

**Simulink®**

Getting Started Guide



**MATLAB® & SIMULINK®**

R2020b



## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*Simulink® Getting Started Guide*

© COPYRIGHT 1990–2020 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

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

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

September 2005	Online only	New for Version 6.3 (Release 14SP3)
March 2006	Online only	Revised for Simulink 6.4 (Release 2006a)
September 2006	Online only	Revised for Simulink 6.5 (Release 2006b)
March 2007	First printing	Revised for Simulink 6.6 (Release 2007a)
September 2007	Second printing	Revised for Simulink 7.0 (Release 2007b)
March 2008	Third printing	Revised for Simulink 7.1 (Release 2008a)
October 2008	Fourth printing	Revised for Simulink 7.2 (Release 2008b)
March 2009	Fifth printing	Revised for Simulink 7.3 (Release 2009a)
September 2009	Online only	Revised for Simulink 7.4 (Release 2009b)
March 2010	Online only	Revised for Simulink 7.5 (Release 2010a)
September 2010	Online only	Revised for Simulink 7.6 (Release 2010b)
April 2011	Online only	Revised for Simulink 7.7 (Release 2011a)
September 2011	Sixth printing	Revised for Simulink 7.8 (Release 2011b)
March 2012	Seventh printing	Revised for Simulink 7.9 (Release 2012a)
September 2012	Eighth printing	Revised for Simulink 8.0 (Release 2012b)
March 2013	Ninth printing	Revised for Simulink 8.1 (Release 2013a)
September 2013	Tenth printing	Revised for Simulink 8.2 (Release 2013b)
March 2014	Eleventh printing	Revised for Simulink 8.3 (Release 2014a)
October 2014	Twelfth printing	Revised for Simulink 8.4 (Release 2014b)
March 2015	Thirteenth printing	Revised for Simulink 8.5 (Release 2015a)
September 2015	Fourteenth printing	Revised for Simulink 8.6 (Release 2015b)
October 2015	Online only	Rereleased for Simulink 8.5.1 (Release 2015aSP1)
March 2016	Fifteenth printing	Revised for Simulink 8.7 (Release 2016a)
September 2016	Sixteenth printing	Revised for Simulink 8.8 (Release 2016b)
March 2017	Seventeenth printing	Revised for Simulink 8.9 (Release 2017a)
September 2017	Eighteenth printing	Revised for Simulink 9.0 (Release 2017b)
March 2018	Nineteenth printing	Revised for Simulink 9.1 (Release 2018a)
September 2018	Twentieth printing	Revised for Simulink 9.2 (Release 2018b)
March 2019	Online only	Revised for Simulink 9.3 (Release 2019a)
September 2019	Online only	Revised for Simulink 10.0 (Release 2019b)
March 2020	Online only	Revised for Simulink 10.1 (Release 2020a)
September 2020	Online only	Revised for Simulink 10.2 (Release 2020b)



## Introduction

### 1

<b>Simulink Product Description</b> .....	<b>1-2</b>
Key Features .....	<b>1-2</b>
<b>Model-Based Design with Simulink</b> .....	<b>1-3</b>
Example Model-Based Design Workflow in Simulink .....	<b>1-4</b>
<b>System Definition and Layout</b> .....	<b>1-7</b>
Determine Modeling Objectives .....	<b>1-7</b>
Identify System Components and Interfaces .....	<b>1-8</b>
<b>Model and Validate a System</b> .....	<b>1-13</b>
Open the System Layout .....	<b>1-13</b>
Model the Components .....	<b>1-13</b>
Validate Components Using Simulation .....	<b>1-17</b>
Validate the Model .....	<b>1-19</b>
<b>Design a System in Simulink</b> .....	<b>1-23</b>
Open System Model .....	<b>1-23</b>
Identify Designed Components and Design Goals .....	<b>1-23</b>
Analyze System Behavior Using Simulation .....	<b>1-24</b>
Design Components and Verify Design .....	<b>1-26</b>

## Modeling in Simulink

### 2

<b>Simulink Block Diagrams</b> .....	<b>2-2</b>
--------------------------------------	------------

## Simple Simulink Model

### 3

<b>Create a Simple Model</b> .....	<b>3-2</b>
Open New Model .....	<b>3-3</b>
Open Simulink Library Browser .....	<b>3-5</b>
Add Blocks to a Model .....	<b>3-7</b>
Connect Blocks .....	<b>3-7</b>
Add Signal Viewer .....	<b>3-8</b>
Run Simulation .....	<b>3-8</b>
Refine Model .....	<b>3-9</b>

<b>Navigate Model</b> .....	<b>4-2</b>
Navigate Through Model Hierarchy .....	<b>4-2</b>
View Signal Attributes .....	<b>4-4</b>
Trace a Signal .....	<b>4-6</b>

# Introduction

---

- “Simulink Product Description” on page 1-2
- “Model-Based Design with Simulink” on page 1-3
- “System Definition and Layout” on page 1-7
- “Model and Validate a System” on page 1-13
- “Design a System in Simulink” on page 1-23

## **Simulink Product Description**

### **Simulation and Model-Based Design**

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

### **Key Features**

- Graphical editor for building and managing hierarchical block diagrams
- Libraries of predefined blocks for modeling continuous-time and discrete-time systems
- Simulation engine with fixed-step and variable-step ODE solvers
- Scopes and data displays for viewing simulation results
- Project and data management tools for managing model files and data
- Model analysis tools for refining model architecture and increasing simulation speed
- MATLAB Function block for importing MATLAB algorithms into models
- Legacy Code Tool for importing C and C++ code into models



## Model-Based Design with Simulink

Modeling is a way to create a virtual representation of a real-world system. You can simulate this virtual representation under a wide range of conditions to see how it behaves.

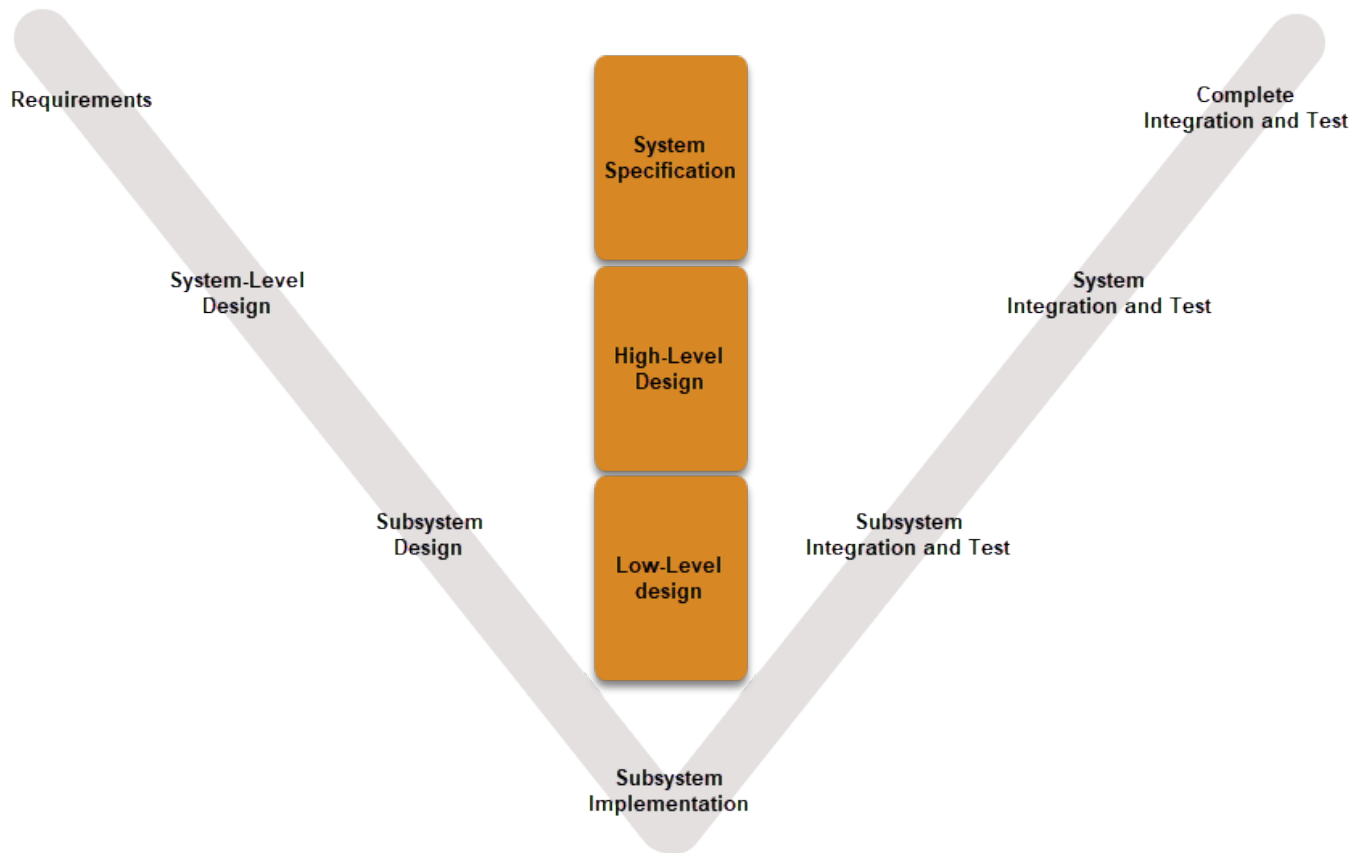
Modeling and simulation are valuable for testing conditions that are difficult to reproduce with hardware prototypes alone. This is especially true in the early phase of the design process when hardware is not yet available. Iterating between modeling and simulation can improve the quality of the system design early, by reducing the number of errors found later in the design process.

You can automatically generate code from a model and, when software and hardware implementation requirements are included, create test benches for system verification. Code generation saves time and prevents the introduction of manually coded errors.

In Model-Based Design, a system model is at the center of the workflow. Model-Based Design enables fast and cost-effective development of dynamic systems, including control systems, signal processing systems, and communications systems.

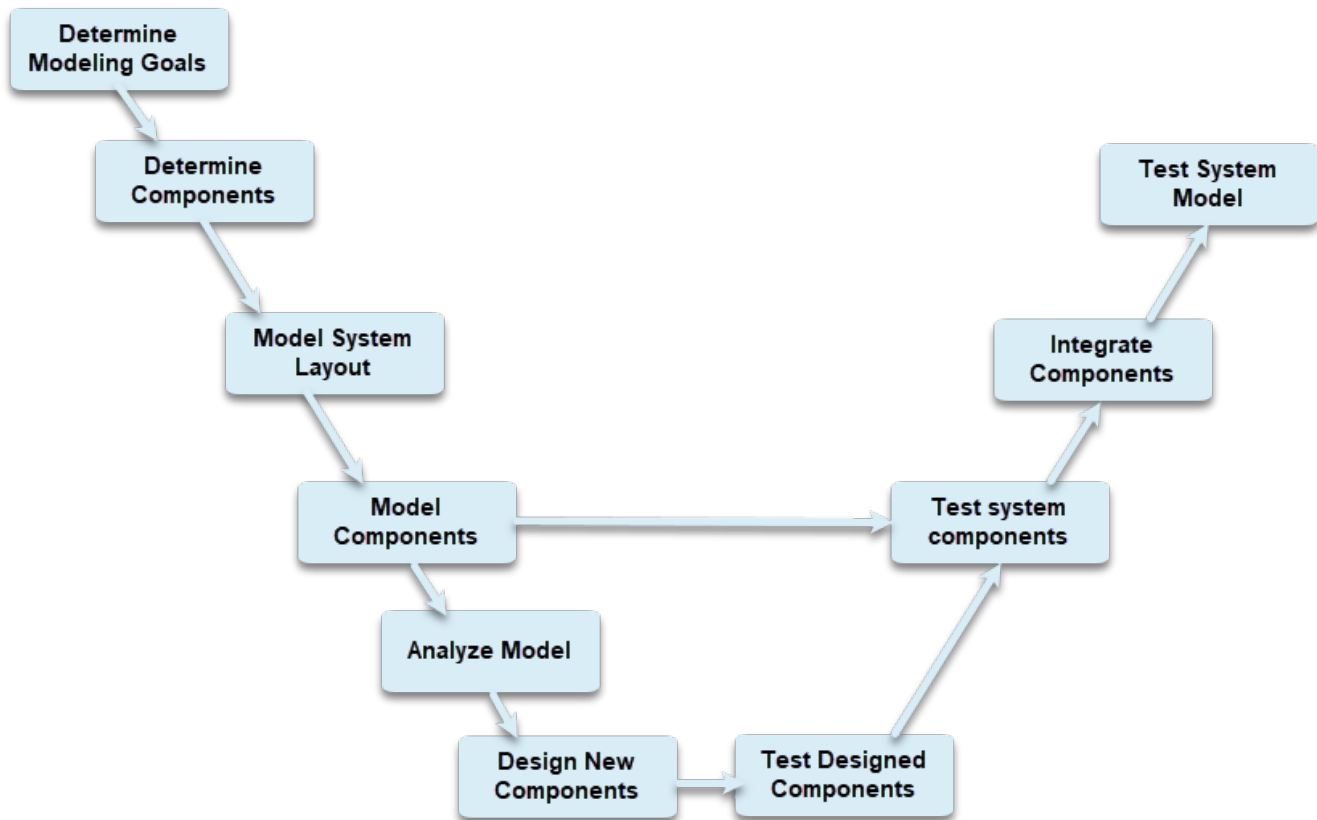
Model-Based Design allows you to:

- Use a common design environment across project teams
- Link designs directly to requirements
- Identify and correct errors continuously by integrating testing with design
- Refine algorithms through multidomain simulation
- Automatically generate embedded software code and documentation
- Develop and reuse test suites



## Example Model-Based Design Workflow in Simulink

To get started with a Model-Based Design task, consider this workflow.



The workflow in this tutorial focuses on fundamental Simulink tasks as they relate to Model-Based Design.

- “System Definition and Layout” on page 1-7 — Identify modeling goals, determine components, model system. layout
- “Model and Validate a System” on page 1-13 — Model and test components, integrate components, test system.
- “Design a System in Simulink” on page 1-23 — Design and test new components.

The first two tasks in this workflow model an existing system and establish the context for designing a component. The next step in this workflow would be to implement the new component. You can use rapid prototyping and embedded code generation products, such as Simulink Real-Time™ and Embedded Coder®, to generate code and use the design with a real, physical system.

## See Also

### Related Examples

- “System Definition and Layout” on page 1-7
- “Model and Validate a System” on page 1-13
- “Design a System in Simulink” on page 1-23
- “Organize Large Modeling Projects”

## **External Websites**

- Simulink Overview
- Model-Based Design with MATLAB and Simulink

## System Definition and Layout

### In this section...

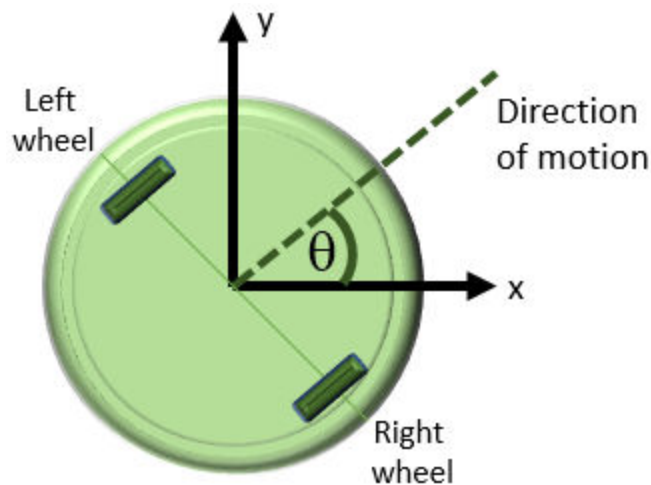
“Determine Modeling Objectives” on page 1-7

“Identify System Components and Interfaces” on page 1-8

The top-level system layout of a Simulink model is a common context that many engineering teams can use and is the basis for many tasks in the Model-Based Design paradigm: analysis, design, test, and implementation. You define a system at the top level by identifying the structure and individual components. You then organize your model in a hierarchical manner that corresponds to the components. Then you define interfaces for each component and the connections between components.

