    x_out (3) = x(4);        y_out (3) = SOC_end;
                    if (SOC_end >= SOCEndTrg); break; end
                    if (abs(y(4) - SOCEndTrg) <= tol); break; end
                end

                subplot (1,2,1);
                plot(x(4),dSOC,'bs','MarkerSize',15)
                grid on

                subplot (1,2,2);
                plot(x(4),y(4),'bs','MarkerSize',15)
                grid on

                if (abs(y(4) - SOCEndTrg) <= tol)
                ECMS_s  = x(4);
                SOC_end = y(4);
                solution_found = true;
            end
        end
    end
    if ((y2 > SOCEndTrg) && ~solution_found)  % y2 is good
        x_min = min (x1, x2); x_max = max (x1, x2); y_min = min(y1, y2); y_max = max (y1, y2); %
        w_min = abs (y_min - SOCEndTrg) / (abs(y_min - SOCEndTrg) + abs(y_max - SOCEndTrg));   %
        x3= (1-w_min) * x_min + w_min * x_max;  % get weighted interpolation between x1, and x2
        in = in.setVariable("ECMS_s_tune", x3);
        [SOC_end, dSOC] =dSOCsim_v1(in);
        y3 = SOC_end;
        subplot (1,2,1);
        plot(x3,dSOC,'bs','MarkerSize',15)
        grid on

        subplot (1,2,2);
        plot(x3,y3,'bs','MarkerSize',15)
        grid on

        if (abs(y3 - SOCEndTrg) <= tol)
            ECMS_s  = x3;
            SOC_end = y3;
            solution_found = true;
        end
    end
    if ~solution_found
        x_out(1) = x1; x_out(2) = x2; x_out(3) = x3; y_out(1) = y1; y_out(2) = y2; y_out(3) = y3
    end
end

end

function [SOC_end, dSOC]=dSOCsim_v1(in)

    evalc('out=sim(in)'); %Simulate suppressing build info

    if isempty(out.ErrorMessage)
        SOC=out.logsout.getElement('Battery SOC').Values.Data;
        dSOC=SOC(end)-SOC(1);
        SOC_end = SOC(end);
    else
        dSOC=NaN;
    end

end

function dx = ECMS_S_dis_up (ECMS_s, y, SOC_target)

% function computes standard dx = increase/decrease in ECMS_s
% the dx is proportional to the distance between SOC_end (y) and SOC_Target
% also, dx is proportional to ECMS_s, a, y, ff can be adjusted

ff = 0.05;
a  = 0.015;
c = 3.5;
b = ECMS_s / c;   % ECMS_s effect
```

```
if (ECMS_s < c-0.1); b = 1; end
dx = a + b * abs(y-SOC_target) * ff;

end

function [z] = Second_order_roots (x, y, y_tar)
```

solve the equation $y_{tar} = az^2 + bz + c$, with $y_i = ax_i^2 + bx_i + c$, for $1 \le i \le 3$.

```
d1 = y(1) / (x(1) - x(2)) / (x(1) - x(3));
d2 = y(2) / (x(2) - x(1)) / (x(2) - x(3));
d3 = y(3) / (x(3) - x(1)) / (x(3) - x(2));

AA = (d1 + d2 + d3);
BB = - (d1 * (x(2) + x(3)) + d2 * (x(1) + x(3)) + d3 * (x(1) + x(2)));
CC = d1 * x(2) * x(3) + d2 * x(1) * x(3) + d3 * x(1) * x(2) - y_tar;

z1 = (-BB + sqrt (BB*BB - 4 * AA * CC)) / 2 / AA;
z = real(z1);

end
```

Copyright 2020-2022, The MathWorks, Inc.

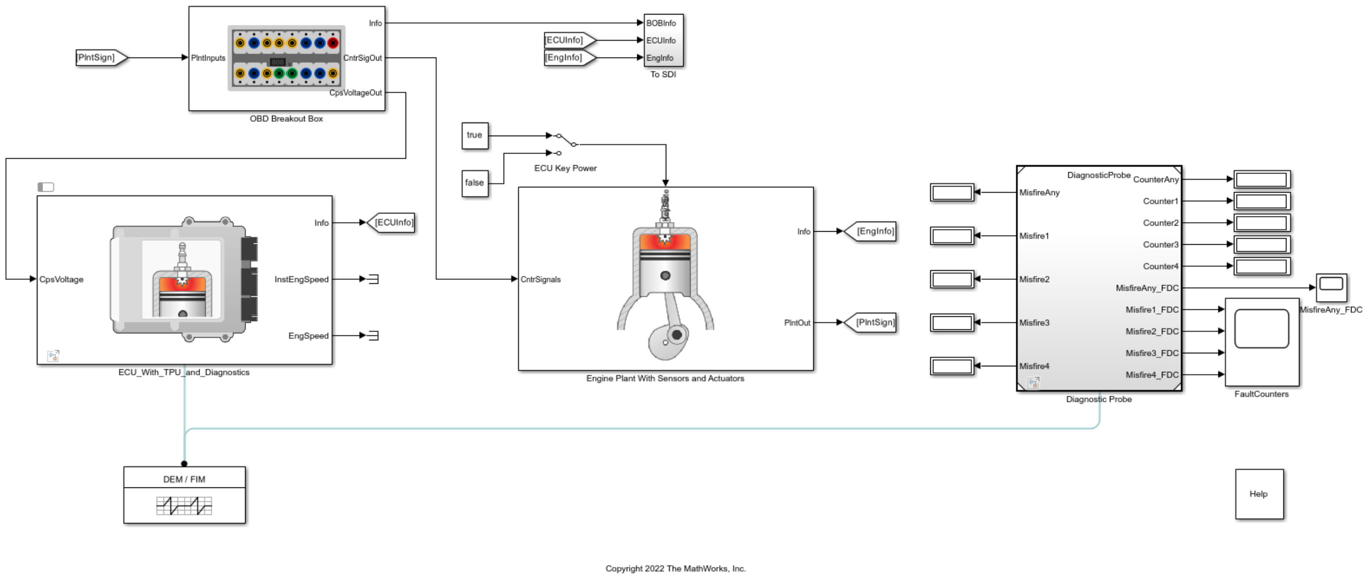
See Also

Equivalent Consumption Minimization Strategy

Detect Misfires Using On-Board Diagnostics

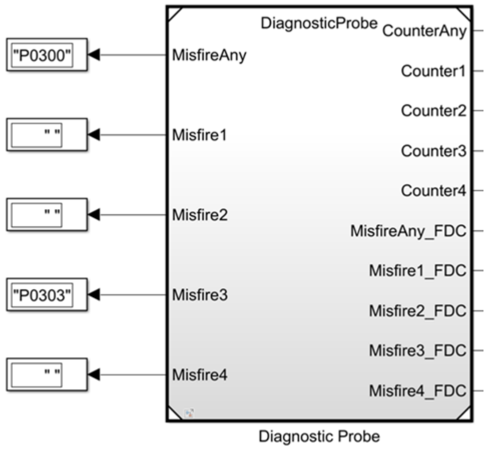
This example shows how to implement onboard-diagnostics (OBD) and detect engine misfires. The example uses Powertrain Blockset™ and AUTOSAR Blockset to introduce, detect, mature, and report misfire events from the SI Core Engine.

Model



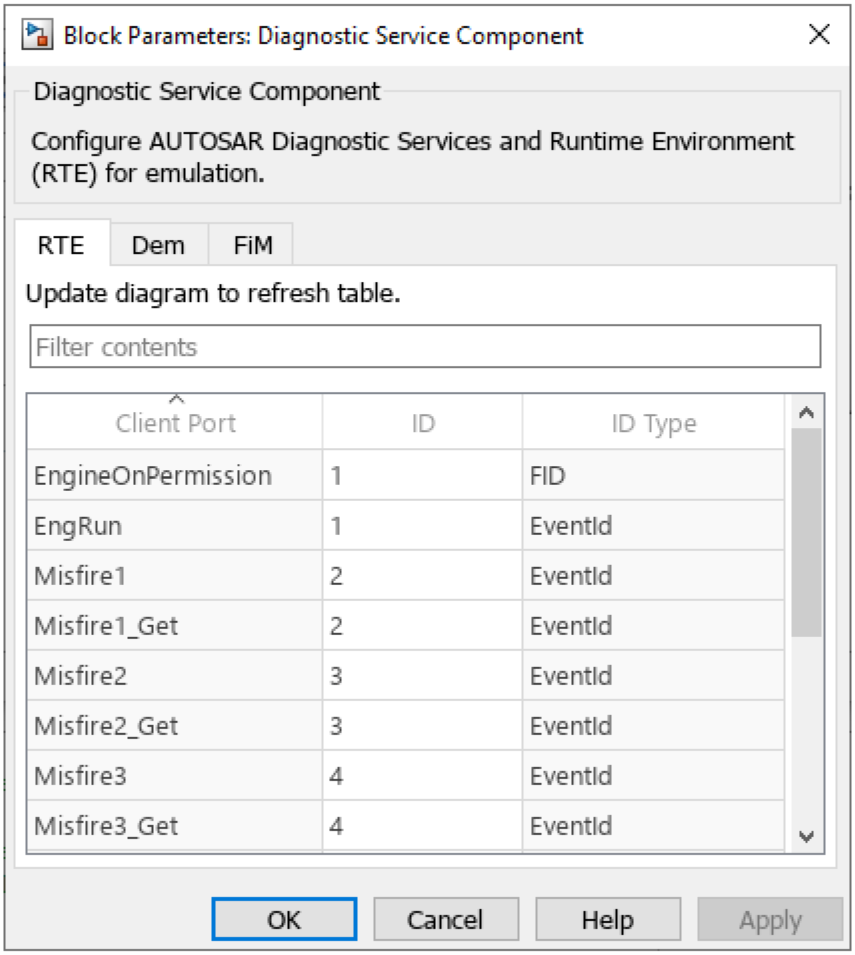
Run Simulation

On the Simulation tab, click **Run**. As the simulation runs, the Diagnostic Probe output displays trouble codes appear. P0300 is a general warning for a misfire issue in one or more cylinders. P0303 indicates a specific misfire issue with Cylinder 3.



Diagnostic Service Component

The Diagnostic Service Component (AUTOSAR Blockset) configures the AUTOSAR Diagnostic Event Manager (Dem) service functions. The Diagnostic Event manager associates the client ports for engine events, including cylinder misfires, to diagnostic Event IDs.



Block Parameters: Diagnostic Service Component

Diagnostic Service Component

Configure AUTOSAR Diagnostic Services and Runtime Environment (RTE) for emulation.

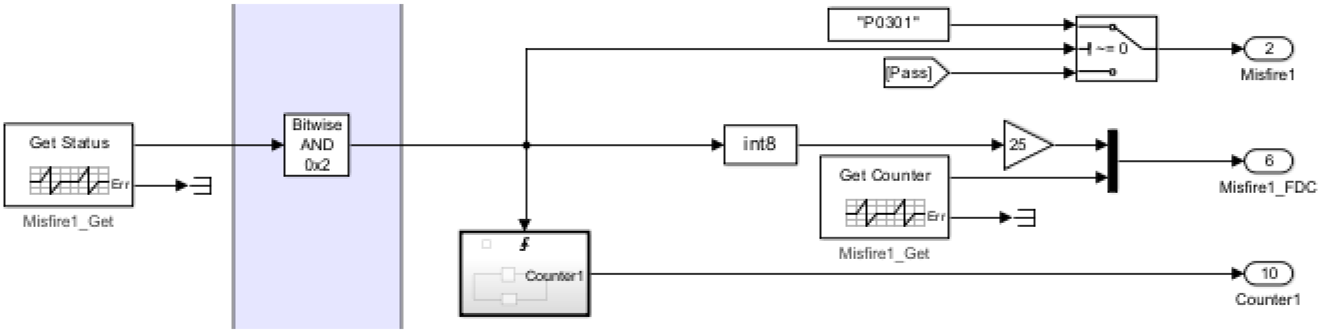
Update diagram to refresh table.

Filter contents

Client Port	ID	ID Type
EngineOnPermission	1	FID
EngRun	1	EventId
Misfire1	2	EventId
Misfire1_Get	2	EventId
Misfire2	3	EventId
Misfire2_Get	3	EventId
Misfire3	4	EventId
Misfire3_Get	4	EventId

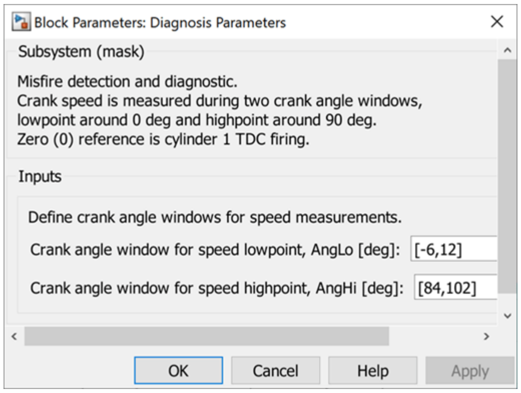
Diagnostic Probe

The Diagnostic Probe model reference uses DiagnosticInfoCaller (AUTOSAR Blockset) blocks to call AUTOSAR Dem service functions. The Get Status blocks get the diagnostic status for the four engine cylinders. The Get Counter blocks get the fault detection counter for the engine cylinders.



ECU With TPU and Diagnostics

This subsystem detects misfire events, using crankshaft acceleration as an indicator. Navigate to autoblkSICoreOBDMisfire > ECU_With_TPU_and_Diagnostics > TPU Level Misfire Detection. Open the Diagnosis Parameters block. You can modify the OBD threshold calibrations for detection and maturation.



Block Parameters: Diagnosis Parameters

Subsystem (mask)

Misfire detection and diagnostic.
Crank speed is measured during two crank angle windows,
lowpoint around 0 deg and highpoint around 90 deg.
Zero (0) reference is cylinder 1 TDC firing.

Inputs

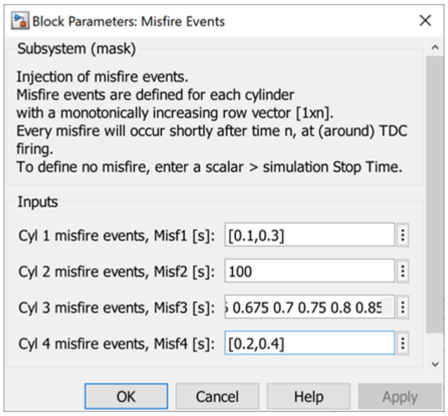
Define crank angle windows for speed measurements.

Crank angle window for speed lowpoint, AngLo [deg]: [-6,12]

Crank angle window for speed highpoint, AngHi [deg]: [84,102]

OBD Breakout Box

This subsystem defines misfire events to inject into the simulation. Navigate to autoblkSICoreOBDMisfire > OBD Breakout Box > Injector Pulsewidth Fault Injection. Open the Misfire Events block. You can modify the misfire fault injection timing.



Block Parameters: Misfire Events

Subsystem (mask)

Injection of misfire events.
Misfire events are defined for each cylinder
with a monotonically increasing row vector [1xn].
Every misfire will occur shortly after time n, at (around) TDC firing.
To define no misfire, enter a scalar > simulation Stop Time.

Inputs

Cyl 1 misfire events, Misf1 [s]: [0.1,0.3]

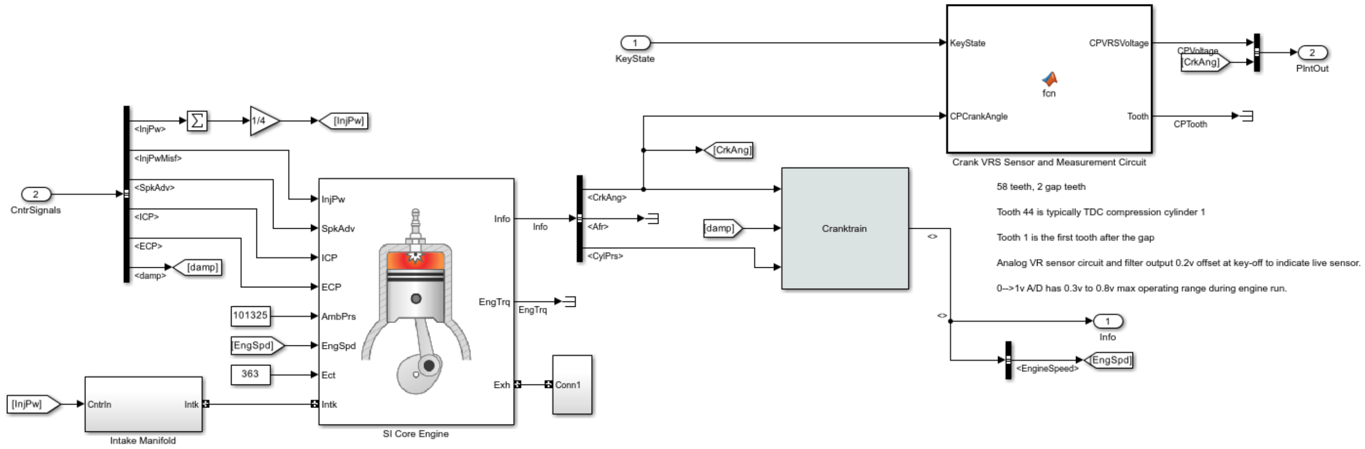
Cyl 2 misfire events, Misf2 [s]: 100

Cyl 3 misfire events, Misf3 [s]: 0.675 0.7 0.75 0.8 0.85

Cyl 4 misfire events, Misf4 [s]: [0.2,0.4]

Engine Plant With Sensors and Actuators

The SI Core Engine computes cylinder pressure as a function of crank angle. The Crank VRS Sensor and Measurement Circuit simulates the 58-tooth wheel that tracks the progression of the 4-stroke engine cycle.



58 teeth, 2 gap teeth

Tooth 44 is typically TDC compression cylinder 1

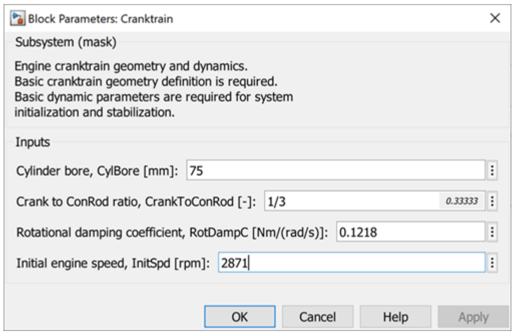
Tooth 1 is the first tooth after the gap

Analog VR sensor circuit and filter output 0.2v offset at key-off to indicate live sensor.

0-->1v A/D has 0.3v to 0.8v max operating range during engine run.

Cranktrain Block

Open the Cranktrain block to modify the engine cranktrain geometry and dynamics.



Block Parameters: Cranktrain

Subsystem (mask)

Engine cranktrain geometry and dynamics.
Basic cranktrain geometry definition is required.
Basic dynamic parameters are required for system initialization and stabilization.

Inputs

Cylinder bore, CylBore [mm]: 75

Crank to ConRod ratio, CrankToConRod [-]: 1/3 0.33333

Rotational damping coefficient, RotDampC [Nm/(rad/s)]: 0.1218

Initial engine speed, InitSpd [rpm]: 2871

See Also

SI Core Engine | DiagnosticInfoCaller | Diagnostic Service Component

More About

- “AUTOSAR Blockset”
- “Configure Calls to AUTOSAR Diagnostic Event Manager Service” (AUTOSAR Blockset)
- “SI Core Engine Air Mass Flow and Torque Production” on page 2-2

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 hybrid electric vehicle P2 reference application.

Open the Hybrid Electric Vehicle P2 Reference Application

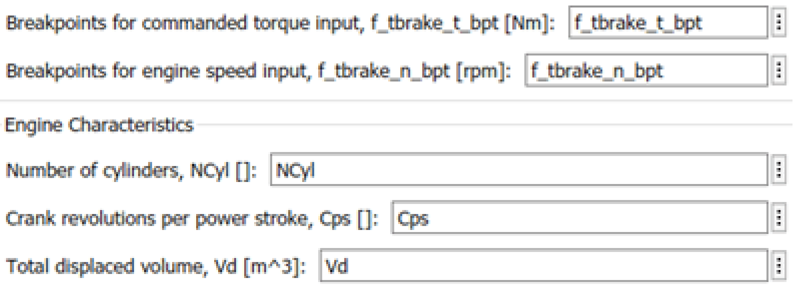
```
workDir = pwd;
autoblkHevP2Start;
cd(workDir);
```

Set a variable equal to the block path.

