MathWorks | Training Services

# Exercises: Vehicle Drive and Control

Student Competition - Physical Modeling Training

## Engine Drive Cycle Data

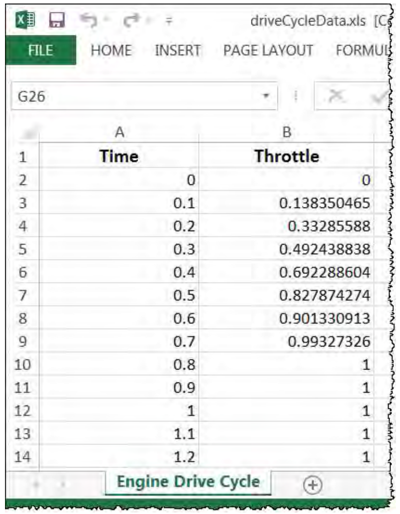
In this exercise, you will import engine drive cycle data and use it as input for a car model. To do this, open the carDriveCycleStart model and then take the following steps.

1. Import the drive cycle data from the `driveCycleData.xls` spreadsheet. The first column represents the time (from 0 to 30 seconds), and the second column contains associated throttle values (from 0 to 1).
2. Open the `carDriveCycleStart` model and
   - Replace the Constant block connected to the engine throttle with an Inport block from the **Simulink → Sources** library.
   - Configure the model to load the imported throttle data via the input port.
3. (Bonus) Add a model callback (**File → Model Properties → Model Properties**) so the throttle data is automatically loaded when the model is opened.

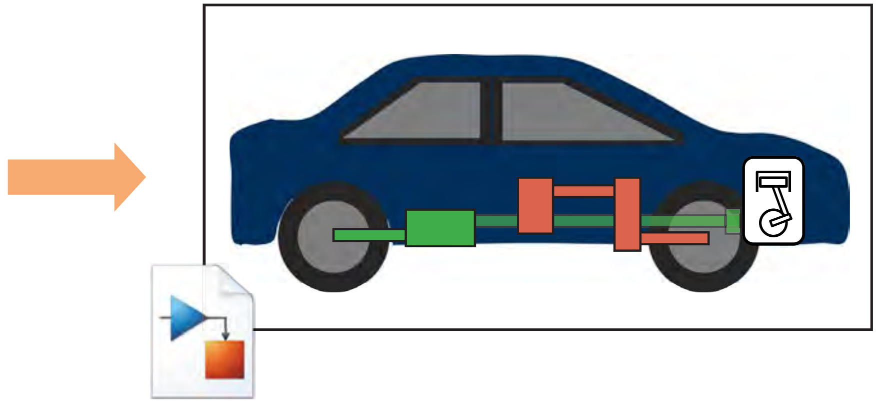
**Try**

```
>> carDriveCycleStart

>> winopen('driveCycleData.xls');
```



| | A | B |
|---|---|---|
| 1 | Time | Throttle |
| 2 | 0 | 0 |
| 3 | 0.1 | 0.138350465 |
| 4 | 0.2 | 0.33285588 |
| 5 | 0.3 | 0.492438838 |
| 6 | 0.4 | 0.692288604 |
| 7 | 0.5 | 0.827874274 |
| 8 | 0.6 | 0.901330913 |
| 9 | 0.7 | 0.99327326 |
| 10 | 0.8 | 1 |
| 11 | 0.9 | 1 |
| 12 | 1 | 1 |
| 13 | 1.1 | 1 |
| 14 | 1.2 | 1 |