The featured model in this tutorial is a flat robot that can move or rotate with the help of two wheels, similar to a home vacuuming robot. This model assumes that the robot moves in one of two ways:

- Linear — Both wheels turn in the same direction with the same speed and the robot moves linearly.
- Rotational — The wheels turn in opposite directions with the same speed and the robot rotates in place.



Each type of motion starts from a resting state, that is, both rotational and linear speeds are zero. With these assumptions, the linear and rotational motion components can be modeled separately.

### Determine Modeling Objectives

Before designing a model, consider your goals and requirements. The goals dictate both the structure and the level of detail for the model. If the goal is simply to figure out how fast the robot can go, modeling just for linear motion is sufficient. If the goal is to design a set of inputs for the device to follow a given path, then the rotational component is involved. If obstacle avoidance is a goal, then the system needs a sensor. This tutorial builds a model with the goal of designing sensor parameters so that the robot stops in time when it detects an obstacle in its path. To achieve this goal, the model must:

- Determine how quickly the robot stops when the motors stop
- Provide a series of commands for linear and rotational motion so that the robot can move in a two-dimensional space

The first modeling objective enables you to analyze the motion so you can design the sensor. The second objective enables you to test your design.

## Identify System Components and Interfaces

Once you understand your modeling requirements, you can begin to identify the components of the system. Identifying individual components and their relationships within a top-level structure help build a potentially complex model systematically. You perform these steps outside Simulink before you begin building your model.

This task involves answering these questions:

- What are the structural and functional components of the system? When a layout reflects the physical and functional structure, it helps you to understand, build, communicate, and test the system. This becomes more important when parts of the system are to be implemented in different stages in the design process.
- What are the inputs and outputs for each component? Draw a picture showing the connections between components. This picture helps you to visualize signal flow within the model, identify the source and sink of each signal, and determine if all necessary components exist.
- What level of detail is necessary? Include major system parameters in your diagram. Creating a picture of the system can help you identify and model the parts that are essential to the behaviors you want to observe. Each component and parameter that contributes to the modeling goal must have a representation in the model, but there is a tradeoff between complexity and readability. Modeling can be an iterative process. You can start with a high-level model with few details and then gradually increase complexity where required.

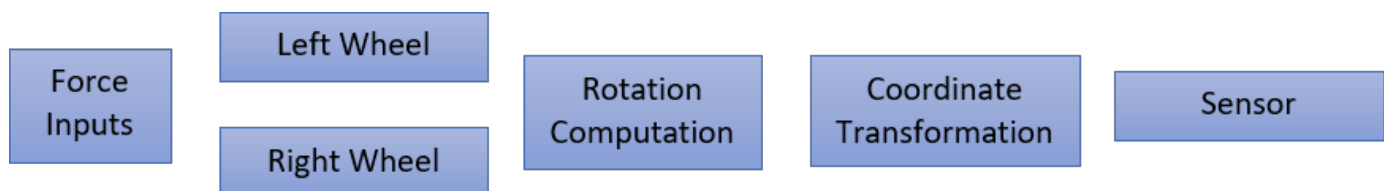
It is often beneficial to consider the following:

- What parts of the system need testing?
- What is the test data and success criteria?
- Which outputs are necessary for analysis and design tasks?

### Identify Robot Motion Components

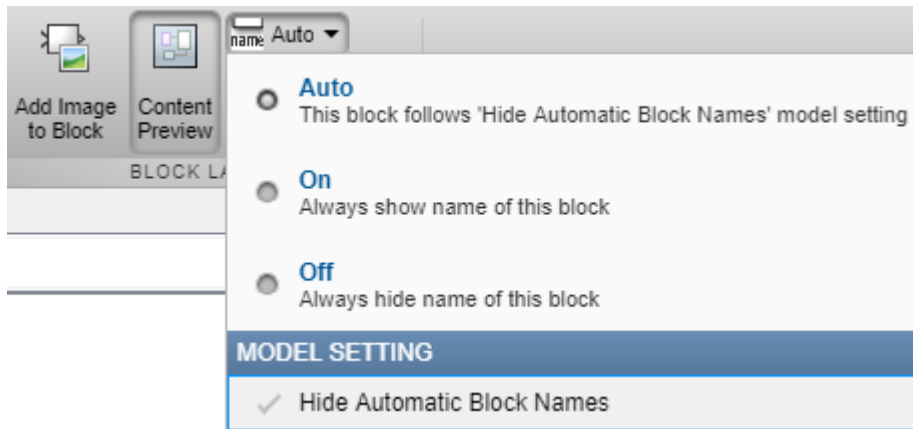
The system in this tutorial defines a robot that moves with two electric wheels in two dimensions. It includes:

- Linear motion characteristics
- Rotational motion characteristics
- Transformations to determine the location of the system in two dimensions
- A sensor to measure the distance of the robot from an obstacle

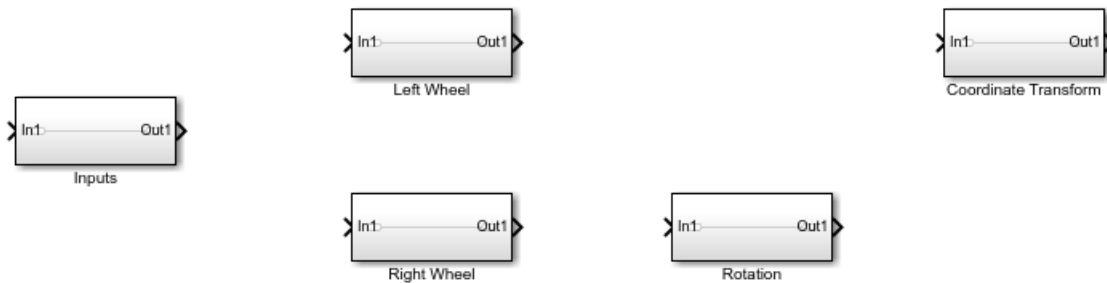


The model for this system includes two identical wheels, input forces applied to the wheels, rotational dynamics, coordinate transformation, and a sensor. The model uses a Subsystem to represent each component:

- 1 Open a new Simulink model. See “Open New Model” on page 3-3.
- 2 Open the Library Browser. See “Open Simulink Library Browser” on page 3-5.
- 3 Add Subsystem blocks. Drag five Subsystem blocks from the Ports & Subsystems library to the new model.
- 4 Click a subsystem. In the **Format** tab, click the **Auto** name drop-down. Clear the **Hide Automatic Block Names** check box.



- 5 Arrange and rename the Subsystem blocks as shown. To change block names, double-click the block name and edit the text.



### Define Interfaces Between Components

Identify input and output connections between subsystems. Input and output values change dynamically during a simulation. Lines connecting blocks represent data transfer. This table shows the inputs and outputs for each component.

Block	Input	Output	Related Information
Inputs	None	Force to right wheel Force to left wheel	Not applicable
Right wheel	Force to right wheel	Right wheel velocity	Directional, negative means reverse direction

Block	Input	Output	Related Information
Left wheel	Force to left wheel	Left wheel velocity	Directional, negative means reverse direction
Rotation	Velocity difference between right and left wheels	Rotational angle	Measured counterclockwise
Coordinate Transformation	Normal speed Rotational angle	Velocity in X Velocity in Y	Not applicable
Sensor	X coordinate Y coordinate	None	No block necessary for modeling

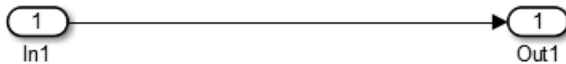
Some block inputs do not exactly match block outputs. Therefore, in addition to the dynamics of the individual components, the model must compute the following:

- Input to the rotation computation — Subtract the velocities of the two wheels and divide by two.
- Input to the coordinate transformation — Average the velocities of the two wheels.
- Input to the sensor — Integrate the outputs of the coordinate transformation.

The wheel velocities are always equal in magnitude and the computations are accurate within that assumption.


Add the necessary components and finalize connections:

- 1 Add the necessary input and output ports to each subsystem. Double-click a Subsystem block.



Each new Subsystem block contains one Inport (In1) and one Outport (Out1) block. These blocks define the signal interface with the next higher level in a model hierarchy.

Each Inport block creates an input port on the Subsystem block, and each Outport block creates an output port. The model reflects the names of these blocks as the input/output port names. Add more blocks for additional input and output signals. On the Simulink Editor toolbar, click the

**Navigate Up To Parent** button  to return to the top level.

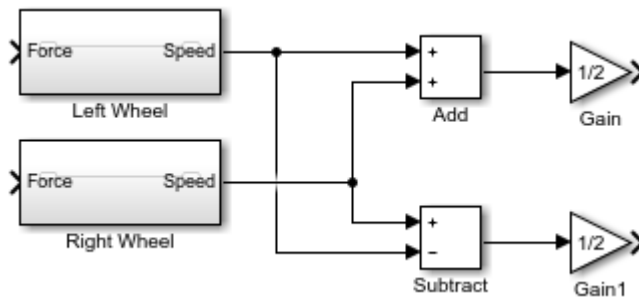
For each block, add and rename Inport and Outport blocks.



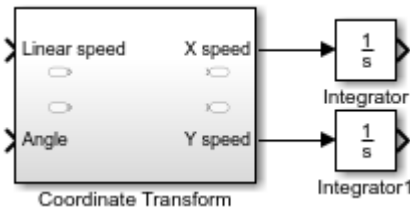


When copying an Inport block to create a new one, use the **Paste** (Ctrl+V) option.

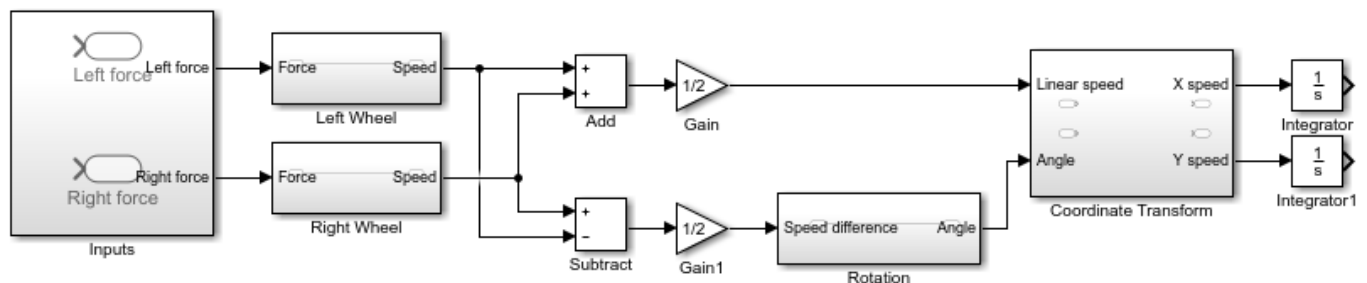
- 2 Compute required inputs to the Coordinate Transform and Rotation subsystems from the left wheel and right wheel velocities.
  - a Compute the Linear speed input to the Coordinate Transform subsystem. Add an Add block from the Math Operations library and connect the outputs of the two-wheel components. Add a Gain block and set the gain parameter to  $1/2$ . Connect the output of the Add block to this Gain block.
  - b Compute the Speed difference input to the Rotation subsystem. Add a Subtract block from the Math Operations library. Connect the right wheel velocity to the  $+$  input and the left wheel velocity to the  $-$  input. Connect the outputs of the two wheel components. Add a Gain block and set the gain parameter to  $1/2$ . Connect the output of the Subtract block to this Gain block.



- 3 Compute the X and Y coordinates from the X and Y velocities. Add two Integrator blocks from the Continuous library and connect the outputs of the Coordinate Transform block. Leave initial conditions of the Integrator blocks set to 0.



- 4 Complete the connections for the system.



### Parameters and Data

Determine the parameters that are part of the model and their values. Use modeling goals to determine whether these values are always fixed or change from simulation to simulation. Parameters

that contribute to the modeling goal require explicit representation in the model. This table helps determine the level of detail when modeling each component.

<b>Parameter</b>	<b>Block</b>	<b>Symbol</b>	<b>Value</b>	<b>Type</b>
Mass	Left Wheel	m	2.5 kg	Variable
	Right Wheel			
Rolling resistance	Left Wheel	k_drag	30 Ns <sup>2</sup> /m	Variable
	Right Wheel			
Robot radius	Rotation	r	0.15 m	Variable
Initial angle	Rotation	None	0 rad	Fixed
Initial velocities	Left Wheel	None	0 m/s	Fixed
	Right Wheel		0 m/s	
Initial (X, Y) coordinates	Integrators	None	(0, 0) m	Fixed

Simulink uses the MATLAB workspace to evaluate parameters. Set these parameters in the MATLAB command window:

```
m = 2.5;  
k_drag = 30;  
r = 0.15;
```

## See Also

### Related Examples

- “Model and Validate a System” on page 1-13
- “Design a System in Simulink” on page 1-23

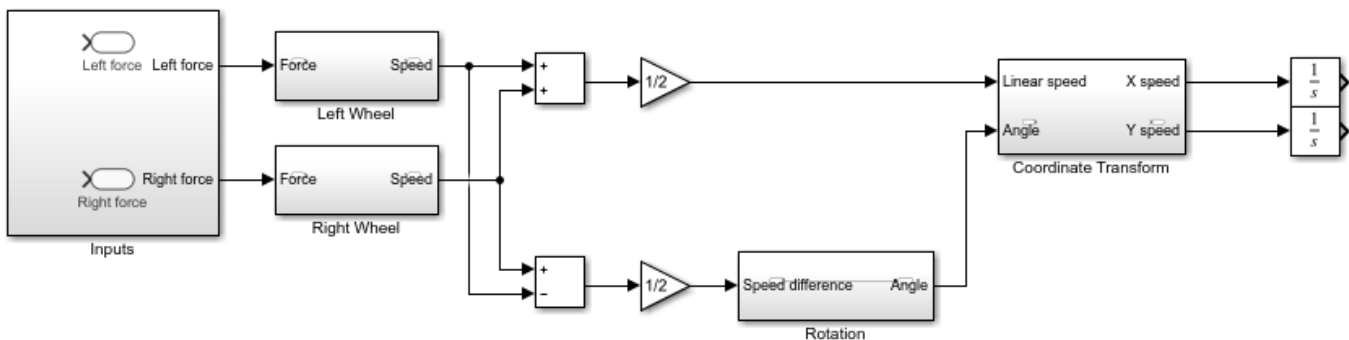
## Model and Validate a System

You model each component within the system structure to represent the physical or functional behavior of that component. You verify the basic component behavior by simulating them using test data.