```
bp = 'SiMappedEngine/Mapped SI Engine'; % block path
```

Open Mapped SI Engine Block

In the `HevP2ReferenceApplication` model, navigate to `Passenger Vehicle` > `Ideal Mapped Engine` > `SiMappedEngine` . 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

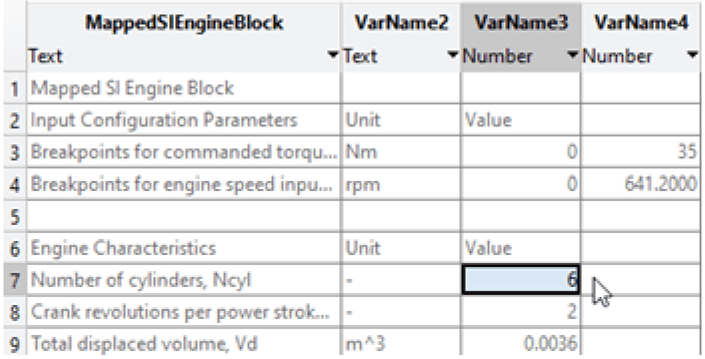
The functions in the example overwrite the workspace variables with the values in the data file.

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 Text	VarName2 Text	VarName3 Number	VarName4 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, 'slBlock
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

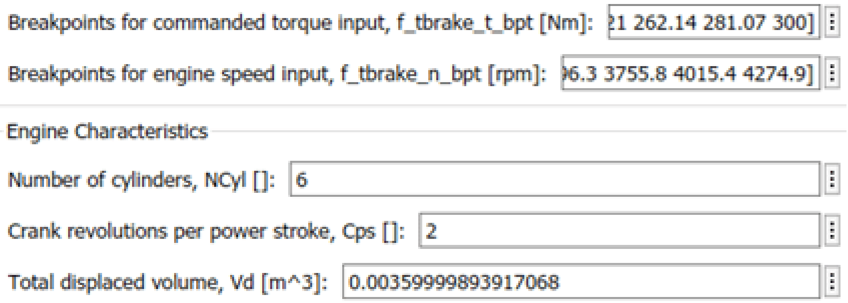
Use this code to read the data file and update the Mapped SI Engine block parameters.

```
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 `HevP2ReferenceApplication` model, navigate to `Passenger Vehicle` > `Ideal Mapped Engine` > `SiMappedEngine`. 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]: 96.3 3755.8 4015.4 4274.9]

Engine Characteristics

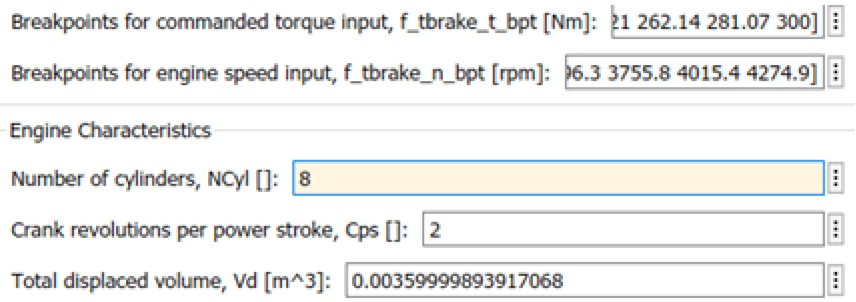
Number of cylinders, NCyl []: 6

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]: 96.3 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('SiMappedEngine');
```

Write Parameter Data to File

First, create a copy of the data file. Write the modified parameter data to the copy of the data file.

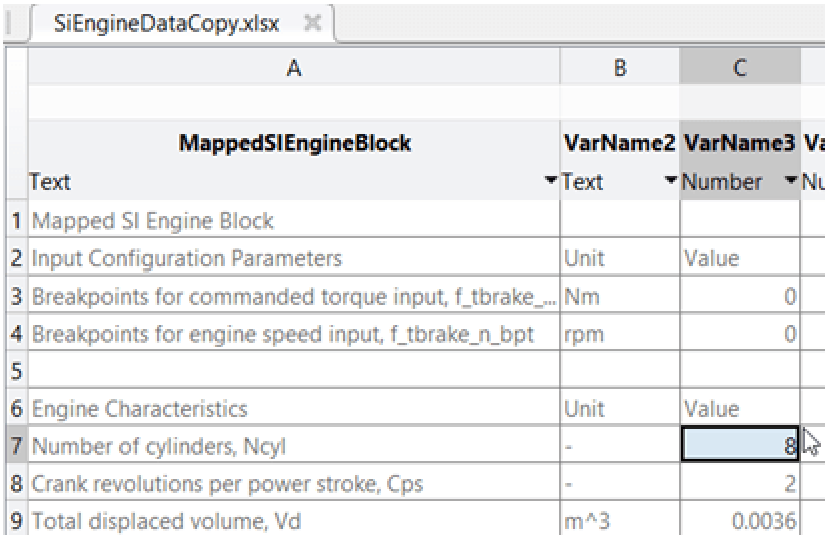
```
copyfile('SiEngineData.xlsx','SiEngineDataCopy.xlsx','f');
fileName = 'SiEngineDataCopy.xlsx';
```

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.



SiEngineDataCopy.xlsx

	A	B	C	
	MappedSIEngineBlock Text	VarName2 Text	VarName3 Number	Va Nu
1	Mapped SI Engine Block			
2	Input Configuration Parameters	Unit	Value	
3	Breakpoints for commanded torque input, f_tbrake_...	Nm	0	
4	Breakpoints for engine speed input, f_tbrake_n_bpt	rpm	0	
5				
6	Engine Characteristics	Unit	Value	
7	Number of cylinders, Ncyl	-	8	
8	Crank revolutions per power stroke, Cps	-	2	
9	Total displaced volume, Vd	m^3	0.0036	

See Also

Mapped SI Engine

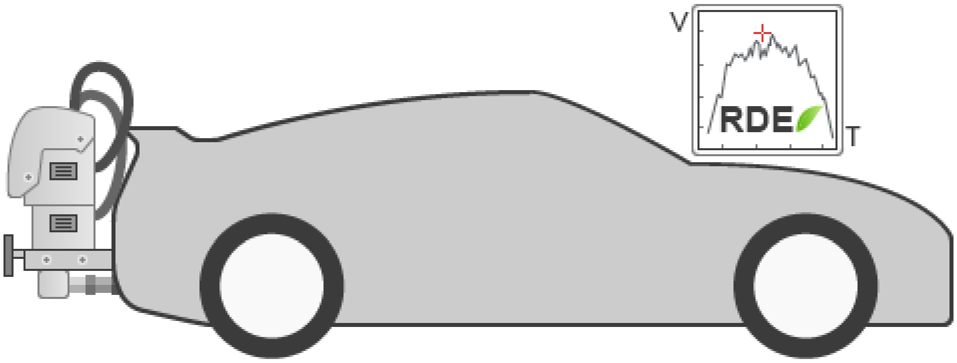
Related Examples

- “HEV P2 Reference Application” on page 7-7

Generate Drive Cycles for Real Driving Emissions

Real Driving Emissions (RDE) is an emissions standard required by the European Union. To meet this standard, a car is driven on public roads and over a wide range of different conditions. Specific equipment installed on the vehicle, the Portable Emission Measuring System (PEMS), collects data to verify that legislative caps for pollutants such as NOx are not exceeded.

This example shows how to generate an RDE-compliant trip.



RDE trip parameters

Instantiate an RDE object