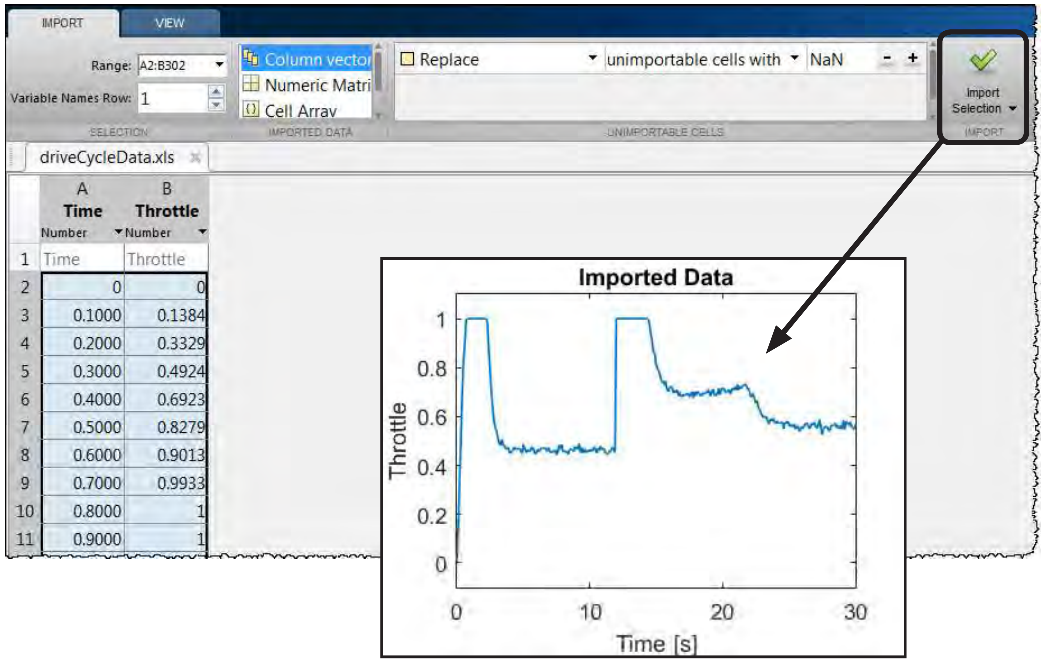


## Solution: Engine Drive Cycle Data

1. Right-click the `driveCycleData.xls` file and select **Import Data**. This will open the Import tool. Then, click **Import Selection**.

   You will see two variables in the MATLAB™ workspace named `Time` and `Throttle`, which correspond to the column names in the spreadsheet Visualize the data in a MATLAB figure window.

```
>> plot(Time,Throttle)
```



Range: A2:B302

Variable Names Row: 1

driveCycleData.xls

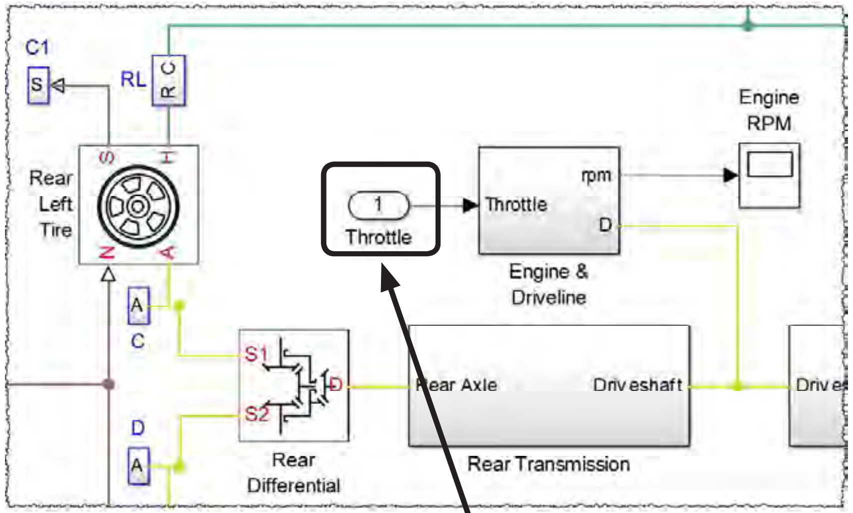
| | A<br>Time<br>Number | B<br>Throttle<br>Number |
|---|---|---|
| 1 | Time | Throttle |
| 2 | 0 | 0 |
| 3 | 0.1000 | 0.1384 |
| 4 | 0.2000 | 0.3329 |
| 5 | 0.3000 | 0.4924 |
| 6 | 0.4000 | 0.6923 |
| 7 | 0.5000 | 0.8279 |
| 8 | 0.6000 | 0.9013 |
| 9 | 0.7000 | 0.9933 |
| 10 | 0.8000 | 1 |
| 11 | 0.9000 | 1 |