### Open the System Layout

A big-picture view of the whole system layout is useful when modeling individual components. Start by loading the layout model. At the MATLAB command line, enter:

```
open_system('system_layout.slx')
```



Copyright 2018 The MathWorks, Inc.

### Model the Components

A Simulink model of a component is based on several starting points:

- An explicit mathematical relationship between the output and the input of a physical component — You can compute the outputs of the component from the inputs, directly or indirectly, through algebraic computations and integration of differential equations. For example, computation of the water level in a tank given the inflow rate is an explicit relationship. Each Simulink block executes based on the definition of the computations from its inputs to its outputs.
- An implicit mathematical relationship between model variables of a physical component — Because variables are interdependent, assigning an input and an output to the component is not straightforward. For example, the voltage at the + end of a motor connected in a circuit and the voltage at the - end have an implicit relationship. To model such a relationship in Simulink, you can either use physical modeling tools such as Simscape™ or model these variables as part of a larger component that allows input/output definition. Sometimes, closer inspection of modeling goals and component definitions helps to define input/output relationships.
- Data obtained from an actual system — You have measured input/output data from the actual component, but a fully defined mathematical relationship does not exist. Many devices have unmodeled components that fit this description. For example, the heat dissipated by a television. You can use the System Identification Toolbox™ to define the input/output relationship for such a system.

- An explicit functional definition — You define the outputs of a functional component from the inputs through algebraic and logical computations. For example, the switching logic of a thermostat. You can model most functional relationships as Simulink blocks and subsystems.

This tutorial models physical and functional components with explicit input/output relationships. In this tutorial, you will:

- 1 Use the system equations to create a Simulink model.
- 2 Add and connect Simulink blocks in the Simulink Editor. Blocks represent coefficients and variables from the equations.
- 3 Build the model for each component separately. The most effective way to build a model of a system is to first consider components independently.
- 4 Start by building simple models using approximations of the system. Identify assumptions that can affect the accuracy of your model. Iteratively add detail until the level of complexity satisfies the modeling and accuracy requirements.

### **Model the Physical Components**

Describe the relationships between components, for example, data, energy, and force transfer. Use the system equations to build a graphical model of the system in Simulink.

Some questions to ask before you begin to model a component:

- What are the constants for each component? What values do not change unless you change them?
- What are the variables for each component? What values change over time?
- How many state variables does a component have?

Derive the equations for each component using scientific principles. Many system equations fall into three categories:

- For continuous systems, differential equations describe the rate of change for variables with the equations defined for all values of time. For example, a first-order differential equation gives the velocity of a car:

$$\frac{dv(t)}{dt} = -\frac{b}{m}v(t) + u(t)$$

- For discrete systems, difference equations describe the rate of change for variables, but the equations are defined only at specific times. For example, the control signal from a discrete proportional-derivative controller:

$$pd[n] = (e[n] - e[n - 1])K_d + e[n]K_p$$

- Equations without derivatives are algebraic equations. For example, an algebraic equation gives the total current in a parallel circuit with two components:

$$I_t = I_a + I_b$$

### **Wheels and Linear Motion**

There are two forces that act on a wheel:

- Force applied by the motor — The force  $F$  acts in the direction of velocity change and is an input to the wheel subsystems.

- Drag force — The force  $F_{drag}$  acts against the direction of velocity change and is a function of velocity.

$$F_{drag} = k_{drag}V|V|$$

Acceleration is proportional to the sum of these forces:

$$(m/2)\dot{V} = F - F_{drag}$$

$$(m/2)\dot{V} = F - k_{drag}V|V|$$

$$\dot{V} = \frac{F - k_{drag}V|V|}{(m/2)}$$

Where  $k_{drag}$  is the drag coefficient and  $m$  is the mass of the robot. Each wheel carries half of this mass.

Build the wheel model:

- 1 In the `system_layout` model, double-click the Right Wheel subsystem to display the empty subsystem.
- 2 Model velocity and acceleration. Add an Integrator block. Leave the initial condition set to 0. The input of this block is the acceleration  $Vdot$  and the output is the velocity  $V$ .




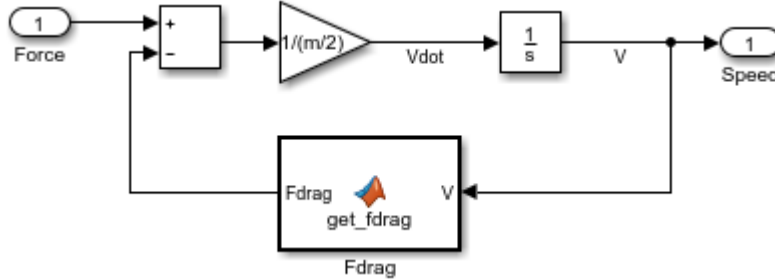
- 3 Model the drag force. Add a MATLAB Function block from the User-Defined Functions library. The MATLAB Function block provides a quick way to implement mathematical expressions in your model. To edit the function, double-click the block to open the MATLAB Function editor.

```


Right Wheel/Fdrag  x  +
1  function Fdrag = get_fdrag(V,k_drag)
2
3  Fdrag = k_drag*V*abs(V);
4

```

- 4 Define arguments for the MATLAB Function block. In the MATLAB Function block editor, click the **Edit Data**  button. Click  $k_{drag}$ , set **Scope** to **Parameter**, and click **Apply**.
- 5 Subtract the drag force from the motor force with Subtract block. Complete the force-acceleration equation with a Gain block with parameter  $1/(m/2)$ .
- 6 To reverse the direction of the MATLAB Function block, right-click the block and select **Rotate & Flip > Flip Block**. Connect the blocks.



7 The dynamics of the two wheels are the same. Make a copy of the Right Wheel subsystem you just modeled and paste it in the Left Wheel subsystem.

8 View the top level of the model. Click the **Navigate Up To Parent** button .

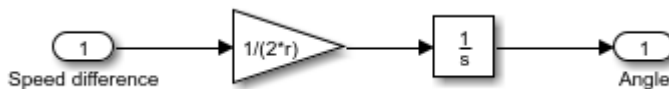
**Rotational Motion**


When the two wheels turn in opposite directions, they move in a circle of radius  $r$ , causing rotational motion of the robot. When the wheels turn in the same direction, there is no rotation. Assuming that the wheel velocities are always equal in magnitude, it is practical to model rotational motion as dependent on the difference of the two wheel velocities  $V_R$  and  $V_L$ :

$$\dot{\theta} = \frac{V_R - V_L}{2r}$$

Build the rotation dynamics model:

- 1 In the top level of the `system_layout` model, double-click the Rotation subsystem to display the empty subsystem. Delete the connection between the Inport and the Outport blocks.
- 2 Model angular speed and angle. Add an Integrator block. Leave the initial condition set to  $\theta$ . The output of this block is the angle  $theta$  and the input is the angular speed  $theta\_dot$ .
- 3 Compute angular speed from tangential speed. Add a Gain with parameter  $1/(2*r)$ .
- 4 Connect the blocks.



5 View the top level of the model. Click the **Navigate Up To Parent** button .

**Model the Functional Components**

Describe the function from the input of a function to its output. This description can include algebraic equations and logical constructs, which you can use to build a graphical model of the system in Simulink.

**Coordinate Transformation**

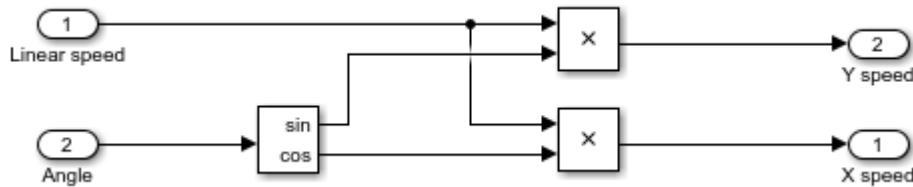
The velocity of the robot in the X and Y coordinates,  $V_X$  and  $V_Y$ , is related to the linear speed  $V_N$  and the angle  $theta$ :


$$V_X = V_N \cos(\theta)$$

$$V_Y = V_N \sin(\theta)$$

Build the coordinate transformation model:

- 1 In the top level of the `system_layout` model, double-click the Coordinate Transform subsystem to display the empty subsystem.
- 2 Model trigonometric functions. Add a SinCos block from the Math Operations library.
- 3 Model multiplications. Add two Product blocks from the Math Operations library.
- 4 Connect the blocks.



- 5 View the top level of the model. Click the **Navigate Up To Parent** button .

### Set Model Parameters

A source for model parameter values can be:

- Written specifications such as standard property tables or manufacturer data sheets
- Direct measurements on an existing system
- Estimations using system input/output

This model uses these parameters:

Parameter	Symbol	Value
Mass	$m$	2.5 kg
Rolling resistance	$k_{\text{drag}}$	30 Ns <sup>2</sup> /m
Robot radius	$r$	0.15 m

Simulink uses the MATLAB workspace to evaluate parameters. Set these parameters in the MATLAB command window:

```
m = 2.5;
k_drag = 30;
r = 0.15;
```


### Validate Components Using Simulation

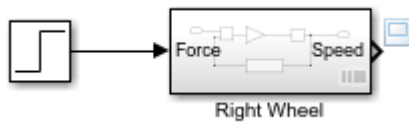
Validate components by supplying an input and observing the output. Even such a simple validation can point out immediate ways to improve the model. This example validates these behaviors:

- When a force is applied continuously to a wheel, the velocity increases until it reaches a steady-state velocity.
- When the wheels turn in opposite directions, the rotation angle increases at a constant rate.

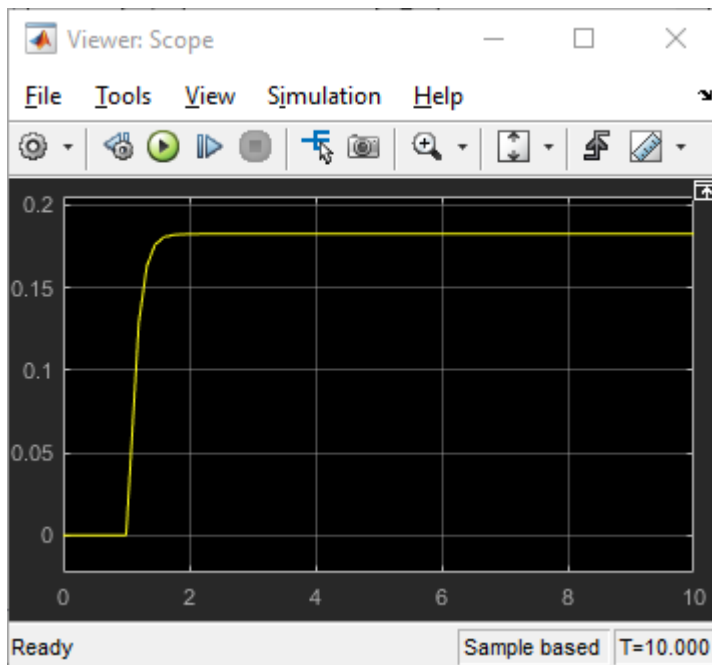
### Validate Wheel Component

Create and run a test model for the wheel component:

- 1 Create a new model. In the **Simulation** tab, click **New** . Copy the Right Wheel block into the new model.
- 2 Create a test input. Add a Step block from the Sources library and connect it to the input of the Right Wheel block. Leave the step time parameter set to 1.
- 3 Add a viewer to the output. Right-click the output port of the Right Wheel block and select **Create & Connect Viewer > Simulink > Scope**.




- 4 Run the simulation. In the **Simulation** tab, click **Run** .



The simulation result exhibits the general expected behavior. There is no motion until force is applied at step time. When force is applied, the speed starts increasing and then settles at a constant when the applied force and the drag force reach an equilibrium. In addition to validation, this simulation also gives information on the maximum speed of the wheel for the given force.

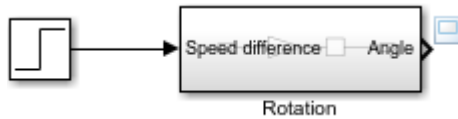
### Validate Rotation Component

Create and run a test model for the rotation model:

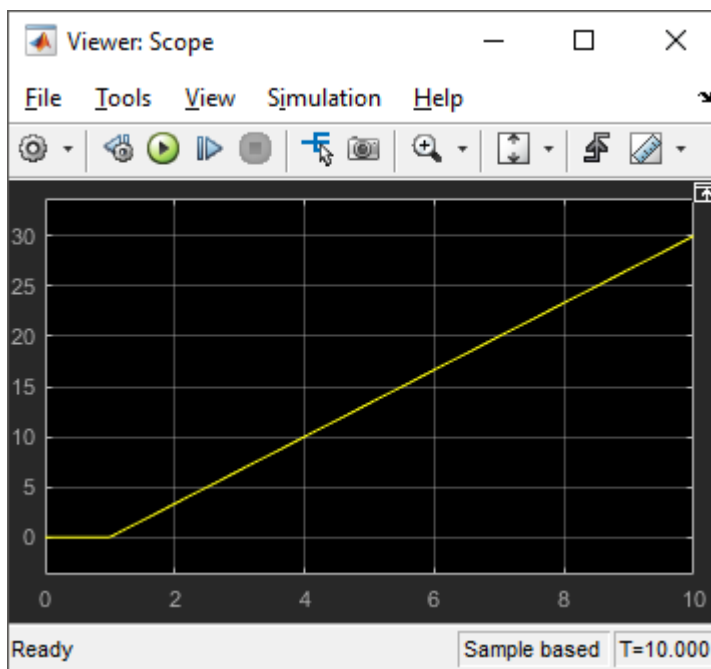
- 1 Create a new model. Click  and copy the Rotation block into the new model.



- 2 Create a test input in the new model. Add a Step block from the Sources library. Leave the step time parameter set to 1. Connect it to the input of the Rotation block. This input represents the difference of the wheel velocities when the wheels are rotating in opposite directions.
- 3 Add a viewer to the output. Right-click the output port of the Rotation block and select **Create & Connect Viewer > Simulink > Scope**.



- 4 Run the simulation. In the **Simulation** tab, click **Run** .



This simulation shows that the angle increases steadily when the wheels are turning with the same speed in opposite directions. You can make some model improvements to make it easier to interpret the angle output, for example:

- You can convert the output in radians to degrees. Add a Gain block with a gain of  $180/\pi$ .
- You can display the degrees output in cycles of 360 degrees. Add a Math Function block with function `mod`.

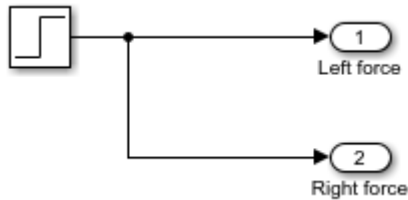
MATLAB trigonometric functions take inputs in radians.

## Validate the Model

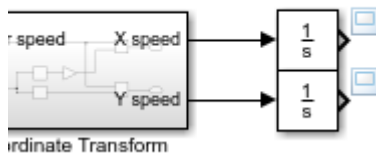
After you validate individual components, you can perform a similar validation on the complete model. This example validates the following behavior:

- When the same force is applied to both wheels in the same direction, the robot moves in a line.