```
DriveCycles = RDE.DriveCycles; % create RDE object
```

Sampling interval [s]

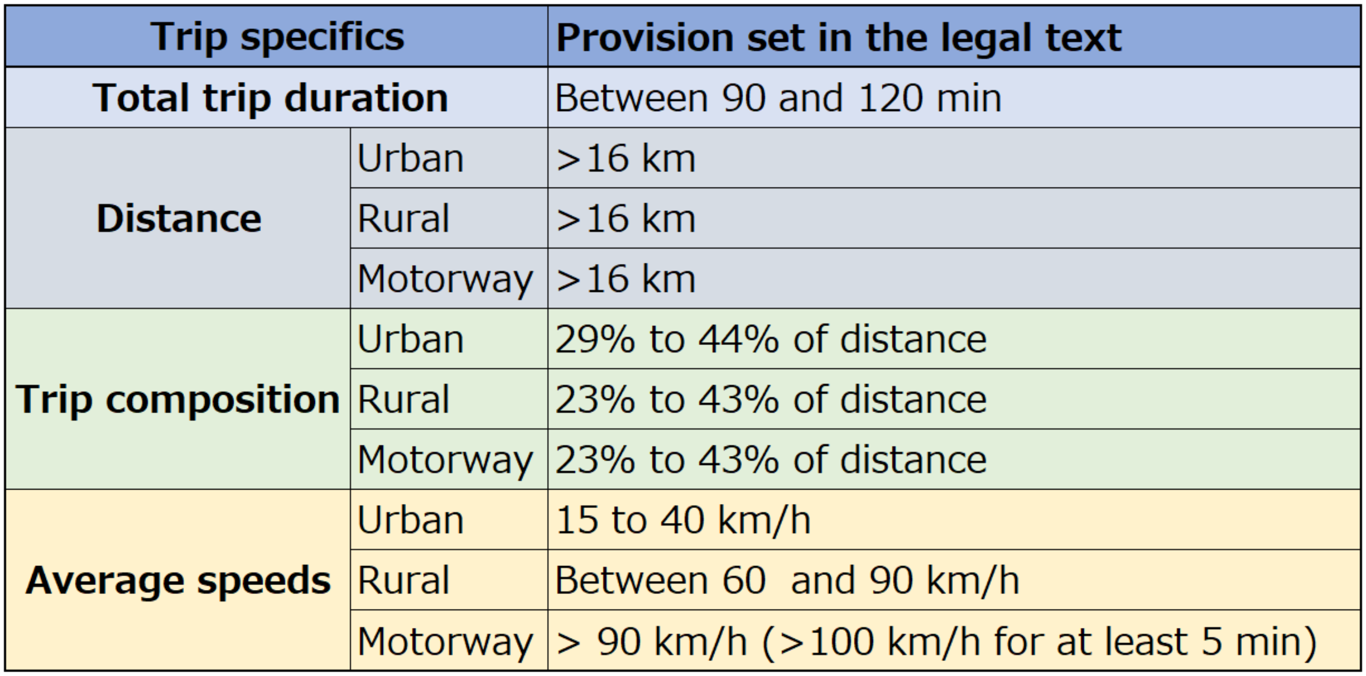
```
DriveCycles.dt = 1; % sampling interval [s]
```

Velocity threshold for stop condition [m/s]

```
DriveCycles.StopSpeedTh = 1/3.6; % stop velocity threshold [m/s]
```

RDE Trip Specifications

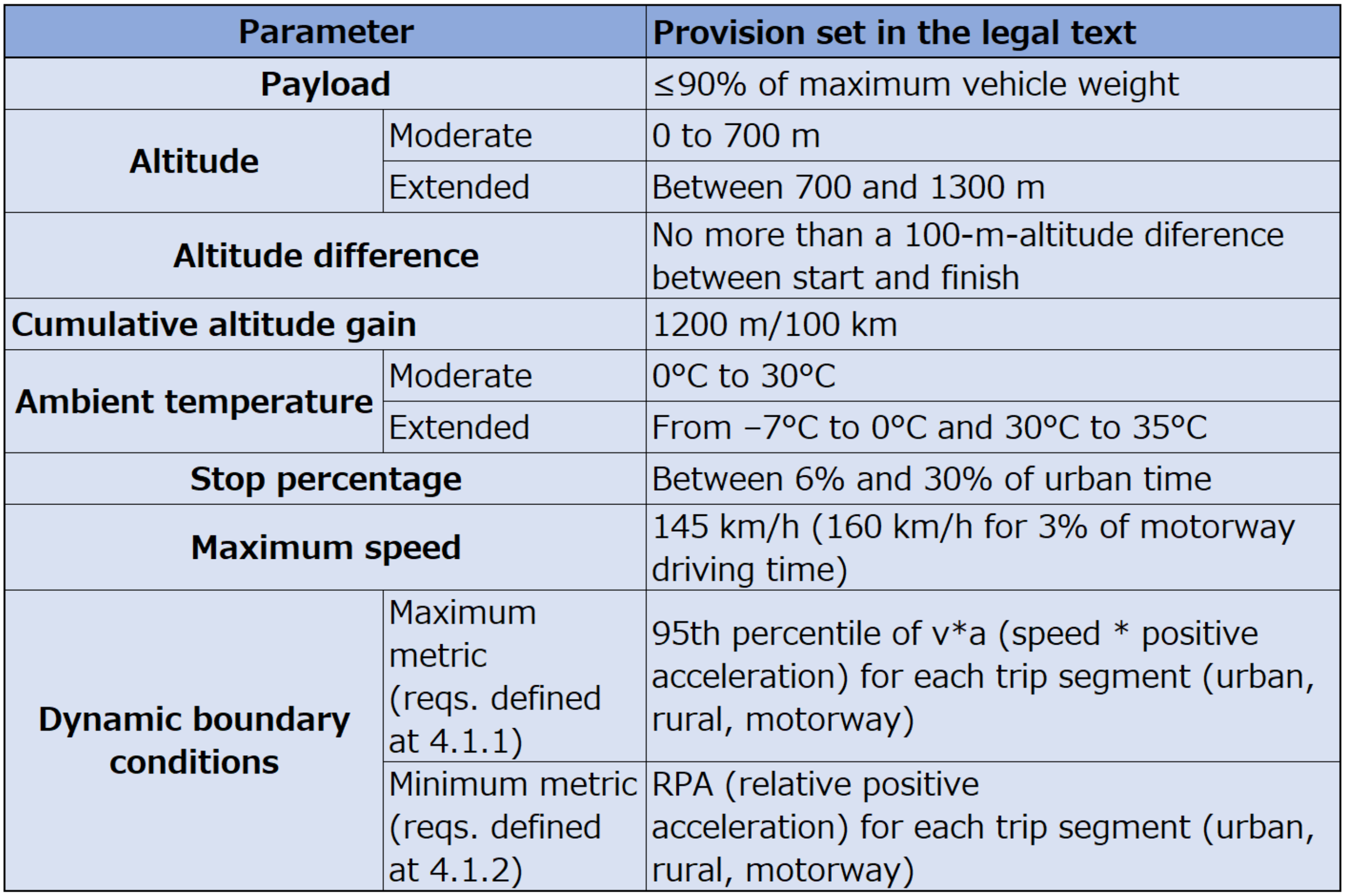
RDE trips cover three types of operation: urban, rural, and motorway. All datasets with v ≤ 60 km/h belong to the ‘urban’ speed bin, all datasets with 60 km/h < v ≤ 90 km/h belong to the ‘rural’ speed bin and all datasets with v > 90 km/h belong to the ‘motorway’ speed bin. These classifications are based purely on speed. The urban/rural/motorway mix, based on the speed definition, should be evenly distributed for each category within a 10% tolerance. The table shows the distance and speed specifications for each urban, rural, and motorway part of the RDE test.



Trip specifics		Provision set in the legal text
Total trip duration		Between 90 and 120 min
Distance	Urban	>16 km
	Rural	>16 km
	Motorway	>16 km
Trip composition	Urban	29% to 44% of distance
	Rural	23% to 43% of distance
	Motorway	23% to 43% of distance
Average speeds	Urban	15 to 40 km/h
	Rural	Between 60 and 90 km/h
	Motorway	> 90 km/h (>100 km/h for at least 5 min)

Boundary Conditions for the RDE Tests

In addition to specifying the trip characterization, other defined boundary conditions include ambient conditions, stop times, maximum speed, and altitude. A set of additional dynamic boundary conditions has been added for the second RDE legislative package to exclude driving that could be regarded as too smooth or too aggressive, based on indicators such as speed and acceleration. The table shows the dynamic boundary conditions for the RDE tests.



Parameter		Provision set in the legal text
Payload		≤90% of maximum vehicle weight
Altitude	Moderate	0 to 700 m
	Extended	Between 700 and 1300 m
Altitude difference		No more than a 100-m-altitude diference between start and finish
Cumulative altitude gain		1200 m/100 km
Ambient temperature	Moderate	0°C to 30°C
	Extended	From −7°C to 0°C and 30°C to 35°C
Stop percentage		Between 6% and 30% of urban time
Maximum speed		145 km/h (160 km/h for 3% of motorway driving time)
Dynamic boundary conditions	Maximum metric (reqs. defined at 4.1.1)	95th percentile of v*a (speed * positive acceleration) for each trip segment (urban, rural, motorway)
	Minimum metric (reqs. defined at 4.1.2)	RPA (relative positive acceleration) for each trip segment (urban, rural, motorway)

NOTE: The present RDE package does not generate altitude or temperature data.

RDE Parameters

Urban/rural/motorway velocity range [m/s]

Parameter defined in the RDE legislation at chapter: 6.3.1, 6.4.1, 6.4.2, 6.5.1

```
DriveCycles.OperationModeBoundaries = [60 90]/3.6; % Boundaries between urban, rural, motorway [
```

Allowed distance normalized percentage for the urban part of the trip

Parameter defined in the RDE legislation at chapter: 6.6.1

```
DriveCycles.UrbanRatioRange = [0.29 0.44]; % Allowed distance normalized percentage for the urba
```

Allowed distance normalized percentage for the rural part of the trip

Parameter defined in the RDE legislation at chapter: 6.6.2

```
DriveCycles.RuralRatioRange = [0.23 0.43]; % Allowed distance normalized percentage for the rura
```

Allowed distance normalized percentage for the motorway part of the trip

Parameter defined in the RDE legislation at chapter: 6.6.3

```
DriveCycles.MotorwayRatioRange = [0.23 0.43]; % Allowed distance normalized percentage for the mo
```

Parameters defined in the RDE legislation at chapter: 6.7.1

Usual max velocity [m/s] (can be occasionally higher, for overpassing, etc.)