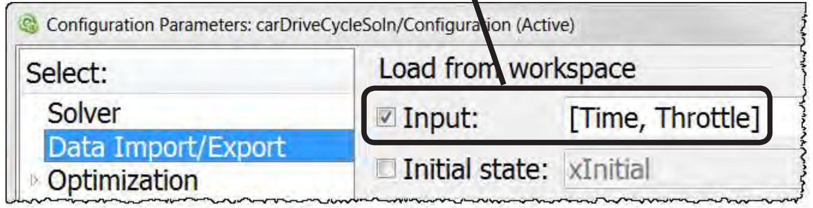
2. Replace the Constant block with an Inport block from the **Simulink → Sources** library. Then, go to **Simulation → Model Configuration Parameters** and set the following options in the **Data Import/Export** pane.

   Simulate the model and view the results in the Scope blocks provided.

**Try**

```
>> carDriveCycleSoln
```





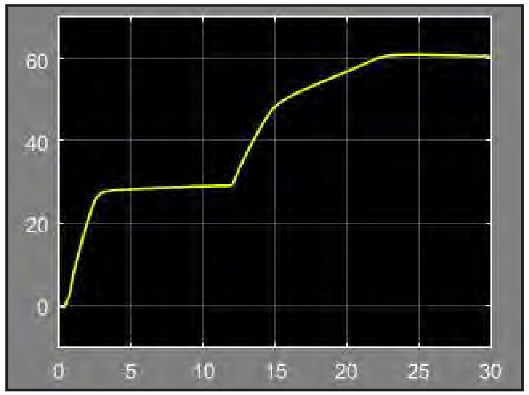
Configuration Parameters: carDriveCycleSoln/Configuration (Active)

Select: Solver, Data Import/Export, Optimization

Load from workspace

☑ Input: [Time, Throttle]

☐ Initial state: xInitial



Vehicle speed (km/hr)

## Engine Throttle Control

In this exercise, you will control the speed of a car model by using a discrete PI controller for the engine throttle.

The `carControlStart` model contains a reference speed signal, a model of a car, and a sensor that outputs a discrete, delayed, and noisy speed signal. The input of the car is the engine throttle signal, which can be any value between 0 (no throttle) and 1 (full throttle).