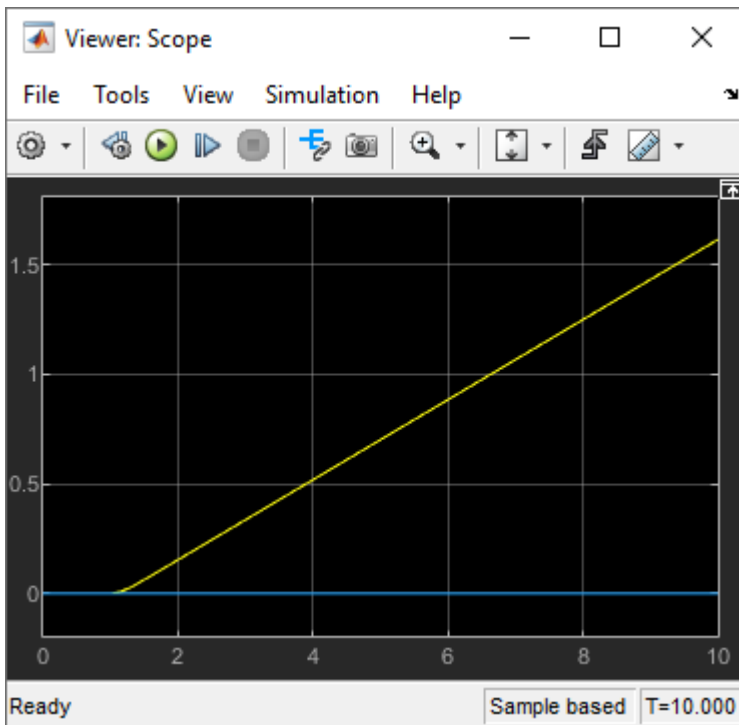
- When the same force is applied to both wheels in opposite directions, the robot rotates in place.
- 1 In the `system_layout` model, double-click the Inputs subsystem to display the empty subsystem.
  - 2 Create a test input by adding a Step block. Leave the step time parameter set to 1. Connect it to both Output blocks.



- 3 At the top level of the model, connect both output signals to the same scope viewer:

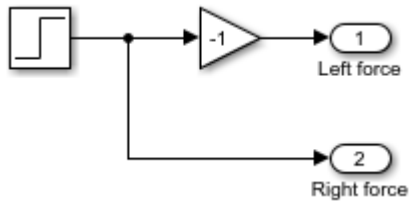


- 4 Run the model.

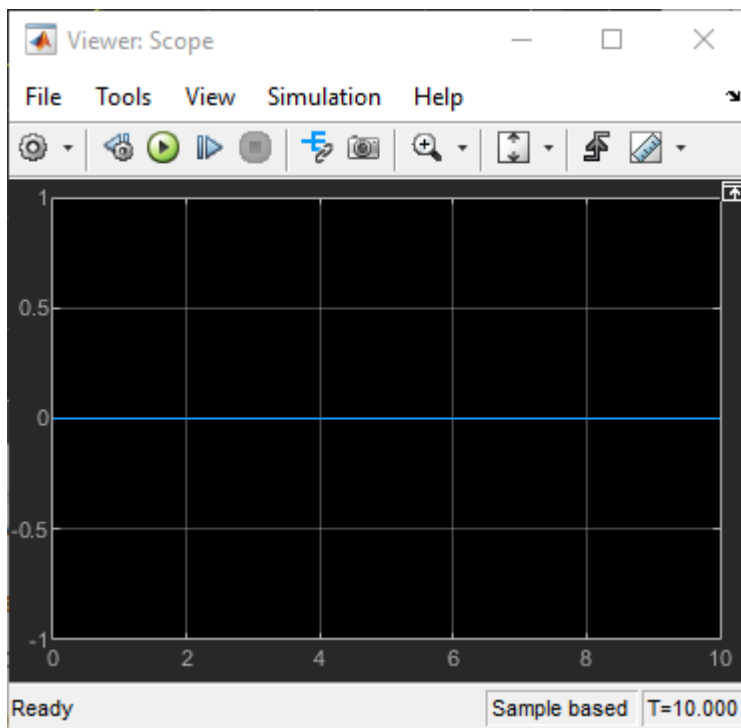


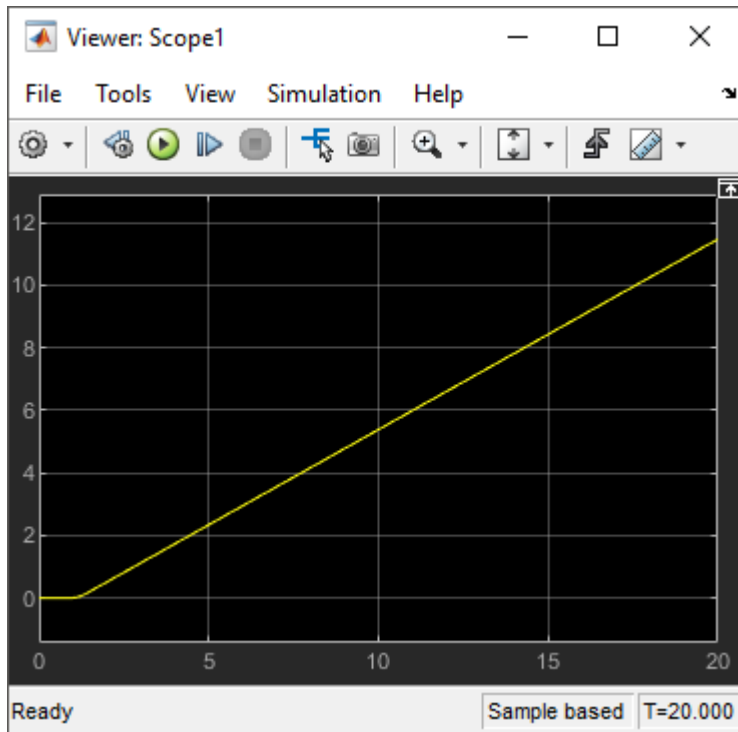
In this figure, the yellow line is the X direction and the blue line is the Y direction. Since the angle is zero and is not changing, the vehicle moves only in the X direction, as expected.

- 5 Double-click the Inputs subsystem and add a Gain with parameter -1 between the source and the second output. This reverses the direction for the left wheel.



- 6 Add a scope to the angle output.
- 7 Run the model.





The first view shows that there is no motion in the X-Y plane. The second view shows that there is steady rotation.

You can use this final model to answer many questions about the model by changing the input. Some examples are:

- What happens when the initial angle is not zero?
- How long does it take for the motion to stop when the force drops to zero?
- What happens when the robot is heavier?
- What happens when the robot moves on a smoother surface, that is, when the drag coefficient is lower?

## See Also

### Related Examples

- “System Definition and Layout” on page 1-7
- “Design a System in Simulink” on page 1-23

## Design a System in Simulink

### In this section...

“Open System Model” on page 1-23

“Identify Designed Components and Design Goals” on page 1-23

“Analyze System Behavior Using Simulation” on page 1-24

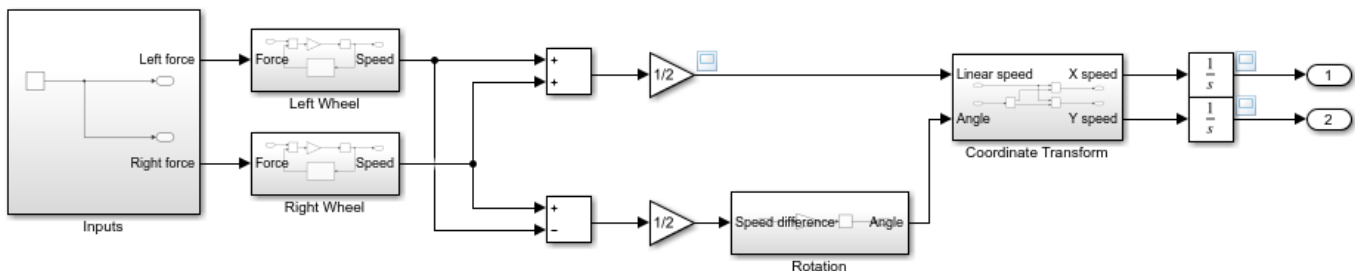
“Design Components and Verify Design” on page 1-26

Model-Based Design paradigm is centered on models of physical components and systems as a basis for design, testing, and implementation activities. This tutorial adds a designed component to an existing system model.

### Open System Model

The model is a flat robot that can move or rotate with the help of two wheels, similar to a home vacuuming robot. Open the model by entering in the MATLAB command line:

```
open_system('system_model.slx')
```



Copyright 2018 The MathWorks, Inc.

This tutorial analyzes this system and adds functionality to it.

### Identify Designed Components and Design Goals

Specification of the design objective is a critical first step to the design task. Even with a simple system, there could be multiple and even competing design goals. Consider these goals for the example model:

- Design a controller that varies the force input so that the wheels turn at a desired speed.
- Design inputs that make the device move in a predetermined path.
- Design a sensor and controller so that the device follows a line.
- Design a planning algorithm so that the device reaches a certain point using the shortest path possible while avoiding obstacles.
- Design a sensor and algorithm so that the device moves over a certain area while avoiding obstacles.

This tutorial designs an alert system. You determine the parameters for a sensor that measures the distance from an obstacle. A perfect sensor measures the distance from an obstacle accurately. An

alert system samples those measurements at fixed intervals so that the output is always within 0.05 m of the measurement. The system generates an alert in time for the robot to come to a stop before hitting the obstacle.

## Analyze System Behavior Using Simulation

The design of the new component requires analyzing the linear motion of the robot to determine:

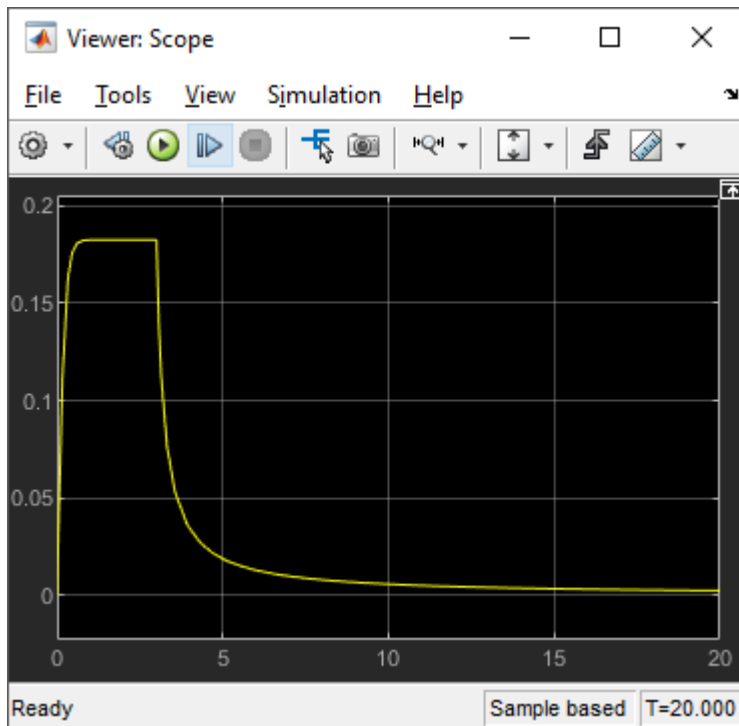
- How far the robot can travel at the top speed when power to the wheels is cut
- The top speed of the robot

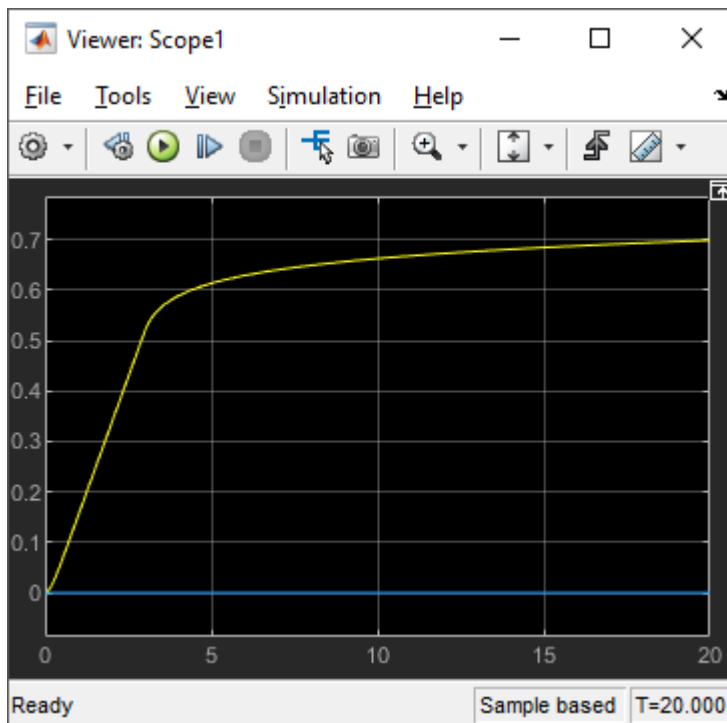
Run the model with a force input that starts motion, waits until the robot reaches a steady velocity, and then sets the input force to zero:

- 1 In the model, double-click the Inputs subsystem.
- 2 Delete the existing step input and add a Pulse Generator block.
- 3 Set parameters for the Pulse Generator block:
  - Amplitude: 1
  - Period: 20
  - Pulse Width: 15

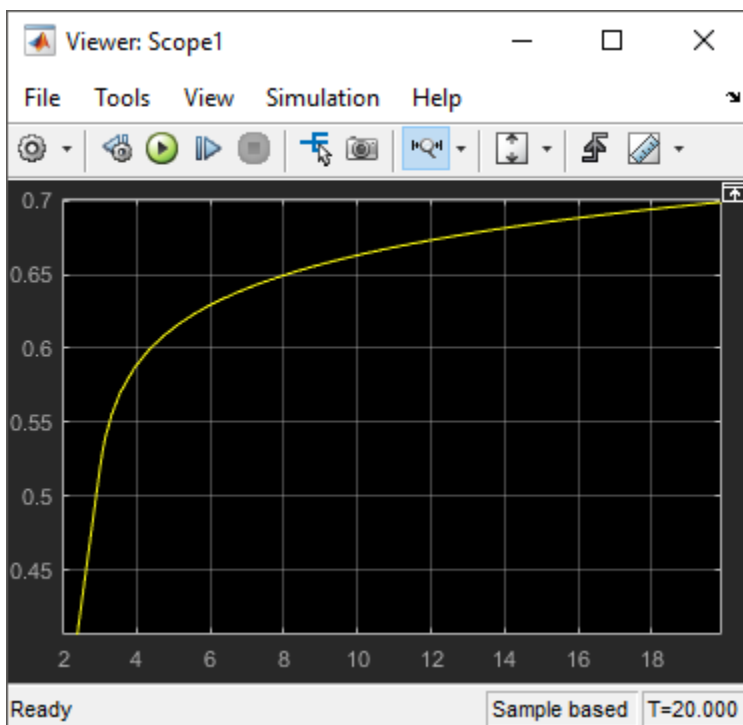
These parameters are designed to ensure that the top speed is reached. You can change parameters to see their effect.

- 4 Run the model for 20 sec.