```
DriveCycles.MotorwayUsualMaxSpeed = 145/3.6; % Usual max velocity [m/s] (can be occasionally higl
```

Time ratio limit for higher velocities

```
DriveCycles.MotorwayAbsoluteSpeedTimeRatio = 0.03; % Time ratio limit for higher velocities []
```

Absolute max velocity [m/s] (cannot be higher than this)

```
DriveCycles.MotorwayAbsoluteMaxSpeed = 160/3.6; % Absolute max velocity [m/s] (cannot be higher t
```

Urban allowed speed range.

Parameter defined in the RDE legislation at chapter: 6.8.1

```
DriveCycles.UrbanAverageSpeedRange = [15 40]/3.6; % urban allowed speed range [m/s]
```

Urban stop normalized percentage range.

Parameter defined in the RDE legislation at chapter: 6.8.2

```
DriveCycles.UrbanStopRatioRange = [0.06 0.3]; % Urban stop normalized percentage range []
```

Urban min stop time and number of stop events.

Parameter defined in the RDE legislation at chapter: 6.8.3

```
DriveCycles.UrbanMinStopTime = 10; % Urban min stop time [s]
DriveCycles.UrbanMinStopCount = 2; % Urban min stop occurrences []
```

Motorway min velocity and time allowed for min velocity.

Parameter defined in the RDE legislation at chapter: 6.9.1

```
DriveCycles.MotorwayUsualMinSpeed = 100/3.6; % Motorway min velocity [m/s]
DriveCycles.MotorwayUsualMinSpeedTime = 5*60; % Time allowed for min velocity [s]
```

Range for RDE trip duration

Parameter defined in the RDE legislation at chapter: 6.10.1

```
DriveCycles.TripDurationRange = [90 120]*60; % Allowed total trip duration range [s]
```

Min distance for urban, rural, and motorway parts

Parameters defined in the RDE legislation at chapter: 6.12

```
DriveCycles.UrbanMinDistance = 16000; % Min distance for urban part [m]
DriveCycles.RuralMinDistance = 16000; % Min distance for rural part [m]
DriveCycles.MotorwayMinDistance = 16000; % Min distance for motorway part [m]
```

Rural and motorway average speed range

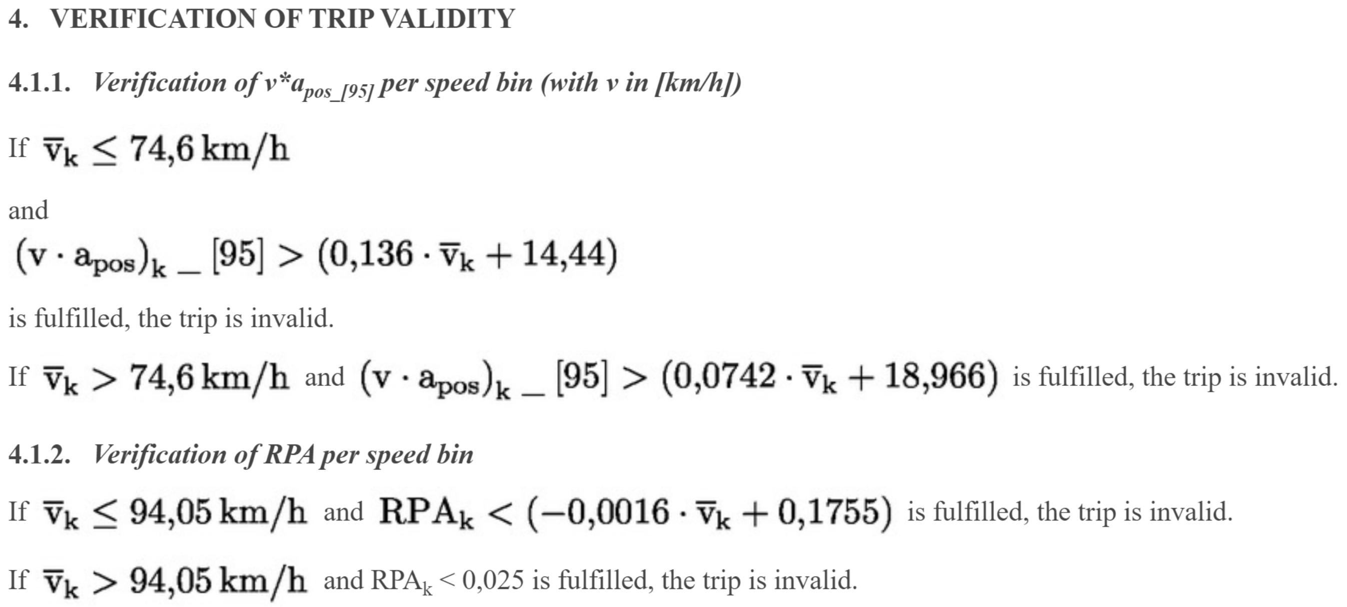
```
DriveCycles.RuralAverageSpeedRange = [60 90]/3.6; % Rural average speed range [m/s]
DriveCycles.MotorwayAverageSpeedRange = [90 145]/3.6; % Motorway average speed range [m/s]
```

Dynamic Boundary Conditions Used by the Relative Positive Acceleration (RPA) and VA95

These conditions are defined in order to exclude driving that could be regarded as too smooth or too aggressive, based on indicators such as speed and acceleration.

To be valid, each urban, rural, and motorway section of an RDE trip must be below the VA95 constraint line and above the RPA constraint line.

The constraints are defined as below



4. VERIFICATION OF TRIP VALIDITY

4.1.1. ***Verification of v*$a_{pos\_[95]}$ per speed bin (with v in [km/h])***

If $\overline{\mathrm{v}}_{\mathrm{k}} \leq 74{,}6\,\mathrm{km/h}$

and

$(\mathrm{v} \cdot \mathrm{a}_{\mathrm{pos}})_{\mathrm{k}}\_[95] > (0{,}136 \cdot \overline{\mathrm{v}}_{\mathrm{k}} + 14{,}44)$

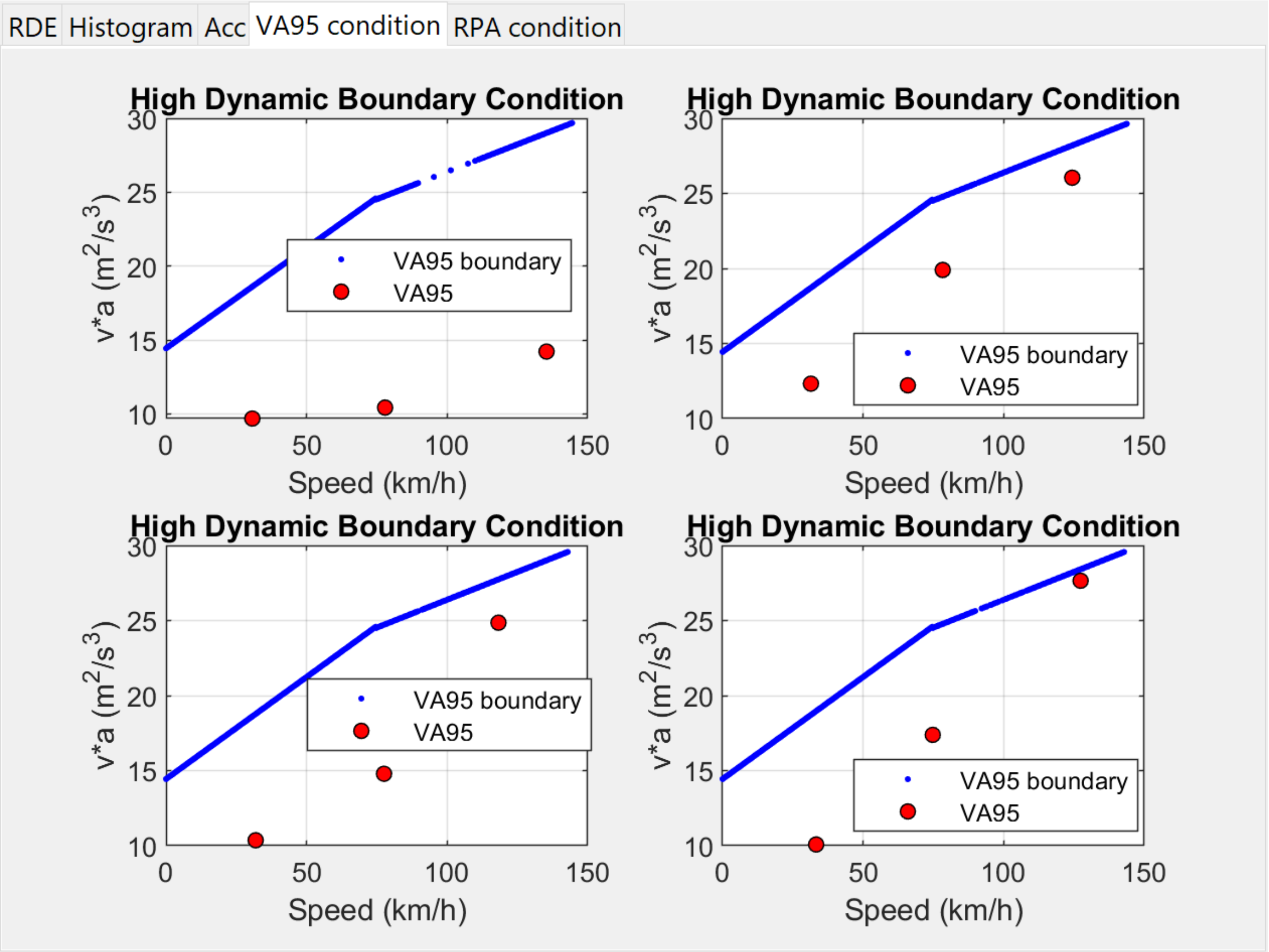
is fulfilled, the trip is invalid.

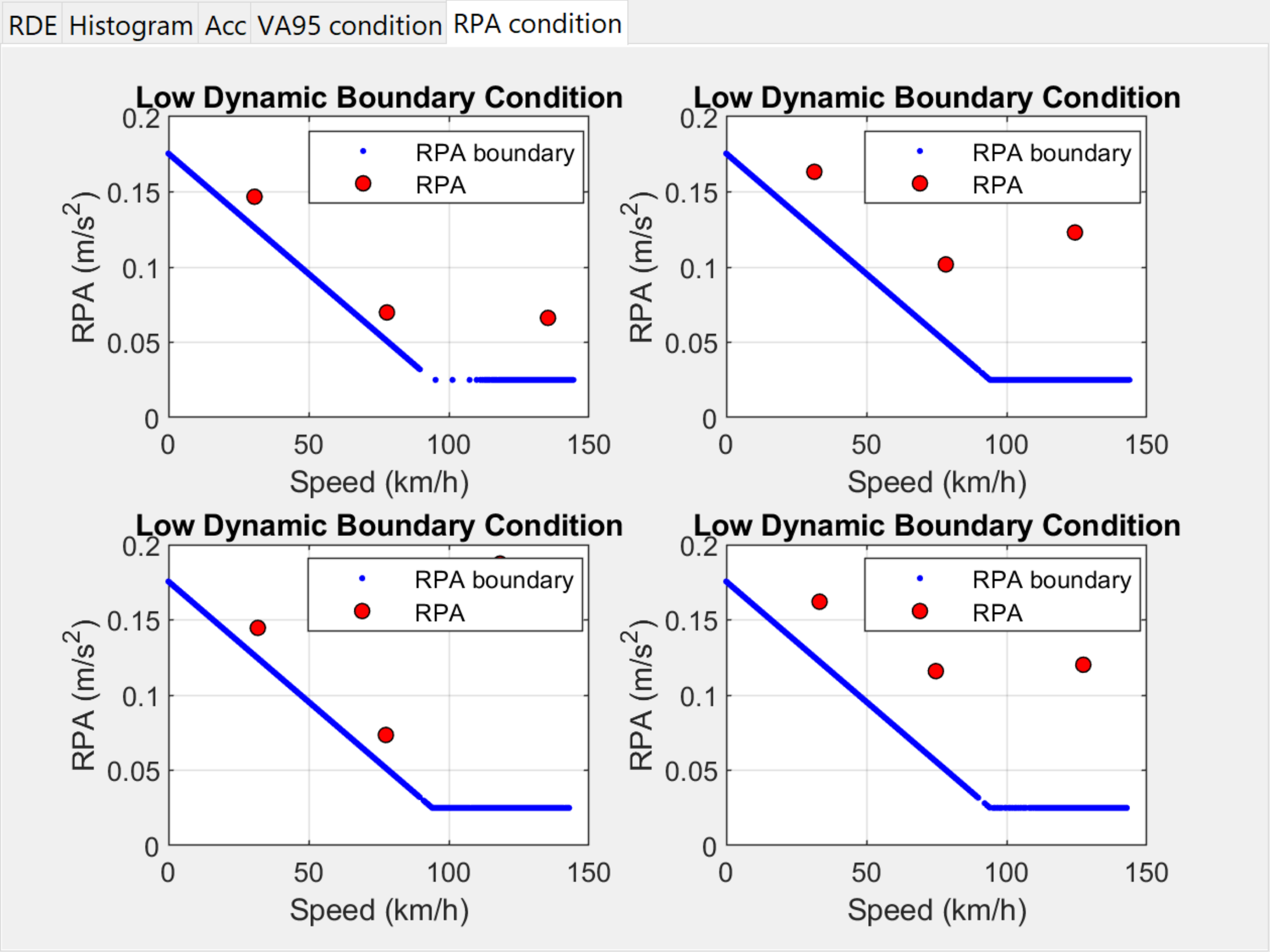
If $\overline{\mathrm{v}}_{\mathrm{k}} > 74{,}6\,\mathrm{km/h}$ and $(\mathrm{v} \cdot \mathrm{a}_{\mathrm{pos}})_{\mathrm{k}}\_[95] > (0{,}0742 \cdot \overline{\mathrm{v}}_{\mathrm{k}} + 18{,}966)$ is fulfilled, the trip is invalid.

4.1.2. ***Verification of RPA per speed bin***

If $\overline{\mathrm{v}}_{\mathrm{k}} \leq 94{,}05\,\mathrm{km/h}$ and $\mathrm{RPA}_{\mathrm{k}} < (-0{,}0016 \cdot \overline{\mathrm{v}}_{\mathrm{k}} + 0{,}1755)$ is fulfilled, the trip is invalid.

If $\overline{\mathrm{v}}_{\mathrm{k}} > 94{,}05\,\mathrm{km/h}$ and $\mathrm{RPA}_{k} < 0{,}025$ is fulfilled, the trip is invalid.


RDE
Histogram
Acc
VA95 condition
RPA condition
High Dynamic Boundary Condition
v*a (m^2/s^3)
Speed (km/h)
0
50
100
150
10
15
20
25
30
VA95 boundary
VA95



RPA has units m/s^2 or kWs/(kg*km) and positive acceleration means values greater than 0.1 m/s^2.

VA95 is the 95th percentile of the product of vehicle speed per positive acceleration greater than 0.1 m/s^2 and has units of m^2/s^3 or W/kg, similar to a power to mass ratio.

Parameters for VA95 indicator verification (used in calcVa95Boundary)

Parameters defined in the RDE legislation at section 4.1.1 (VERIFICATION OF TRIP VALIDITY)