**Try**

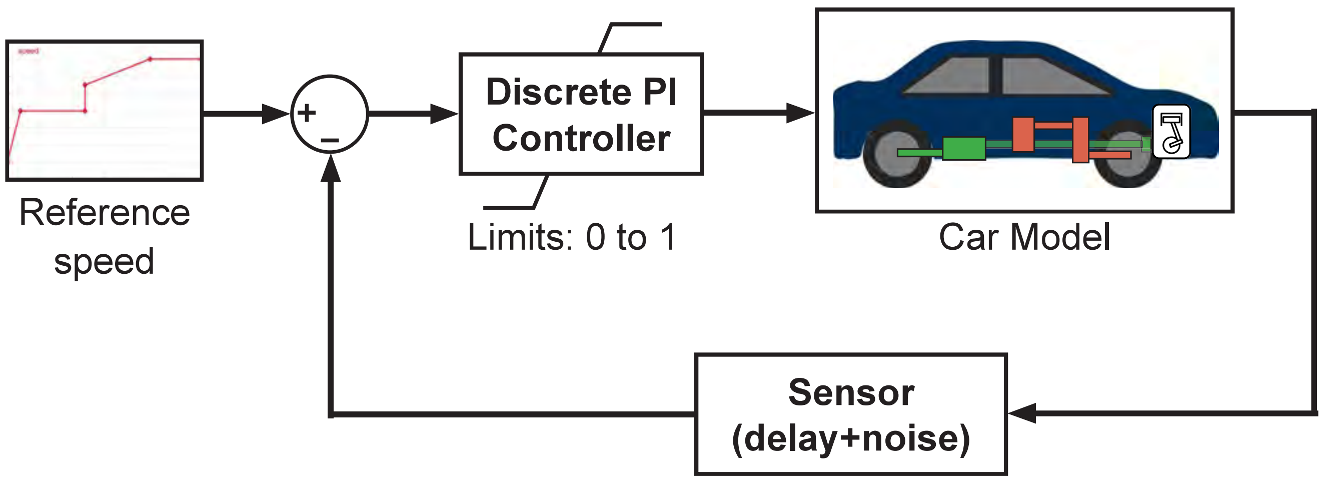
```
>> carControlStart
```

1. Complete the feedback control loop by connecting the signals as shown in the diagram below.
2. Add a PI controller with the following specifications.

| | |
|---|---|
| Proportional gain, P: | 0.1 |
| Integral gain, I: | 0.01 |
| Controller sample time: | 0.1 |

   Throttle must be between 0 and 1.

3. Simulate the model and compare the actual and reference vehicle speeds. Visualize the throttle command signal and find when the engine is under full throttle.



## Solution: Engine Throttle Control

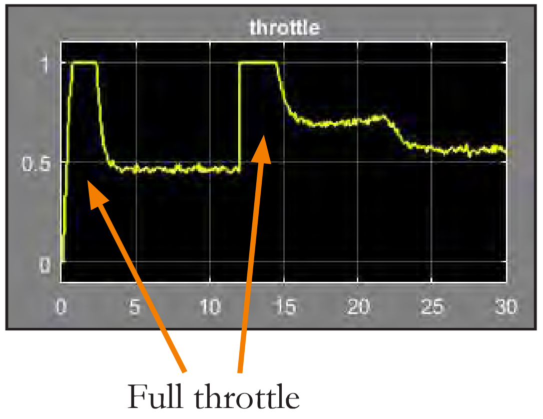
1. Add a Sum block from the **Simulink → Math Operations** library. Set its **List of signs** parameter to |+-.
2. Add a Discrete PID Controller block from the **Simulink → Discrete** library, and set the parameters as shown in the diagram.