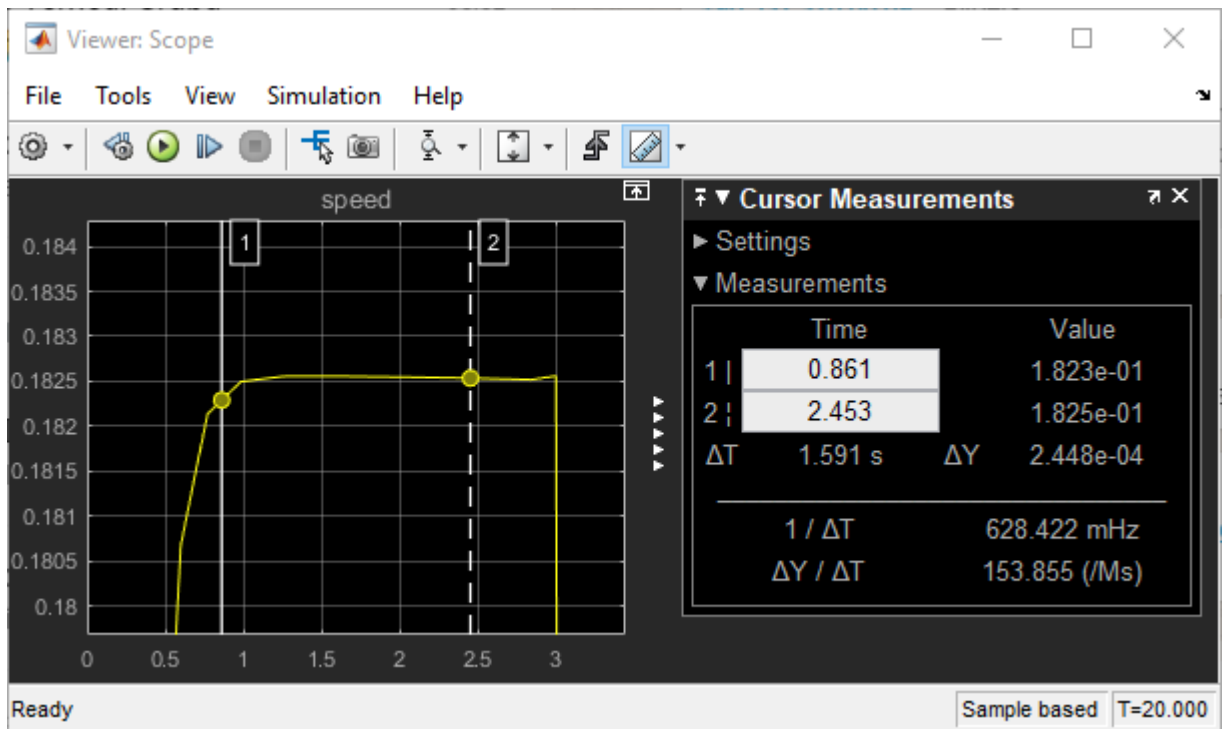
The first scope shows that the speed rapidly starts decreasing when the power is cut at time 3. The speed then asymptotically approaches zero but does not quite reach it. This is a limitation of modeling; dynamics at low speeds without external force requires a more complex representation. For the objective here, however, it is possible to make approximations. Zoom in on the position signal.



At time 3, the position of the robot is at about 0.55 m. When the simulation ends, the position is less than 0.71 m. It is safe to say that the robot travels less than 0.16 m after the power is cut.

To find the top speed:

- 1 Zoom on the stable region of the velocity output in time, from 1 s to 3 s.
- 2 Leave zoom mode by clicking the zoom button again. Click the **Cursor Measurements** button .
- 3 Set the second cursor to the region where the velocity curve is flat.



The **Value** column in the **Cursor Measurements** panel indicates that the top speed of the robot is 0.183 m/s. To calculate the time it takes for the robot to travel 0.05 m, divide 0.05 m by 0.183 m/s. You get 0.27 sec.

## Design Components and Verify Design

Sensor design consists of these components:

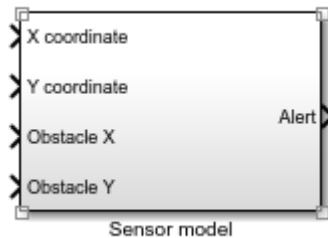
- Measurement of the distance between the robot and the obstacle — This example assumes that the measurement is perfect.
- The time interval at which the alert system measures the distance — To keep the measurement error below 0.05 m, this sampling interval must be less than 0.27 sec. Use 0.25 sec.
- The distance at which the sensor produces an alert — Analysis shows that slow down must start at 0.16 m from the obstacle. The actual alert distance must also take the error from discrete measurements, 0.05 m, into account.



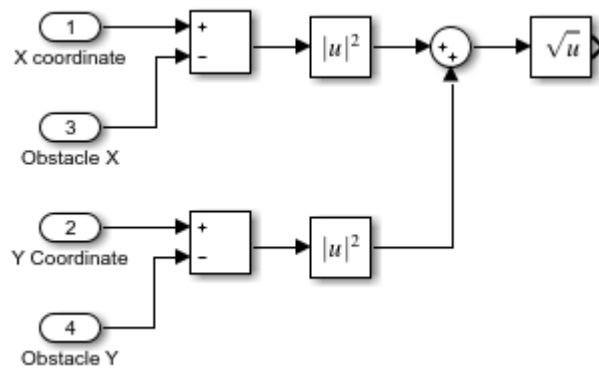
## Add Designed Component

Build the sensor:

- 1 Create a subsystem with the ports as shown.



- 2 Construct the distance measurement subsystem. In the Sensor model block, use Subtract, Math Function with function  $\text{magnitude}^2$ , Sum, and Sqrt blocks as shown. Note the reordering of the input ports.

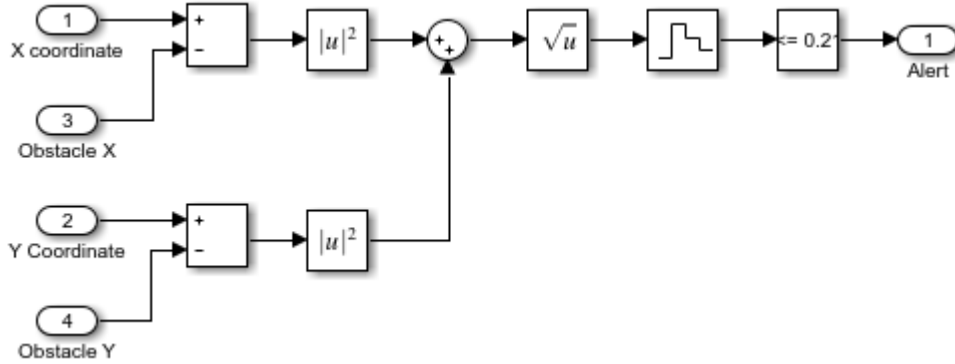


- 3 Model sampling. Add a Zero-Order Hold block from the Discrete library to the subsystem and set the **Sample time** parameter to 0.25.
- 4 Model the alert logic. Add a Compare to Constant block from the Logic and Bit Operations library and set the parameters:

- **Operator:**  $\leq$
- **Constant Value:** 0.21
- **Output data type:** boolean

This logical block sets its output to 1 when its input is less than or equal to 0.21.

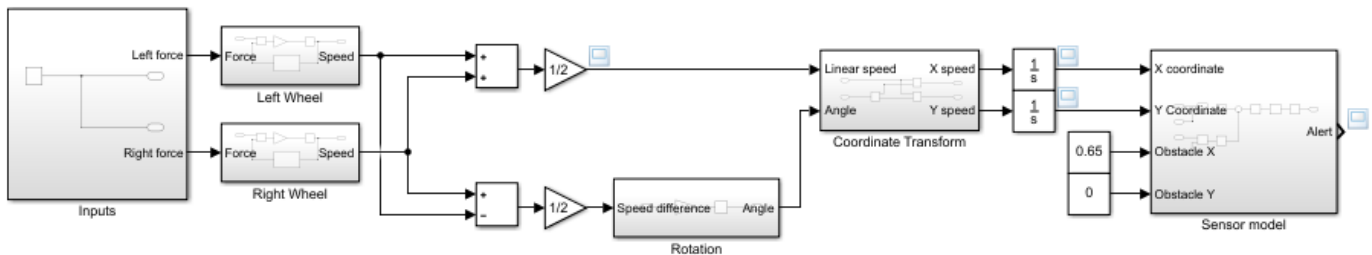
- 5 Finish connecting the blocks.



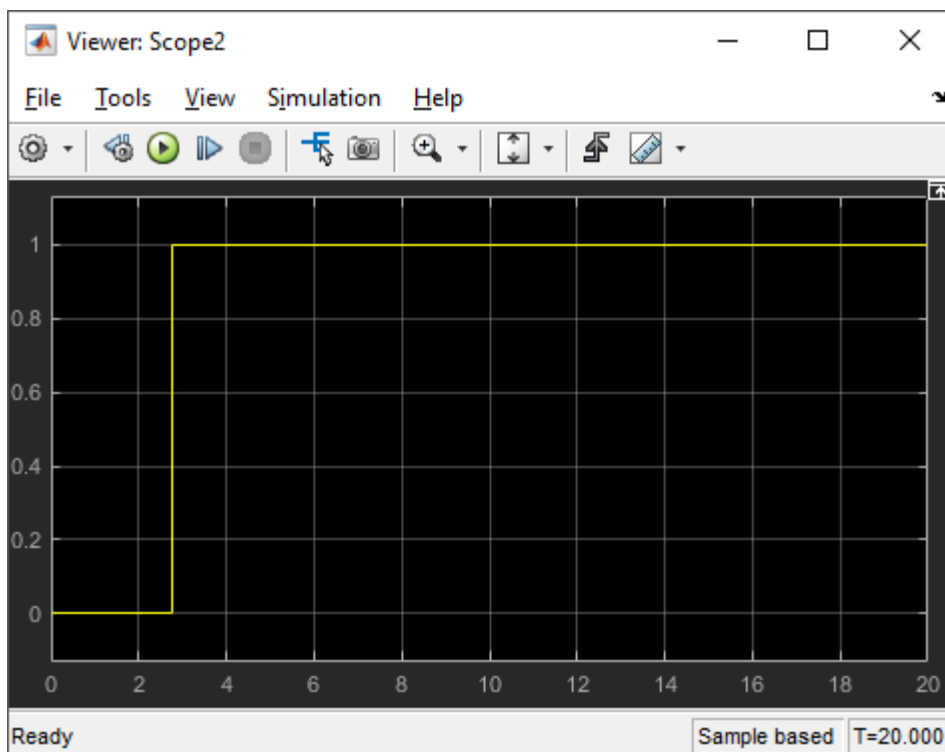
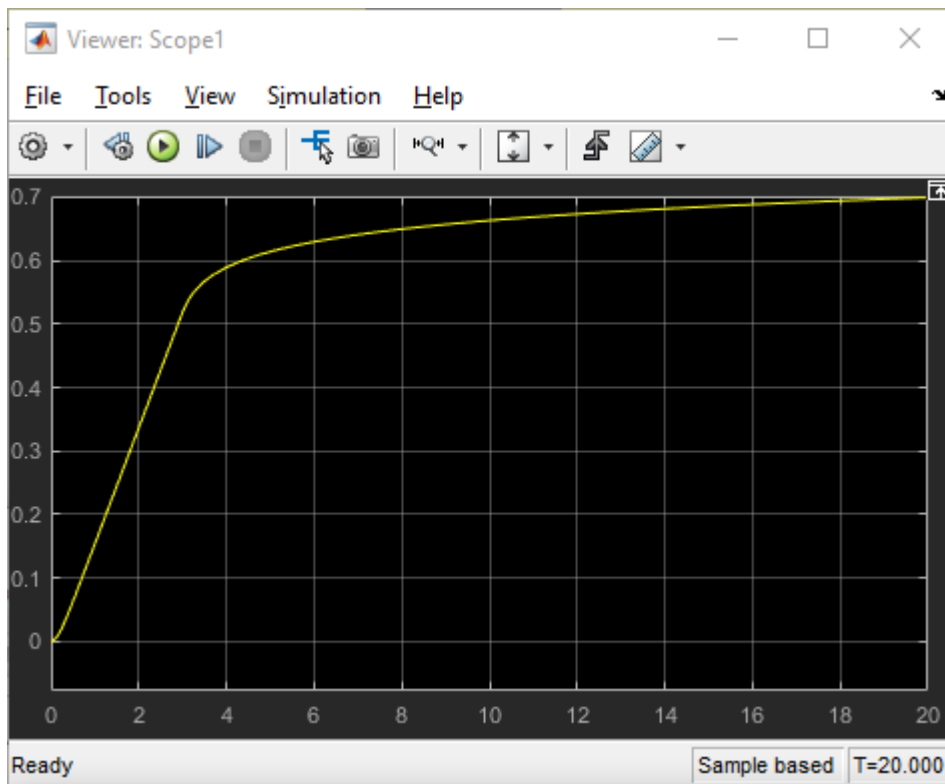
### Verify Design

Test the design with an obstacle location of  $X = 0.65$ ,  $Y = 0$  using Constant blocks as inputs to the Sensor model subsystem. This test verifies design functionality in the X direction. You can create similar tests for different paths. This model only generates an alert. It does not control the robot.

- 1 Set the obstacle location. Add two Constant blocks from the Sources library set the constant values to  $0.65$  and  $0$ . Connect the position outputs of the robot to the inputs of the sensor.
- 2 Add a scope to the Alert output.



- 3 Run the model.



Observe that the alert status becomes 1 once the position is within 0.21 m of the obstacle location and the design requirement for this component is satisfied.

For real-world systems with complex components and formal requirements, the Simulink product family includes additional tools to refine and automate the design process. Simulink Requirements™ provide tools to formally define requirements and link them to model components. Simulink Control Design™ can facilitate the design if you want to build a controller for this robot. Simulink Verification and Validation™ products establish a formal framework for testing components and systems.

## **See Also**

### **Related Examples**

- “Model-Based Design with Simulink” on page 1-3
- “System Definition and Layout” on page 1-7
- “Model and Validate a System” on page 1-13

# Modeling in Simulink

---

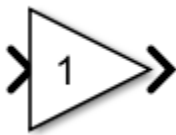
## Simulink Block Diagrams

Simulink is a graphical modeling and simulation environment for dynamic systems. You can create block diagrams, where blocks represent parts of a system. A block can represent a physical component, a small system, or a function. An input/output relationship fully characterizes a block. Consider these examples:

- A faucet fills a bucket — Water goes into the bucket at a certain flow rate, and the bucket gets heavier. A block can represent the bucket, with flow rate as the input and its weight as the output.
- You use a megaphone to make your voice heard — The sound produced at one end of the megaphone is amplified at the other end. The megaphone is the block, the input is the sound wave at its source, and the output is the sound wave as you hear it.
- You push a cart and it moves — The cart is the block, the force you apply is the input, and the cart's position is the output.

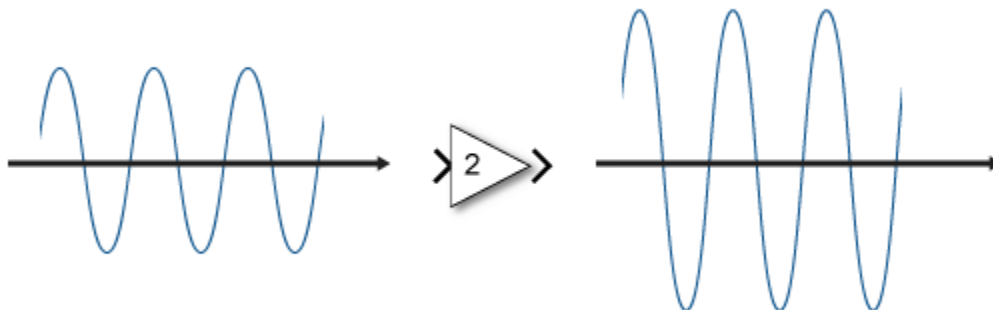
The definition of a block is only complete with its inputs and outputs defined; this task relates to the goal of the model. For example, the cart velocity may be a natural choice as an output if the modeling goal does not involve its location.