```
DriveCycles.VA95VelocityThreshold = 74.6/3.6; % m/s
DriveCycles.VA95BoundarySpeedCoeff1 = 0.136; % units equivalent to m/s^2
DriveCycles.VA95BoundaryBias1  = 14.44; %W/kg
DriveCycles.VA95BoundarySpeedCoeff2 = 0.0742; % units equivalent to m/s^2
DriveCycles.VA95BoundaryBias2  = 18.966; % W/kg
```

Parameters for RPA verification (used in calcRpaBoundary)

Parameters defined in the RDE legislation at section 4.1.2 (VERIFICATION OF TRIP VALIDITY)

```
DriveCycles.RPAVelocityThreshold = 94.05/3.6; % m/s
DriveCycles.RPABoundarySpeedCoeff = -0.0016; % units equivalent to 1/s
DriveCycles.RPABoundaryBias = 0.1755; % m/s^2
DriveCycles.RPALowerBound = 0.025; % m/s^2
```

RDE generator parameters

```
DriveCycles.ShapeParameter = 1; % Velocity generator uses the gamma PDF and this is the PDF's tu
DriveCycles.SmoothingMethod = loess; % velocity using MATLAB smooth function
DriveCycles.SmoothingWindowLength = 5; % window length
```

Number of RDE trips to be generated

```
DriveCycles.NTrips = 4; % number of RDE trips
```

Number of tries to generate a valid RDE trip

```
DriveCycles.NumberOfIterations = 10000; % iteration find valid trip limit
```

Output folder for RDE data

```
DriveCycles.OutputFolder = fullfile(pwd(), 'results'); % output folder location
```

Generate RDE trips

This will also save the timeseries data as separate CSV files.

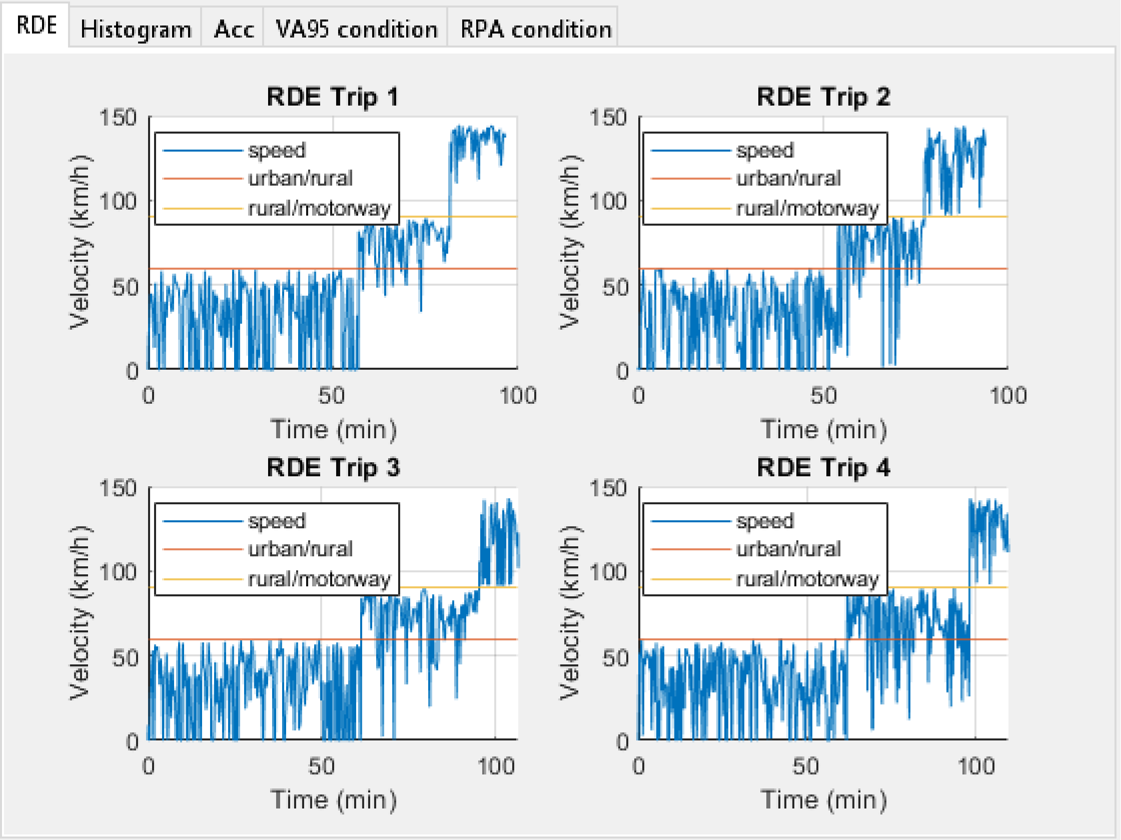
```
DriveCycles.generateDriveCycles(); % generated the results
```

```
RDE compliant drive cycle successfully generated in 81 iterations
RDE compliant drive cycle successfully generated in 92 iterations
RDE compliant drive cycle successfully generated in 33 iterations
RDE compliant drive cycle successfully generated in 120 iterations
```

To view and check the drive cycle files, open them using an application available on your platform. For example, on the Windows® platform, use the command `winopen("./results/drive_cycle_1.csv")`.

Plot the generated RDE trips