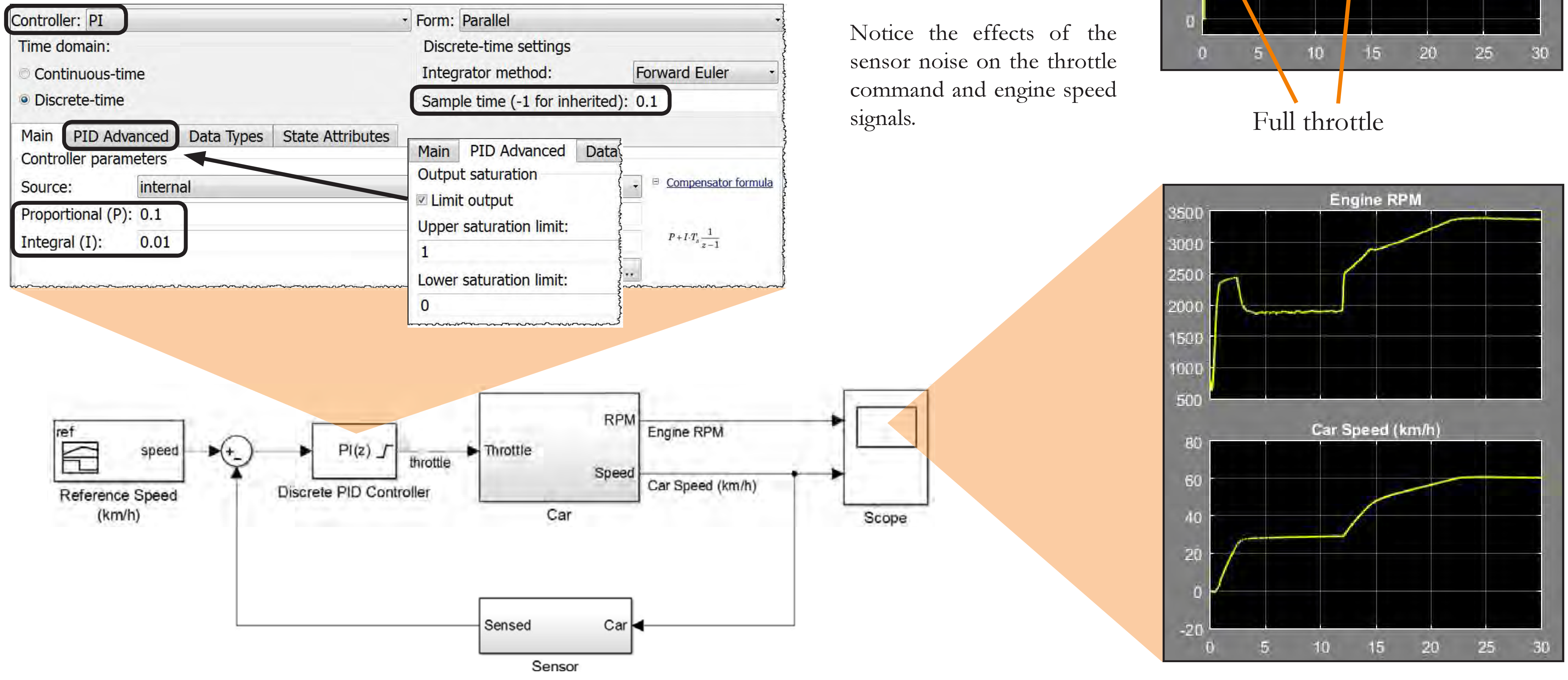
**Try**

```
>> carControlSoln
```

3. The throttle command saturates during the initial acceleration period, and again at the jump in reference speed 12 seconds after the start of simulation.

Notice the effects of the sensor noise on the throttle command and engine speed signals.





Controller: PI
Form: Parallel
Time domain:
Continuous-time
Discrete-time [✓]
Discrete-time settings
Integrator method: Forward Euler
Sample time (-1 for inherited): 0.1
Main | PID Advanced | Data Types | State Attributes
Controller parameters
Source: internal
Proportional (P): 0.1
Integral (I): 0.01

Main | PID Advanced | Data
Output saturation
Limit output [✓]
Upper saturation limit: 1
Lower saturation limit: 0

## Battery Temperature Control

In this exercise, you will model a closed-loop battery cooling system.

1. Open the `batteryCoolingStart` model. This model consists of a battery with thermal effects and a variable load. Simulate the model and view the battery temperature.
2. **Model the fan.**
   - Load the `coolerData.mat` file. This file contains two variables named `airspeed` and `coolingPower`. These variables relate the speed of the air in the fan (in m/s) to the thermal power (in watts) that the fan can remove from the system.
   - Plot the data by typing the following command.
     ```
     >> plot(airspeed,coolingPower)
     ```
   - Open the Cooling System subsystem. Replace the PS Constant block with an Inport, Simulink-PS Converter, and PS Lookup Table blocks.

     The PS Lookup Table block can be found in the **Simscape** → **Foundation Library** → **Physical Signals** → **Lookup Tables**. library.

     Use the `airspeed` and `coolingPower` variables in this table. Set the **Interpolation method** to `Cubic`.