Simulink provides block libraries that are collections of blocks grouped by functionality. For example, to model a megaphone that multiplies its input by a constant, you use a Gain block from the Math Operations library.



A sound wave goes into the megaphone as its input, and a louder version of the same wave comes out as its output.

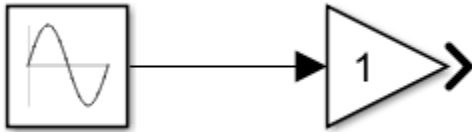
The > signs denote the inputs and outputs of a block, which can be connected to other blocks.



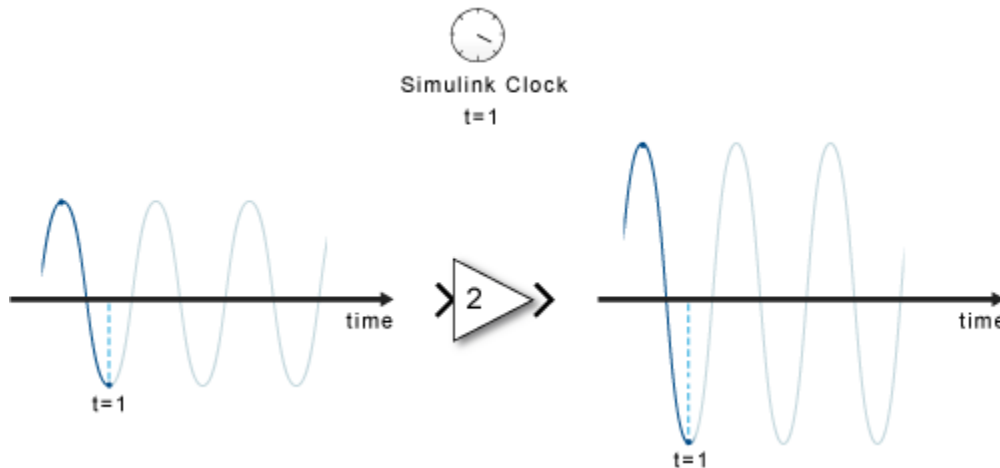
You can connect blocks to other blocks to form systems and represent more complex functionality. For example, an audio player turns a digital file into sound. A digital representation is read from storage,

is interpreted mathematically, and then turned into physical sound. The software that processes the digital file to compute the sound waveform can be one block; the speaker that takes the waveform and turns it into sound can be another block. A component that generates the input is another block.

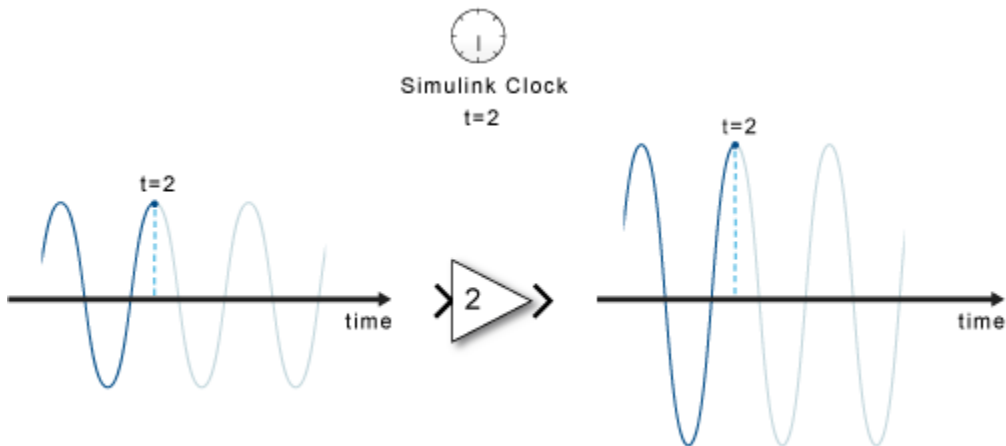
To model the sine wave input to the megaphone in Simulink, include a Sine Wave source.



The primary function of Simulink is to simulate behavior of system components over time. In its simplest form, this task involves keeping a clock, determining the order in which the blocks are to be simulated, and propagating the outputs computed in the block diagram to the next block. Consider the megaphone. At each time step, Simulink must compute the value of the sine wave, propagate it to the megaphone, and then compute the value of its output.



At each time step, each block computes its outputs from its inputs. Once all of the signals in a diagram are computed at a given time step, Simulink determines the next time step (based on the model configuration and numerical solver algorithms) and advances the simulation clock. Then each block computes their output for this new time step.



In simulation, time progresses differently from a real clock. Each time step takes as much time as it takes to finish the computations for that time step, whether that time step represents a fraction of a second or a few years.

Often, the effect of a component's input on its output is not instantaneous. For example, turning on a heater does not result in an instant change in temperature. Rather, this action provides input to a differential equation. The history of the temperature (a *state*) is also a factor. When simulation requires solving a differential or difference equation, Simulink employs memory and numerical solvers to compute the state values for the time step.

Simulink handles data in three categories:

- Signals — Block inputs and outputs, computed during simulation
- States — Internal values, representing the dynamics of the block, computed during simulation
- Parameters — Values that affect the behavior of a block, controlled by the user

At each time step, Simulink computes new values for signals and states. By contrast, you specify parameters when you build the model and can occasionally change them while simulation is running.

## See Also

### Related Examples

- “Create a Simple Model” on page 3-2
- “Model-Based Design with Simulink” on page 1-3



# Simple Simulink Model

---

## Create a Simple Model

### In this section...

“Open New Model” on page 3-3

“Open Simulink Library Browser” on page 3-5

“Add Blocks to a Model” on page 3-7

“Connect Blocks” on page 3-7

“Add Signal Viewer” on page 3-8

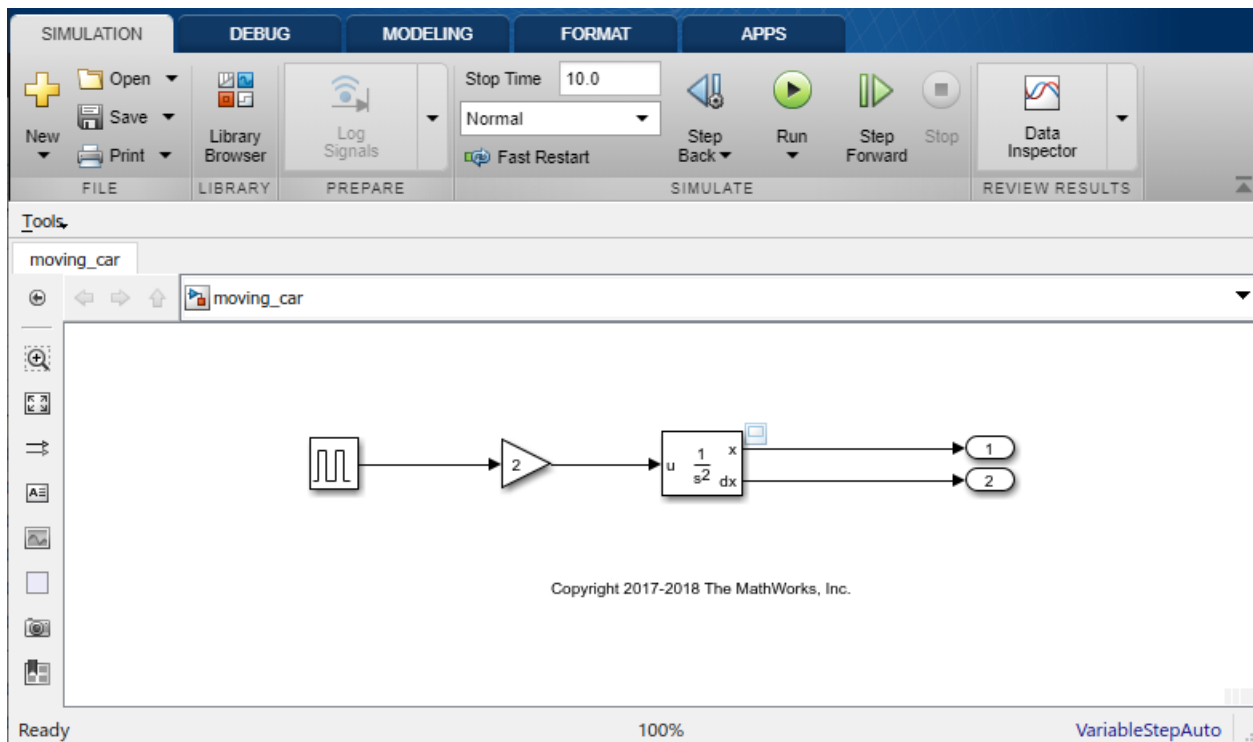
“Run Simulation” on page 3-8

“Refine Model” on page 3-9

You can use Simulink to model a system and then simulate the dynamic behavior of that system. The basic techniques you use to create a simple model in this tutorial are the same as those you use for more complex models. This example simulates simplified motion of a car. A car is typically in motion while the gas pedal is pressed. After the pedal is released, the car idles and comes to a stop.

A Simulink block is a model element that defines a mathematical relationship between its input and output. To create this simple model, you need four Simulink blocks.


Block Name	Block Purpose	Model Purpose
Pulse Generator	Generate an input signal for the model	Represent the accelerator pedal
Gain	Multiply the input signal by a constant value	Calculate how pressing the accelerator affects the car acceleration
Integrator, Second-Order	Integrate the input signal twice	Obtain position from acceleration
Outport	Designate a signal as an output from the model	Designate the position as an output from the model



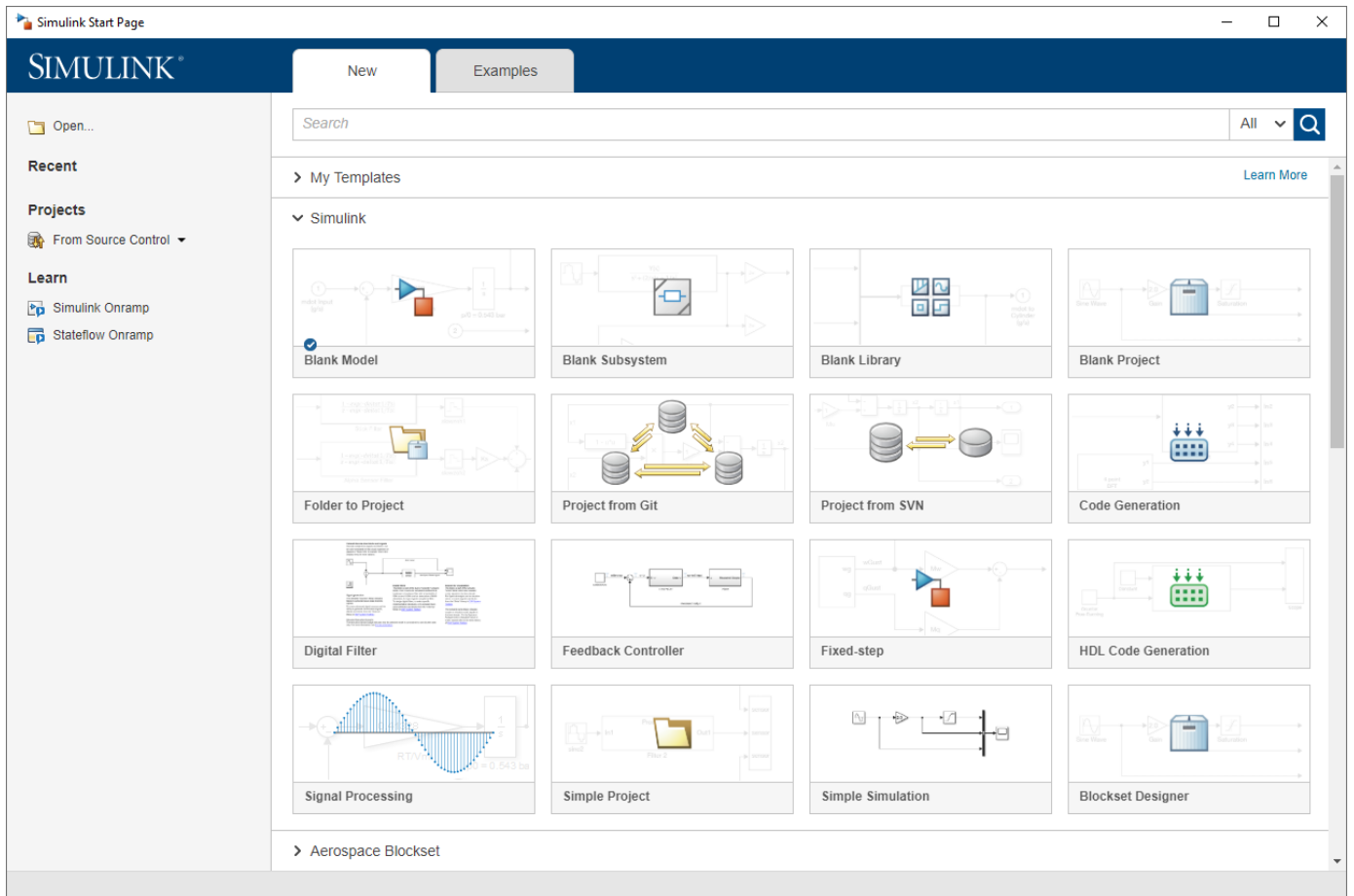
Simulating this model integrates a brief pulse twice to get a ramp. The results display in a Scope window. The input pulse represents a press of the gas pedal — 1 when the pedal is pressed and 0 when it is not. The output ramp is the increasing distance from the starting point.

## Open New Model

Use the Simulink Editor to build your models.