```
DriveCycles.plotDriveCycles % plot the results
```



References

The RDE legislation has been divided into four legislative packages, covering various areas such as specifications for measurement equipment, trip definitions, and boundary conditions.

RDE legislative packages:

RDE package 1

Basic features of the RDE test, such as characterization of the RDE trip, the vehicle family concept, description of the data evaluation tools, technical requirement of the PEMS equipment, and reporting obligations.

Official Journal of the European Union, L 82, 31 March 2016. 2016/427

RDE package 2

Determination of the conformity factors and the timetable for RDE implementation. Technical features include the introduction of dynamic boundary conditions and a limit for altitude gain together with a detailed approach to calculating it.

Official Journal of the European Union, L 109, 26 April 2016. 2016/646

RDE package 3

Particulate number measurement along, provisions for hybrids, and a procedure to include cold starts and regeneration events in the RDE test.

Official Journal of the European Union, L 175, 7 July 2017. 2017/1151

RDE package 4

In service compliance and surveillance tests along with specific provisions for light commercial vehicles (vans)

Official Journal of the European Union, L 301, 27 November 2018. 2018/1832

Other references

1 Vehicle Regulations Informal Working Groups UNECE Transport Division RDE background material

2 REAL DRIVING EMISSIONS (RDE) Challenges for On-Road Tests

3 Building a cycle for Real Driving Emissions

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 configure, build, operate, and analyze your virtual vehicle, use the **Composer** tab. To get started with an example, click **New** .

For this example, follow the workflow steps to build a four-wheeled electric vehicle (EV), operate it in a FTP–75 drive cycle, and analyze the results.

Step		**Button**		**Description**
1	"Setup Virtual Vehicle" on page 8-4		**Setup**	Specify: • Project path and Configuration name • Vehicle class • Powertrain architecture • Model template • Vehicle dynamics Click **Configure**
2	"Configure Virtual Vehicle Data" on page 8-7		**Data and Calibratio n**	Specify the chassis, tire, brake type, powertrain, driver, and environment. For each selection, enter the parameter data describing it.
3	"Configure Virtual Vehicle Scenario and Test" on page 8-10		**Scenario and Test**	Construct a test plan including one or more virtual vehicle test scenarios. Options include drive cycles for fuel economy and energy management analysis, and vehicle handling maneuvers.

Step		Button		Description
4	“Configure Virtual Vehicle Data Logging” on page 8-12		**Logging**	Select the model signal data to log while testing your virtual vehicle. Options include energy-related quantities, vehicle position, velocity, acceleration, and many others. You can also set the default signals for further tests.
5	“Build Virtual Vehicle” on page 8-14		**Virtual Vehicle**	Build your virtual vehicle. When you build, the **Virtual Vehicle Composer** creates a Simulink model that contains the vehicle and powertrain architectures and parameters you specified, and associates it with the test plan.
6	“Operate Virtual Vehicle” on page 8-15		**Run Test Plan**	Simulate your model according to your test plan and log the resulting output data. **Note** To operate the model, use the **Run Test Plan** button in the **Operate** section on the **Composer** tab.
7	“Analyze Virtual Vehicle” on page 8-16		**Simulation Data Inspector**	Use the Simulation Data Inspector to view and inspect the data signals that you logged. You can store your data for further processing.

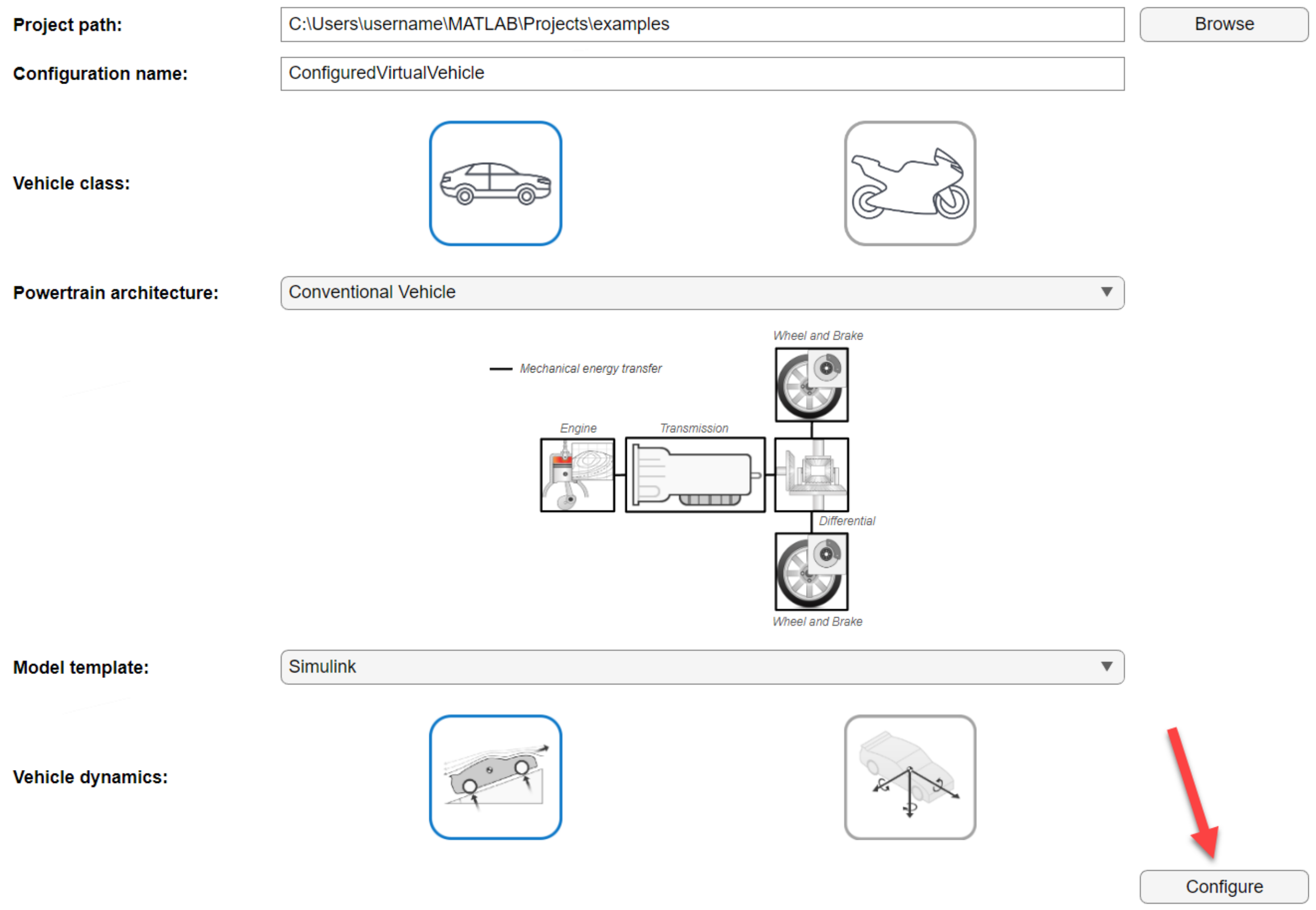
See Also

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** ✚. The app opens a default virtual vehicle template and creates virtual vehicle project files.



Project path: C:\Users\username\MATLAB\Projects\examples

Configuration name: ConfiguredVirtualVehicle

Vehicle class:

Powertrain architecture: Conventional Vehicle

Model template: Simulink