Try

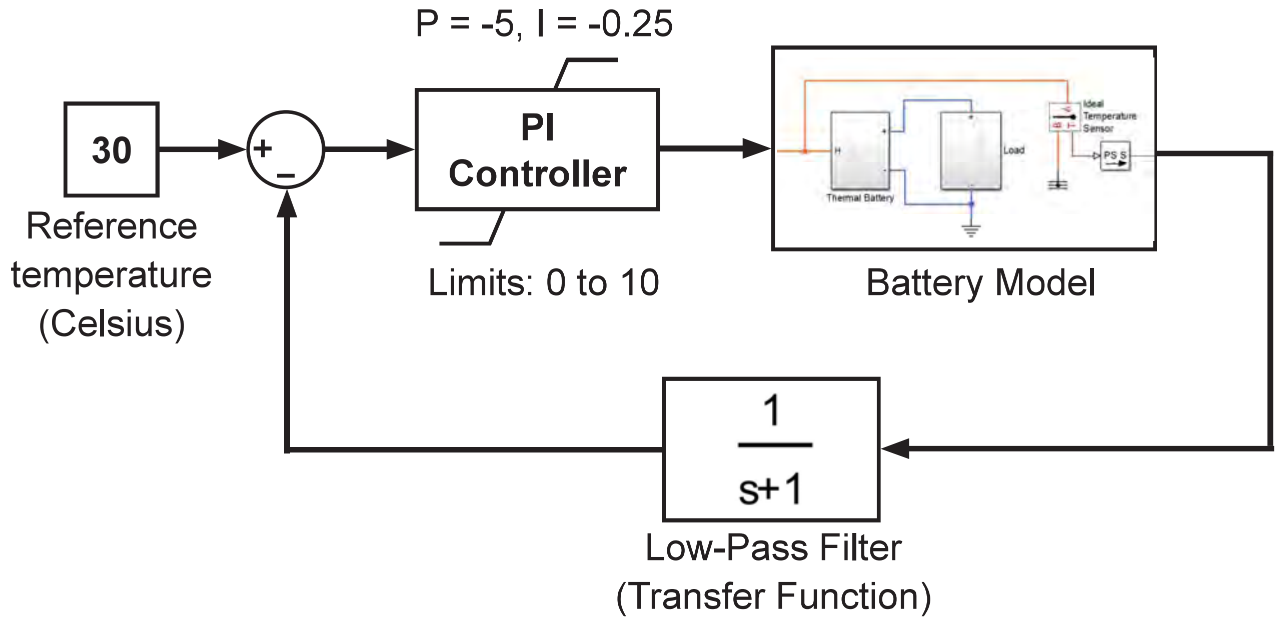
```
>> batteryCoolingStart

>> load coolerData.mat
```

3. **Model the control loop.**

   Use the schematic and parameters shown on the right to model a continuous-time PI controller and low-pass filter transfer function. Both the controller and low-pass filter can be modeled using blocks from the **Simulink** → **Continuous** library.

   **Note** The low-pass filter can be used to smooth out sensor readings. In addition, this type of transfer function is often useful for resolving algebraic loops when dealing with continuous-time systems.