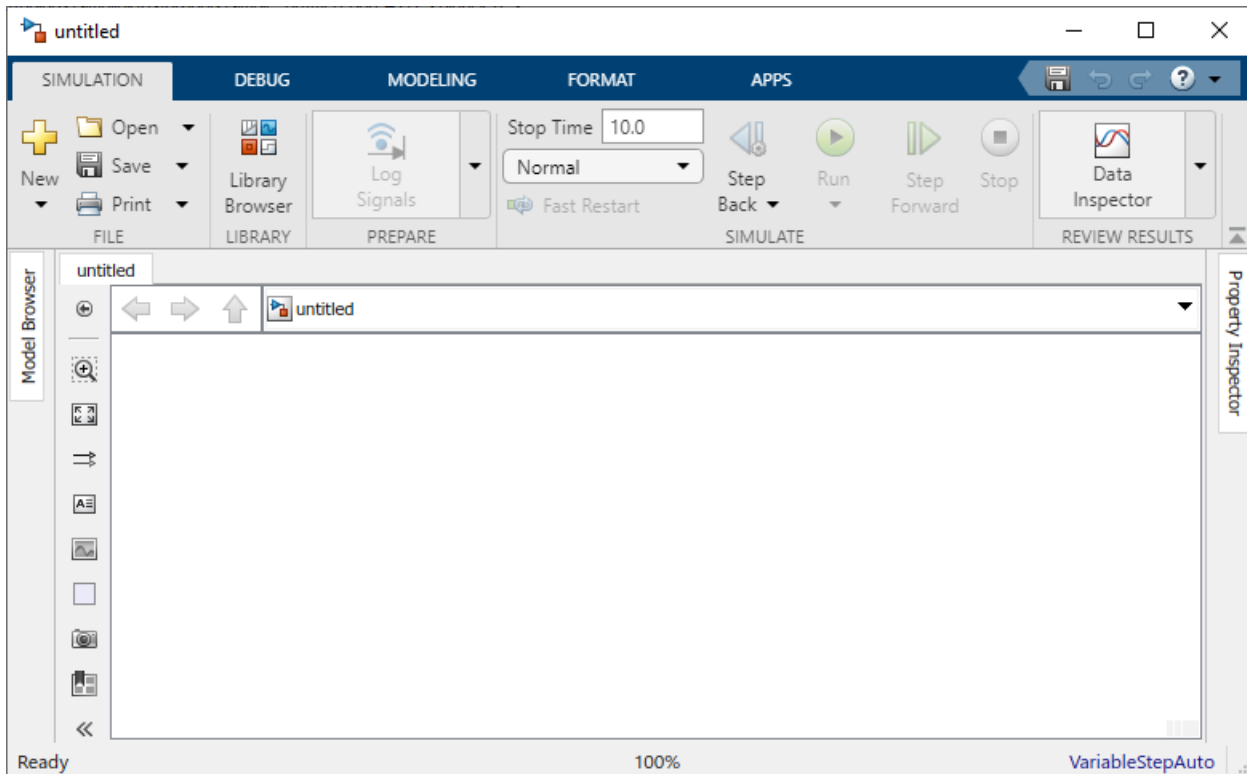
- 1 Start MATLAB. From the MATLAB toolstrip, click the **Simulink** button .

### 3 Simple Simulink Model



2 Click the **Blank Model** template.

The Simulink Editor opens.



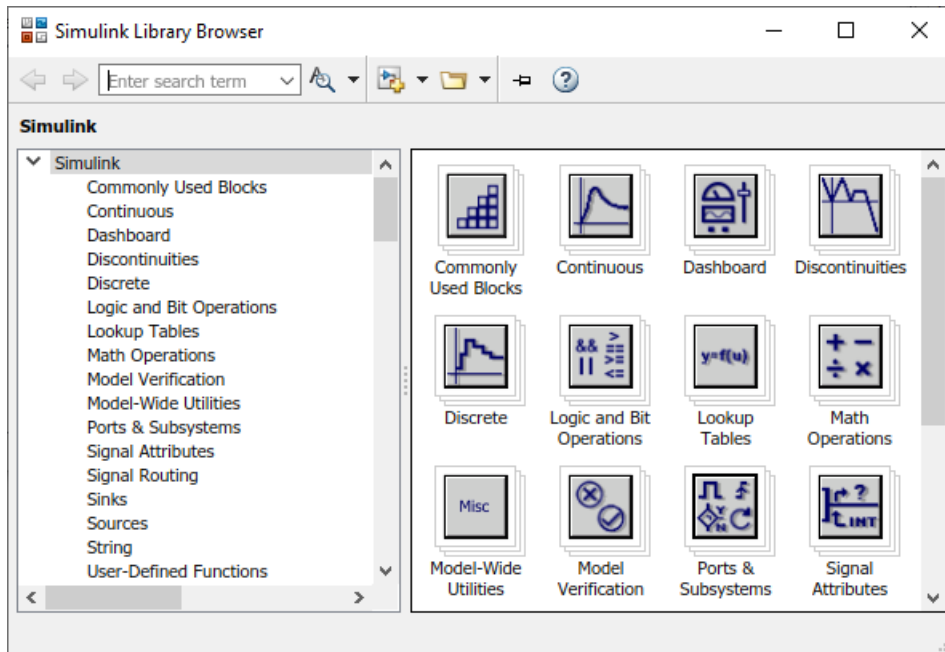
- 3 From the **Simulation** tab, select **Save > Save as**. In the **File name** text box, enter a name for your model. For example, `simple_model`. Click **Save**. The model is saved with the file extension `.slx`.


## Open Simulink Library Browser

Simulink provides a set of block libraries, organized by functionality in the Library Browser. The following libraries are common to most workflows:

- Continuous — Blocks for systems with continuous states
- Discrete — Blocks for systems with discrete states
- Math Operations — Blocks that implement algebraic and logical equations
- Sinks — Blocks that store and show the signals that connect to them
- Sources — Blocks that generate the signal values that drive the model

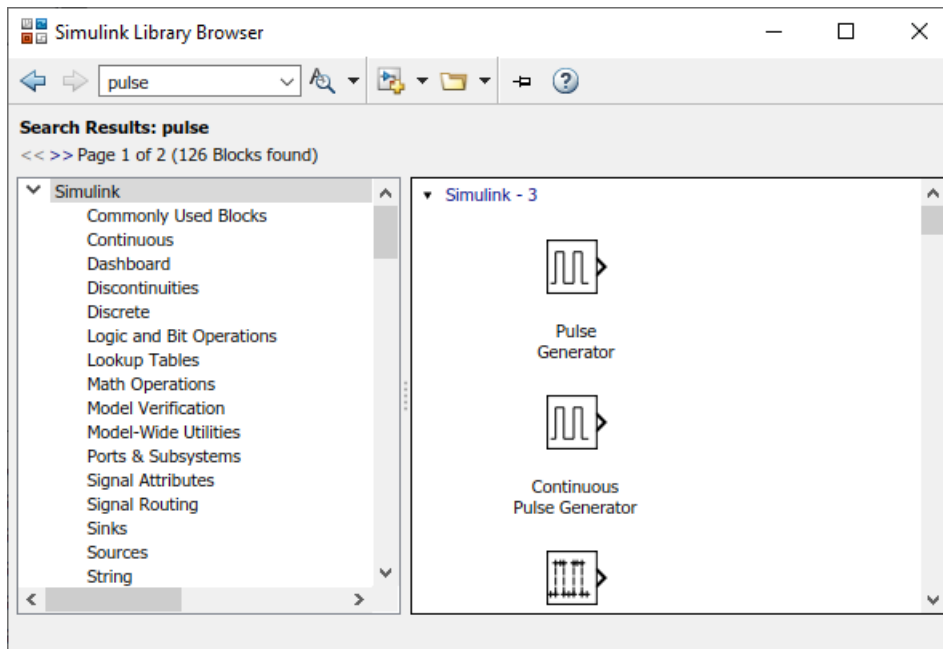
- 1 From the **Simulation** tab, click the **Library Browser** button .



- 2 Set the Library Browser to stay on top of the other desktop windows. On the Simulink Library Browser toolbar, select the **Stay on top** button  .

To browse through the block libraries, select a category and then a functional area in the left pane. To search all of the available block libraries, enter a search term.

For example, find the Pulse Generator block. In the search box on the browser toolbar, enter `pulse`, and then press Enter. Simulink searches the libraries for blocks with `pulse` in their name or description and then displays the blocks.



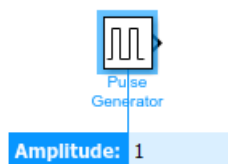
Get detailed information about a block. Right-click the Pulse Generator block, and then select **Help for the Pulse Generator block**. The Help browser opens with the reference page for the block.

Blocks typically have several parameters. You can access all block parameters by double-clicking the block.

## Add Blocks to a Model

To start building the model, browse the library and add the blocks.

- From the Sources library, drag the Pulse Generator block to the Simulink Editor. A copy of the Pulse Generator block appears in your model with a text box for the value of the **Amplitude** parameter. Enter 1.



Parameter values are held throughout the simulation.

- Add the following blocks to your model using the same approach.

Block	Library	Parameter
Gain	Simulink/Math Operations	Gain: 2
Integrator, Second-Order	Simulink/Continuous	Initial condition: 0
Output	Simulink/Sinks	Port number: 1

Add a second Output block by copying the existing one and pasting it at another point using keyboard shortcuts.

Your model now has the blocks you need.

- Arrange the blocks by clicking and dragging each block. To resize a block, drag a corner.



## Connect Blocks

Connect the blocks by creating lines between output ports and input ports.

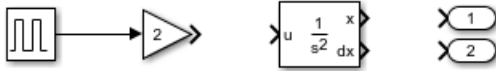
- Click the output port on the right side of the Pulse Generator block.

The output port and all input ports suitable for a connection are highlighted.

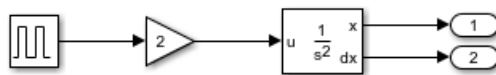


- 2 Click the input port of the Gain block.

Simulink connects the blocks with a line and an arrow indicating the direction of signal flow.



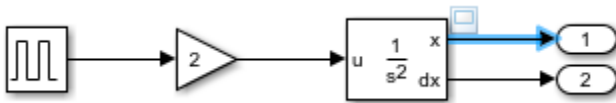
- 3 Connect the output port of the Gain block to the input port on the Integrator, Second-Order block.
- 4 Connect the two outputs of the Integrator, Second-Order block to the two Output blocks.
- 5 Save your model. In the **Simulation** tab, click the **Save** button.



### Add Signal Viewer

To view simulation results, connect the first output to a Signal Viewer.

Click the signal. In the **Simulation** tab under **Prepare**, click **Add Viewer**. Select **Scope**. A viewer icon appears on the signal and a scope window opens.

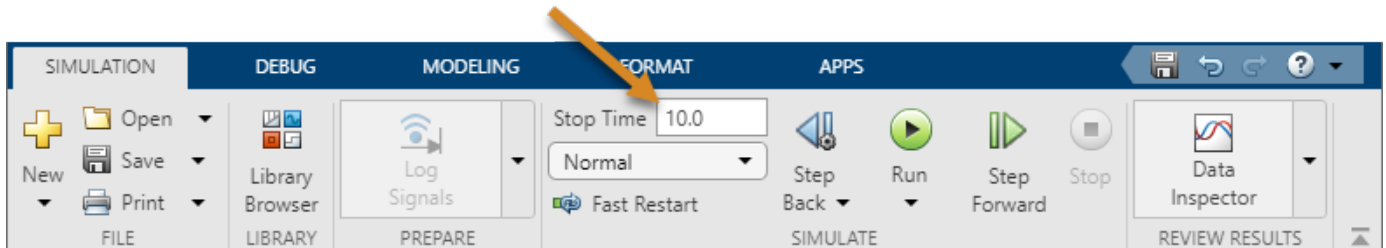


You can open the scope at any time by double-clicking the icon.

### Run Simulation

After you define the configuration parameters, you are ready to simulate your model.

- 1 In the **Simulation** tab, set the simulation stop time by changing the value in the toolbar.

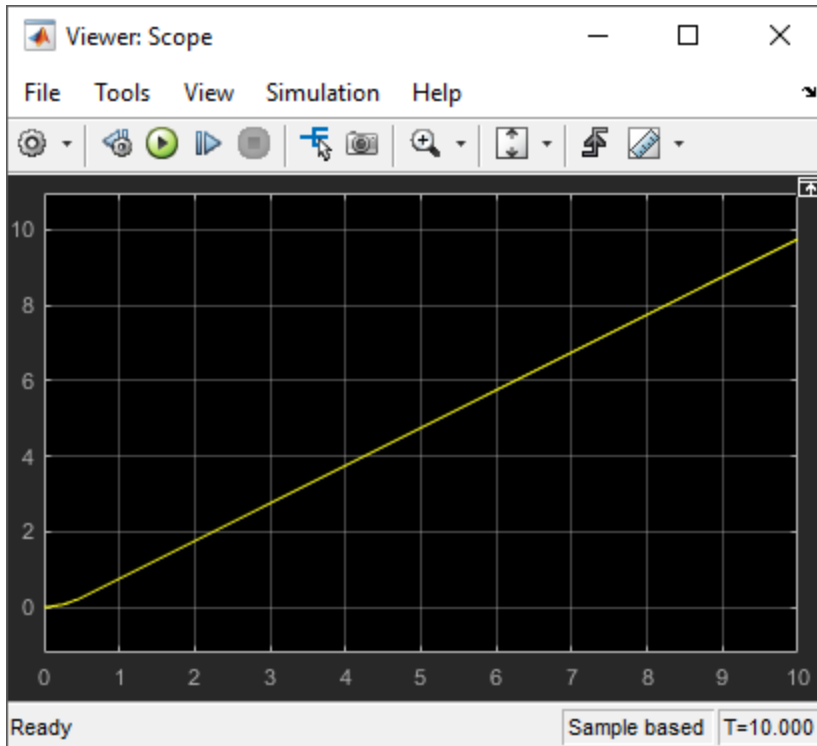


The default stop time of 10.0 is appropriate for this model. This time value has no unit. The time unit in Simulink depends on how the equations are constructed. This example simulates the simplified motion of a car for 10 seconds — other models could have time units in milliseconds or years.

- 2 To run the simulation, click the **Run** button



The simulation runs and produces the output in the viewer.



## Refine Model

This example takes an existing model, `moving_car.slx`, and models a proximity sensor based on this motion model. In this scenario, a digital sensor measures the distance between the car and an obstacle 10 m (30 ft) away. The model outputs the sensor measurement and the position of the car, taking these conditions into consideration:

- The car comes to a hard stop when it reaches the obstacle.
- In the physical world, a sensor measures the distance imprecisely, causing random numerical errors.
- A digital sensor operates at fixed time intervals.

### Change Block Parameters

To start, open the `moving_car` model. At the MATLAB command line, enter:

```
open_system('moving_car.slx')
```



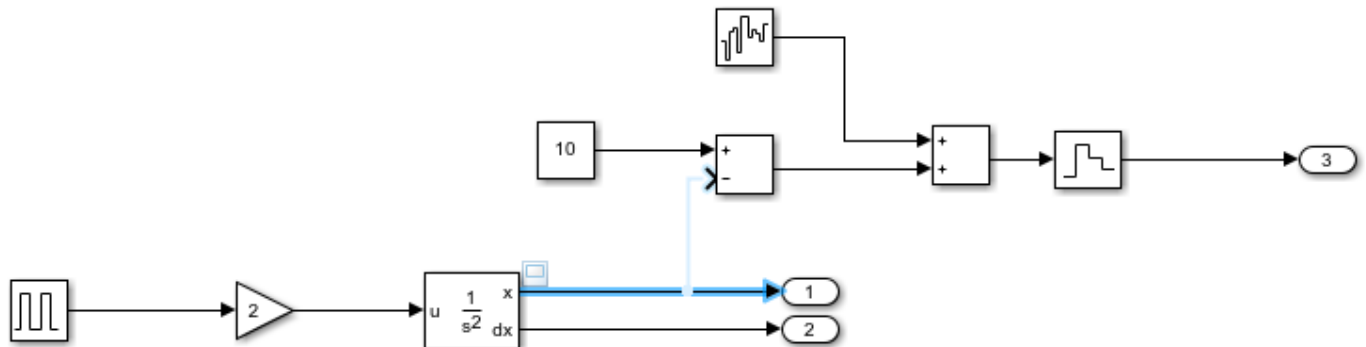
You first need to model the hard stop when the car position reaches 10 . The Integrator, Second-Order block has a parameter for that purpose.