Vehicle dynamics:

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 8-7.

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

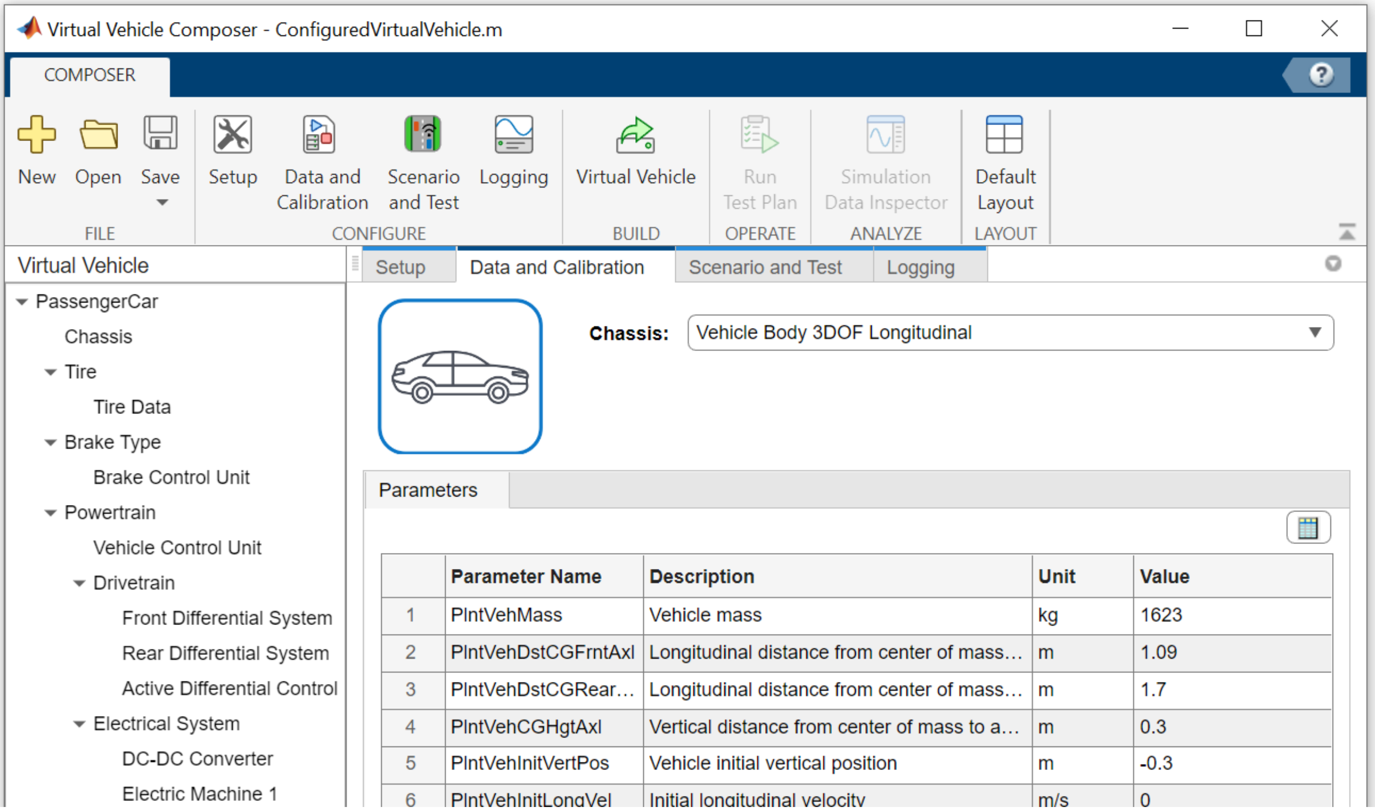
Related Examples

- “Coordinate Systems in Vehicle Dynamics Blockset” (Vehicle Dynamics Blockset)
- “Create Projects”
- “Get Started with the Virtual Vehicle Composer” on page 8-2

Configure Virtual Vehicle Data

Before completing this step, see “Setup Virtual Vehicle” on page 8-4, 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: Vehicle Body 3DOF Longitudinal

	Parameter Name	Description	Unit	Value
1	PlntVehMass	Vehicle mass	kg	1623
2	PlntVehDstCGFrntAxl	Longitudinal distance from center of mass…	m	1.09
3	PlntVehDstCGRear…	Longitudinal distance from center of mass…	m	1.7
4	PlntVehCGHgtAxl	Vertical distance from center of mass to a…	m	0.3
5	PlntVehInitVertPos	Vehicle initial vertical position	m	-0.3
6	PlntVehInitLongVel	Initial longitudinal velocity	m/s	0

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 8-10.

See Also

Virtual Vehicle Composer

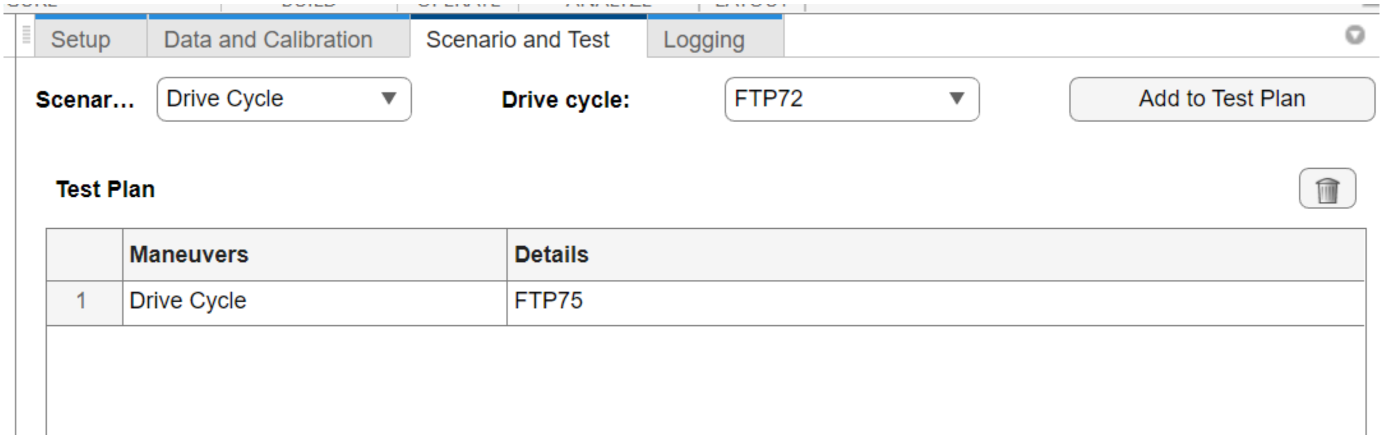
Related Examples

- “Get Started with the Virtual Vehicle Composer” on page 8-2

Configure Virtual Vehicle Scenario and Test

Before completing this step, see “Configure Virtual Vehicle Data” on page 8-7.

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.



Scenar... Drive Cycle

Drive cycle: FTP72

Test Plan

	Maneuvers	Details
1	Drive Cycle	FTP75

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 “Install Drive Cycle Data” on page 5-2. 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 8-12.

See Also

Virtual Vehicle Composer

Related Examples

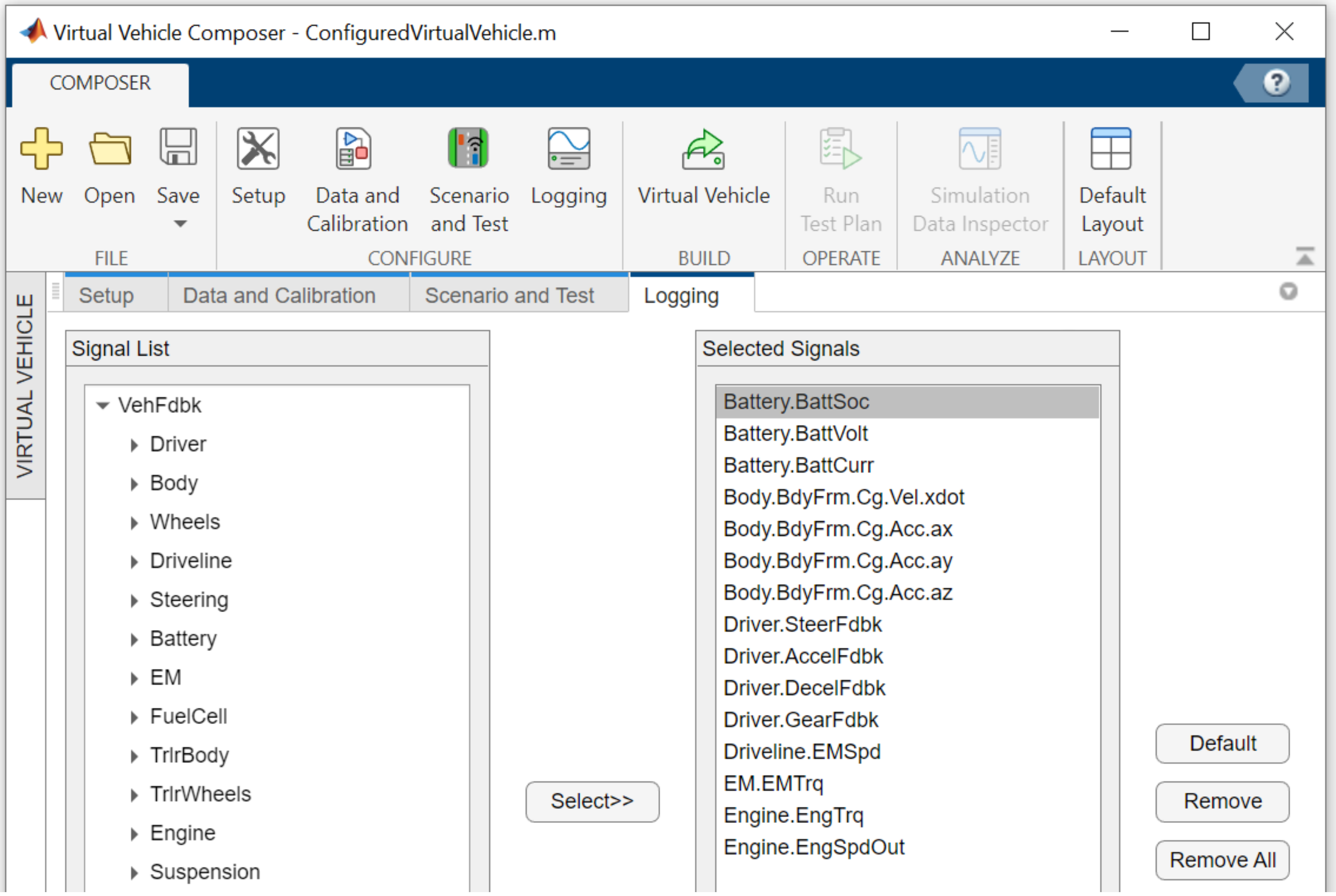
- “Get Started with the Virtual Vehicle Composer” on page 8-2

Configure Virtual Vehicle Data Logging

Before completing this step, see “Configure Virtual Vehicle Scenario and Test” on page 8-10.

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.



Signal List

- VehFdbk
  - Driver
  - Body
  - Wheels
  - Driveline
  - Steering
  - Battery
  - EM
  - FuelCell
  - TrlrBody
  - TrlrWheels
  - Engine
  - Suspension

Selected Signals

Battery.BattSoc
Battery.BattVolt
Battery.BattCurr
Body.BdyFrm.Cg.Vel.xdot
Body.BdyFrm.Cg.Acc.ax
Body.BdyFrm.Cg.Acc.ay
Body.BdyFrm.Cg.Acc.az
Driver.SteerFdbk
Driver.AccelFdbk
Driver.DecelFdbk
Driver.GearFdbk
Driveline.EMSpd
EM.EMTrq
Engine.EngTrq
Engine.EngSpdOut

After completing this step, see “Build Virtual Vehicle” on page 8-14.

See Also

Virtual Vehicle Composer

Related Examples

- “Get Started with the Virtual Vehicle Composer” on page 8-2

More About

- “Simulation Data Inspector”

Build Virtual Vehicle

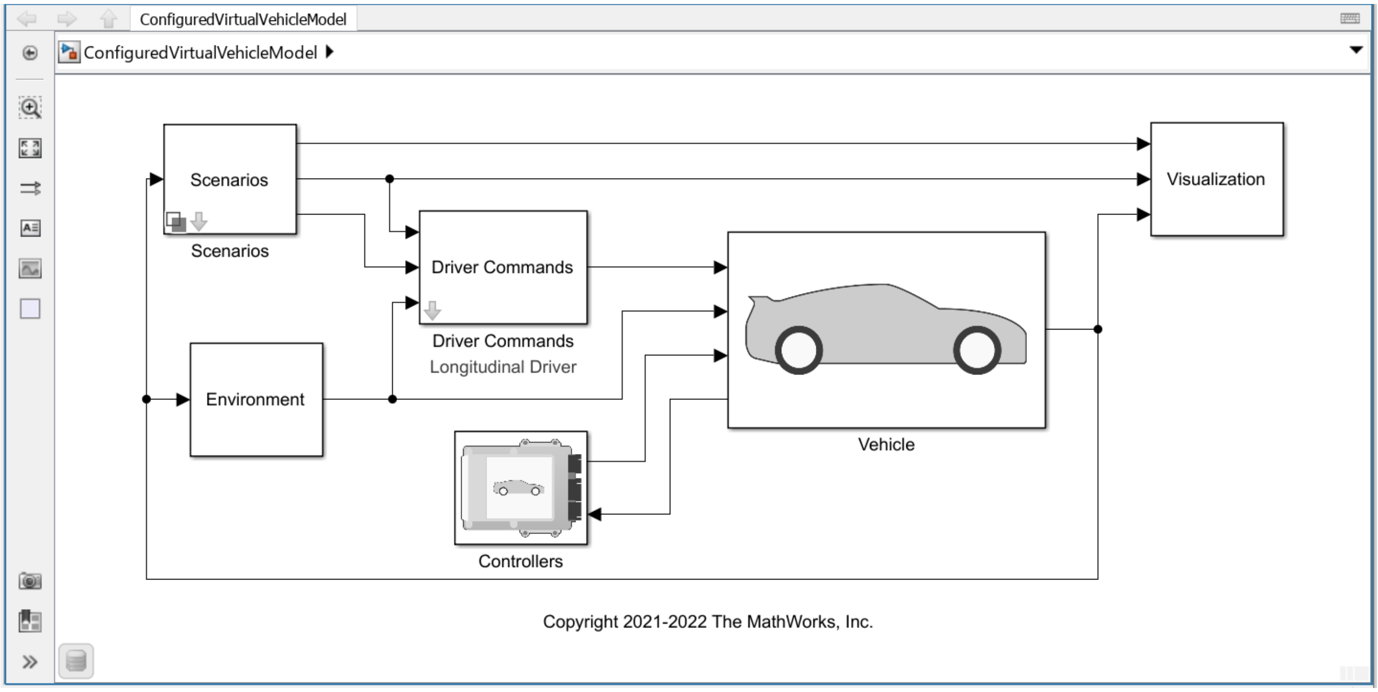
Before completing this step, see “Configure Virtual Vehicle Data Logging” on page 8-12.

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 8-15.

See Also

Virtual Vehicle Composer

Related Examples

- “Get Started with the Virtual Vehicle Composer” on page 8-2

Operate Virtual Vehicle

Before completing this step, see “Build Virtual Vehicle” on page 8-14.

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 8-16.

See Also

Virtual Vehicle Composer

Related Examples

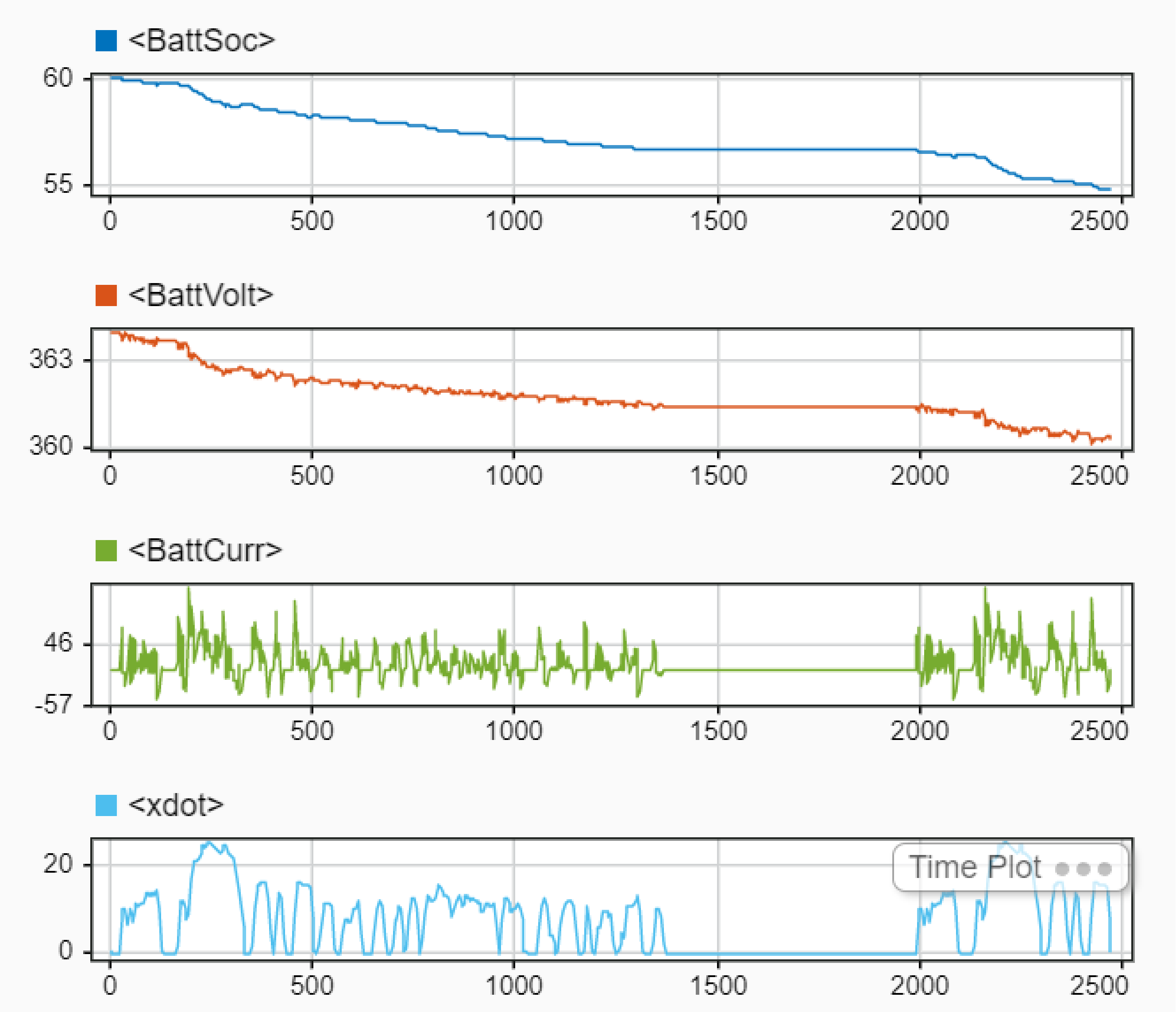
- “Get Started with the Virtual Vehicle Composer” on page 8-2

Analyze Virtual Vehicle

Before completing this step, see “Operate Virtual Vehicle” on page 8-15.

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 8-2

More About

- “Simulation Data Inspector”

Resize Engines and Mapped Motors

Use the **Virtual Vehicle Composer** app to:

- Resize an engine based on desired maximum engine power or engine displacement.
- Resize a motor based on high-level specifications.

You can use the dynamometer reference applications to resize the engines a motors. For more information, see:

- “Resize the CI Engine” on page 3-94
- “Resize the SI Engine” on page 3-101
- “Resize Motors” on page 3-92

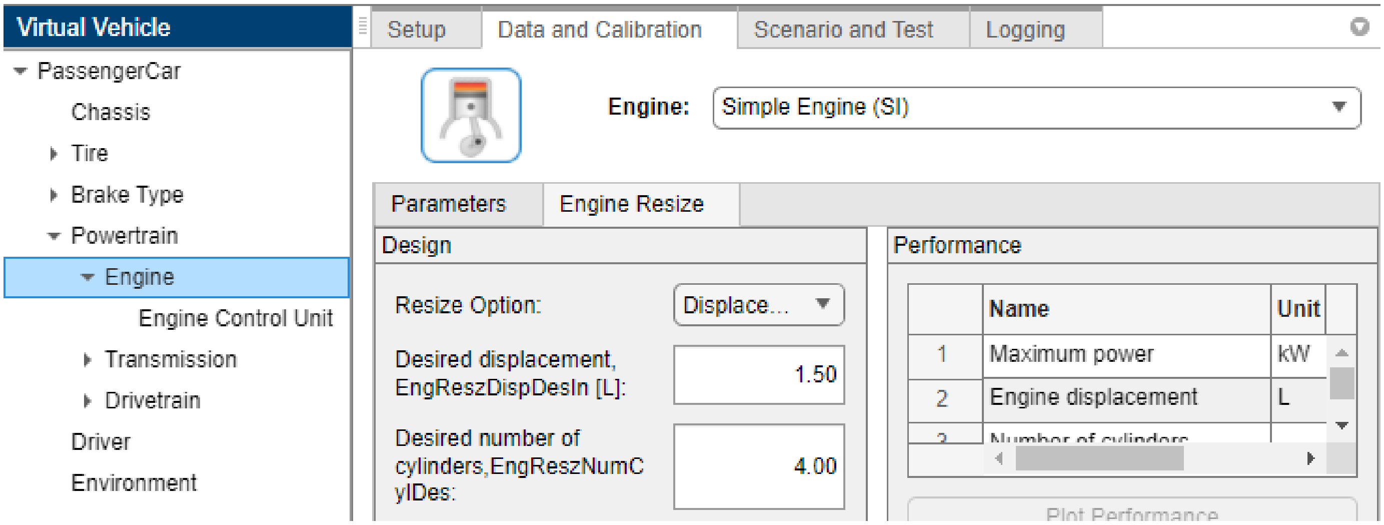
Resize Engine

For virtual vehicles configured with a conventional or hybrid electric vehicle (HEV) powertrain architecture, you can resize the engine based on a desired maximum engine power or engine displacement.