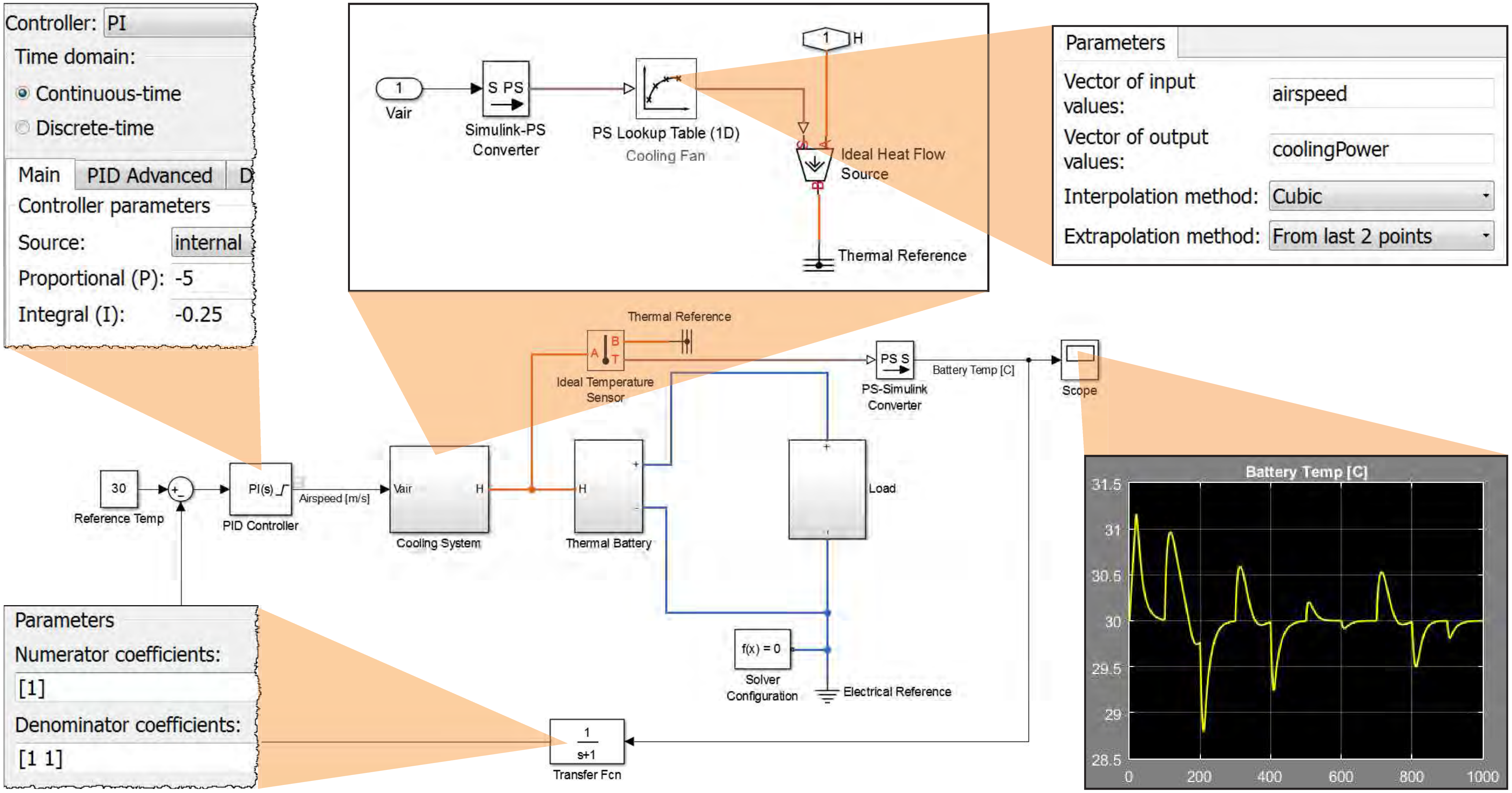


## Solution: Battery Temperature Control

The solution model looks as follows.

Try

```
>> batteryCoolingSoln
```



Controller: PI

Time domain:

[✓ Continuous-time] Discrete-time

Main | PID Advanced | D

Controller parameters

| | |
|---|---|
| Source: | internal |
| Proportional (P): | -5 |
| Integral (I): | -0.25 |

Parameters

| | |
|---|---|
| Vector of input values: | airspeed |
| Vector of output values: | coolingPower |
| Interpolation method: | Cubic |
| Extrapolation method: | From last 2 points |

Parameters

Numerator coefficients:

[1]

Denominator coefficients:

[1 1]