Resizing the engine requires Stateflow.

After you set up your virtual vehicle, in the **Virtual Vehicle Composer** app, follow these steps.

1 In the **Virtual Vehicle** pane, select **Engine**, then click the **Data and Calibration** tab.



Engine: Simple Engine (SI)

Resize Option: Displace...

Desired displacement, EngReszDispDesIn [L]: 1.50

Desired number of cylinders,EngReszNumCylDes: 4.00

	Name	Unit
1	Maximum power	kW
2	Engine displacement	L
3	Number of cylinders	

2 Specify a **Resize Option** option:

- `Power` — Enter a **Desired maximum power**
- `Displacement` — Enter a **Desired displacement**

For either power or displacement, set the **Desired number of cylinders**.

Depending on your **Engine**, the resize options may enable additional architecture and performance parameters.

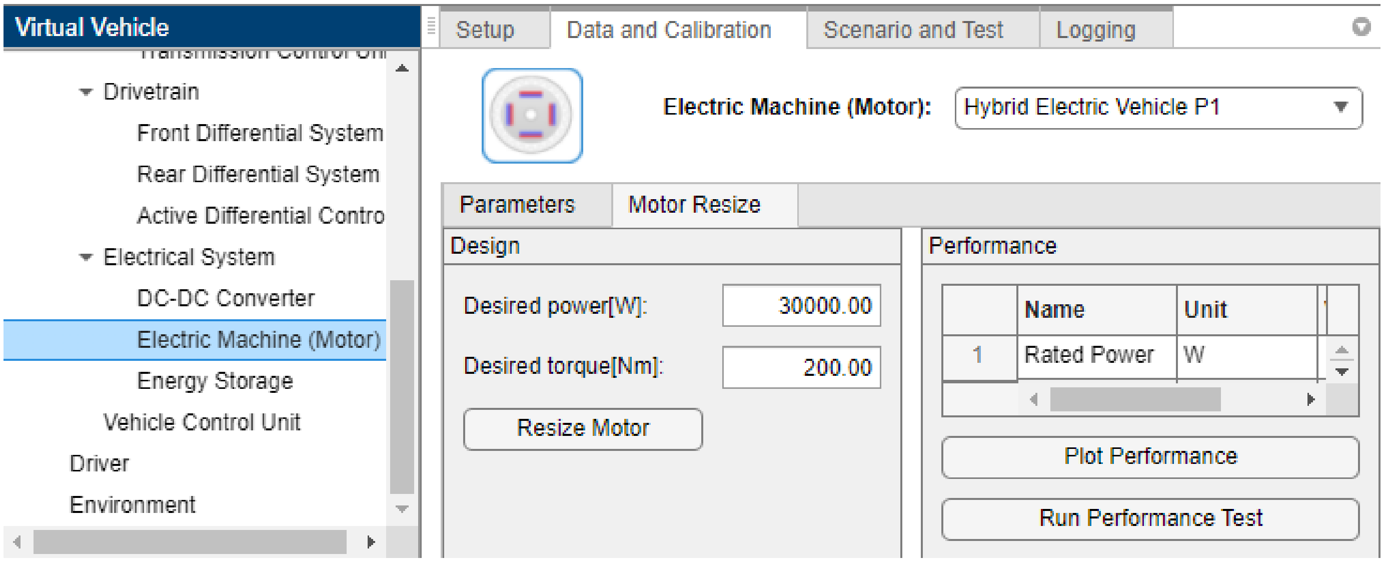
3 Select **Resize Engine**. The **Virtual Vehicle Composer** resizes the engine and adjusts the calibration parameters. The **Performance** pane provides the updated engine performance characteristics based on the resized engine.

Resize Mapped Motor

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.



Electric Machine (Motor): Hybrid Electric Vehicle P1

Desired power[W]: 30000.00

Desired torque[Nm]: 200.00

	Name	Unit
1	Rated Power	W

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 8-2
- “Calibrate, Validate, and Optimize CI Engine with Dynamometer Test Harness” on page 3-11
- “Calibrate, Validate, and Optimize SI Engine with Dynamometer Test Harness” on page 3-15

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*

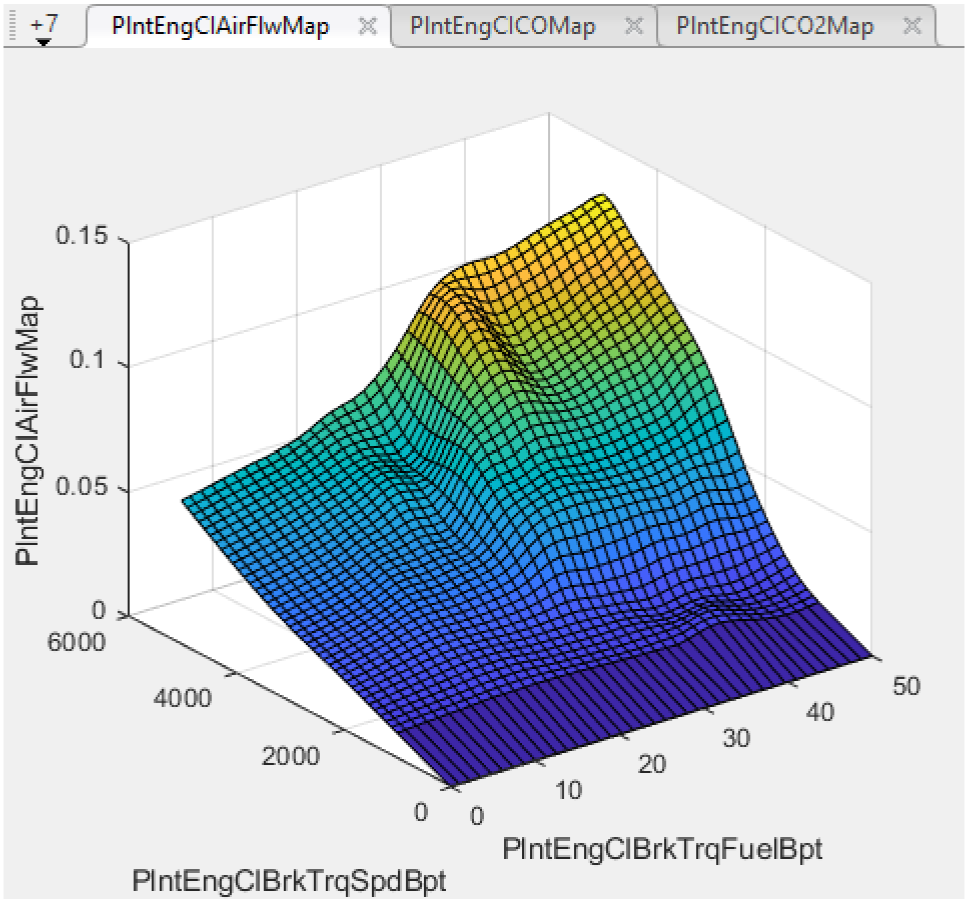
Nonfiring data contains data collected at different engine speeds without fuel consumption.

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” on page 3-108
- “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*

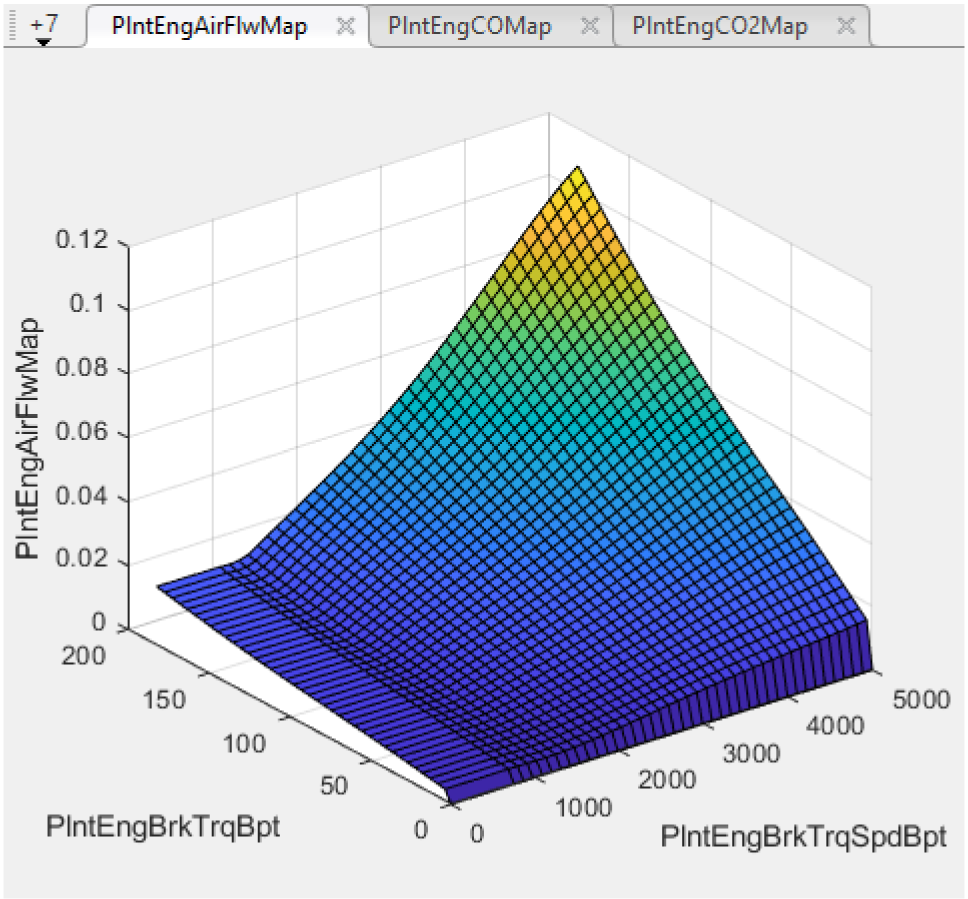
Nonfiring data contains data collected at different engine speeds without fuel consumption.

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” on page 3-113
- “Model Assessment” (Model-Based Calibration Toolbox)
- “Using Data” (Model-Based Calibration Toolbox)

Calibrate Mapped DC-to-DC Converters Using Data

If you have the Model-Based Calibration Toolbox, the **Virtual Vehicle Composer** can use a dataset to calibrate a mapped DC-to-DC converter 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**.

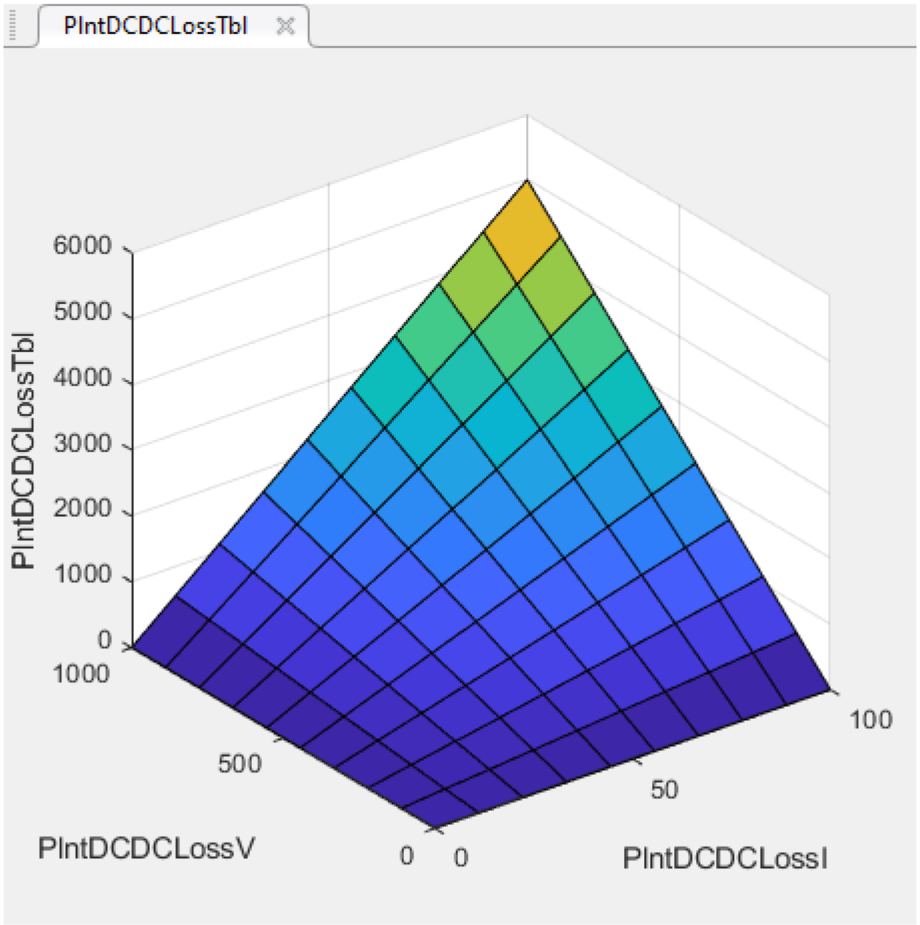
   b. Set **Powertrain architecture** to a configuration that uses an EV or HEV, for example, `Hybrid Electric Vehicle P2`.

   c. Click **Configure**.
3. On the **Data and Calibration** tab, navigate to **Powertrain** > **Electrical System** > **DC-DC Converter**.
4. Set **DC-DC Converter** to `DC-DC Converter`.
5. Select the **Calibrate from Data** tab.
6. Use **Browse** to specify the data file. By default, the app uses the file `MappedDCDCConverterDataset.xlsx`, which contains the required data.

   The table summarizes the data file requirements for generating calibrations from measured DC-to-DC power loss data at steady-state operating conditions.

Calibration Type	Required Data
`Tabulated loss data`	• Current, A • Voltage, V • Power loss, W
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.

   When the process completes, the app updates the powertrain subsystem Bidirectional DC-DC block parameters with the calibrated data.
8. Review the DC-to-DC loss characteristics response surface model.



9 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

Bidirectional DC-DC

More About

- “What Is CAGE?” (Model-Based Calibration Toolbox)
- “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`.

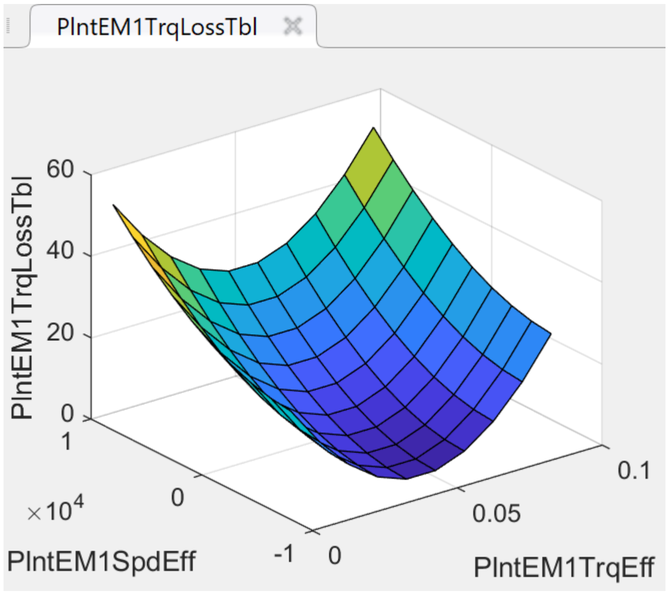
   c. Click **Configure**.
3. On the **Data and Calibration** tab, select the electric motor that you want to calibrate. Specifically, select **Powertrain** > **Electrical System** > **Electric Machine *x***, where *x* is 1, 2, 3, or 4 and corresponds to the motor.

   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)