- 1 Double-click the Integrator, Second-Order block. The Block Parameters dialog box appears.
- 2 Select **Limit x** and enter 10 for **Upper limit x**. The background color for the parameter changes to indicate a modification that is not applied to the model. Click **OK** to apply the changes and close the dialog box.

### Add New Blocks and Connections

Add a sensor that measures the distance from the obstacle.

- 1 Modify the model. Expand the model window to accommodate the new blocks as necessary.
  - Find the actual distance. To find the distance between the obstacle position and the vehicle position, add the Subtract block from the **Math Operations** library. Also add the Constant block from the **Sources** library to set the constant value of 10 for the position of the obstacle.
  - Model the imperfect measurement that would be typical to a real sensor. Generate noise by using the Band-Limited White Noise block from the **Sources** library. Set the **Noise power** parameter to 0.001. Add the noise to the measurement by using an Add block from the **Math Operations** library.
  - Model a digital sensor that fires every 0.1 seconds. In Simulink, sampling of a signal at a given interval requires a sample and hold. Add the Zero-Order Hold block from the **Discrete** library. After you add the block to the model, change the **Sample Time** parameter to 0.1.
  - Add another Outport to connect to the sensor output. Keep the default value of the **Port number** parameter.
- 2 Connect the new blocks. The output of the Integrator, Second-Order block is already connected to another port. To create a branch in that signal, left-click the signal to highlight potential ports for connection, and click the appropriate port.



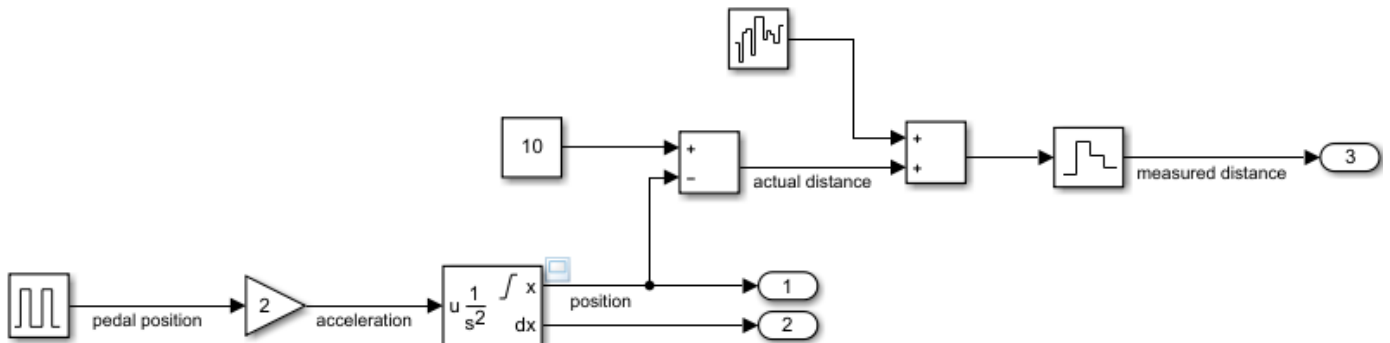
### Annotate Signals

Add signal names to the model.

- 1 Double-click the signal and type the signal name.



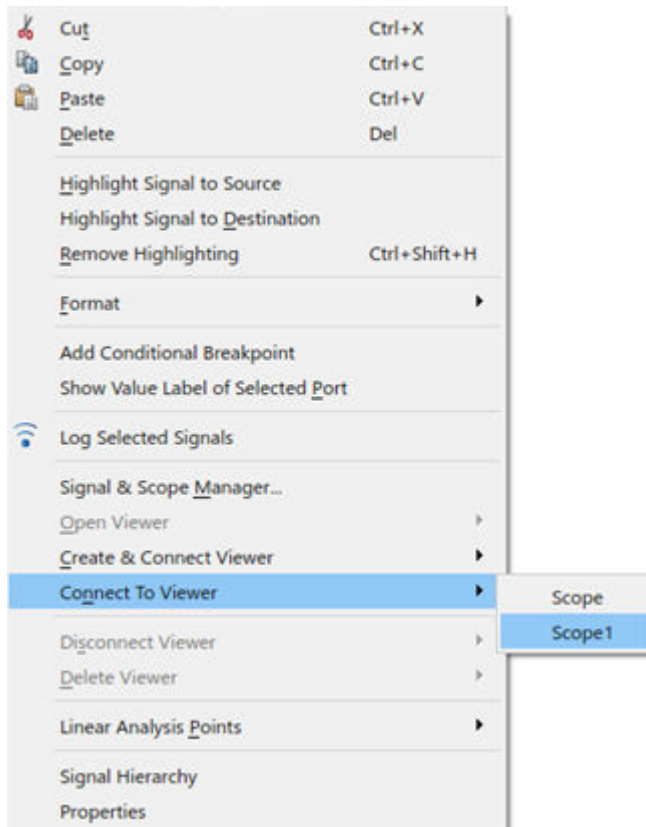
- 2 To finish, click away from the text box.
- 3 Repeat these steps to add the names as shown.



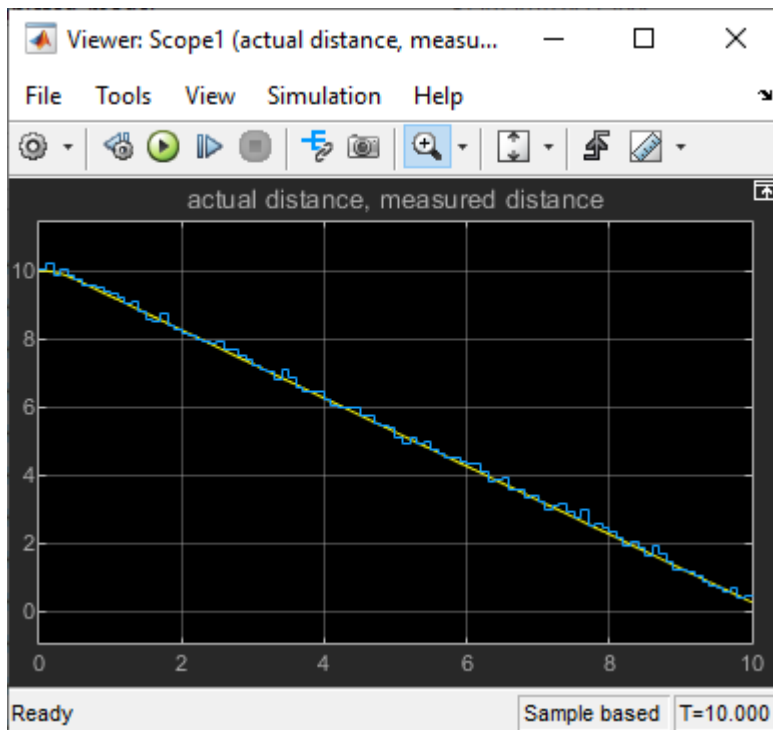
### Compare Multiple Signals

Compare the `actual distance` signal with the `measured distance` signal.


- 1 Create and connect a Scope Viewer to the `actual distance` signal. Right-click the signal and select **Create & Connect Viewer > Simulink > Scope**. The name of the signal appears in the viewer title.
- 2 Add the `measured distance` signal to the same viewer. Right-click the signal and select **Connect to Viewer > Scope1**. Make sure that you are connecting to the viewer you created in the previous step.

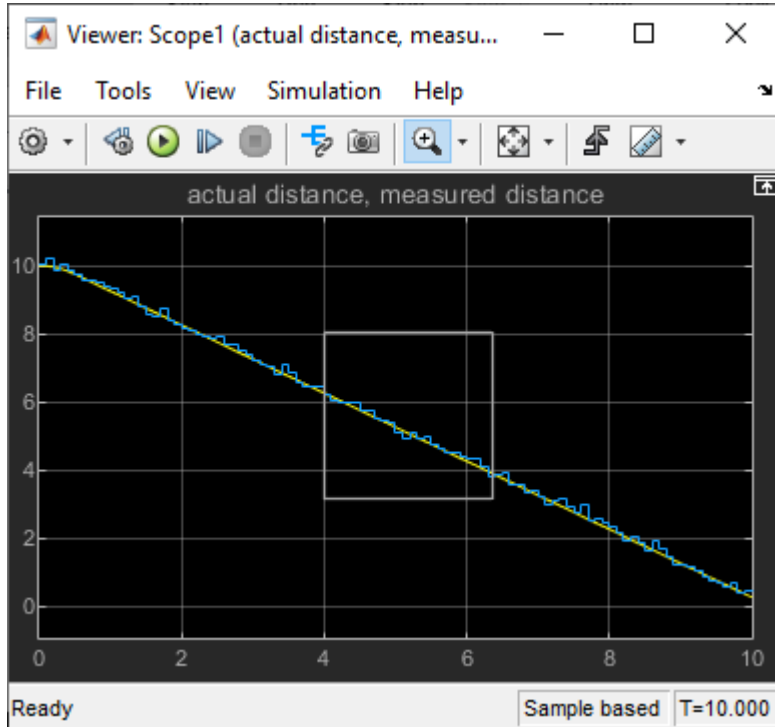


- 3 Run the model. The Viewer shows the two signals, actual distance in yellow and measured distance in blue.

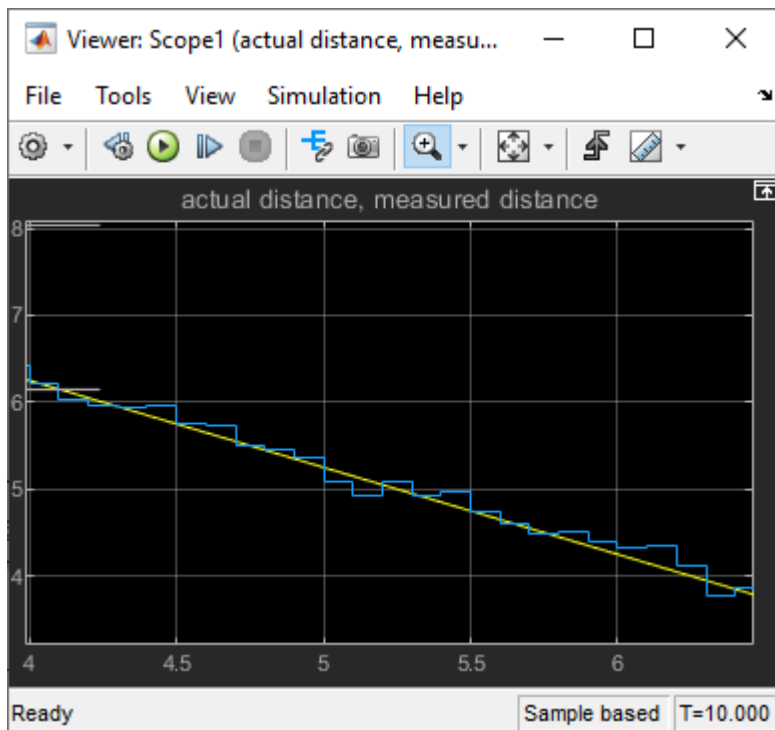


4

Zoom into the graph to observe the effect of noise and sampling. Click the **Zoom** button . Left-click and drag a window around the region you want to see more closely.



You can repeatedly zoom in to observe the details.



From the plot, note that the measurement can deviate from the actual value by as much as 0.3 m. This information becomes useful when designing a safety feature, for example, a collision warning.

### See Also

#### Blocks

Add | Band-Limited White Noise | Constant | Gain | Pulse Generator | Second-Order Integrator | Zero-Order Hold

### Related Examples

- “Model and Validate a System” on page 1-13

# Navigate a Simulink Model

---

## Navigate Model

### In this section...

“Navigate Through Model Hierarchy” on page 4-2

“View Signal Attributes” on page 4-4

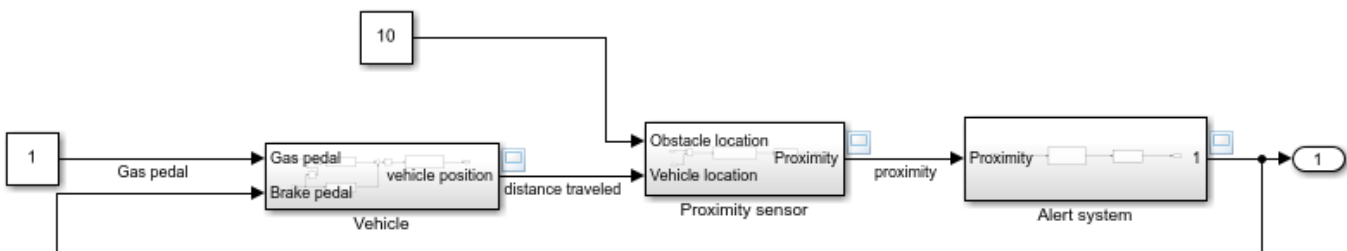
“Trace a Signal” on page 4-6

Simulink models can be organized into hierarchical components. In a hierarchical model, you can choose to view the system at a high level, or navigate down the model hierarchy to see increasing levels of model detail.

### Navigate Through Model Hierarchy

To start, open the `smart_braking` model. At the MATLAB command line, enter:

```
open_system('smart_braking.slx')
```



Copyright 2018 The MathWorks, Inc.

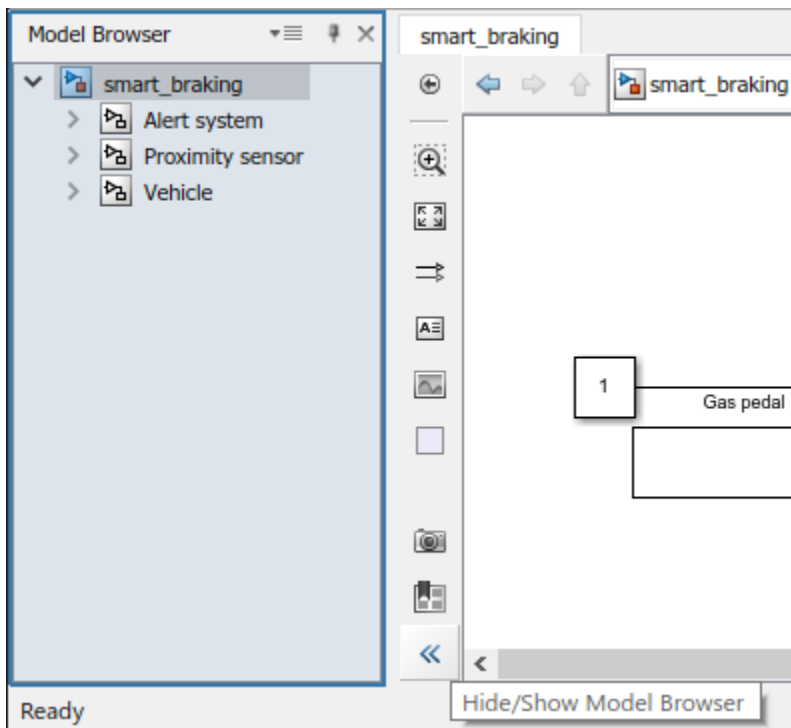
In the model:

- A vehicle moves as the gas pedal is pressed.
- A proximity sensor measures the distance between the vehicle and an obstacle.
- An alert system generates an alarm based on that proximity.
- The alarm automatically controls the brake to prevent a collision.

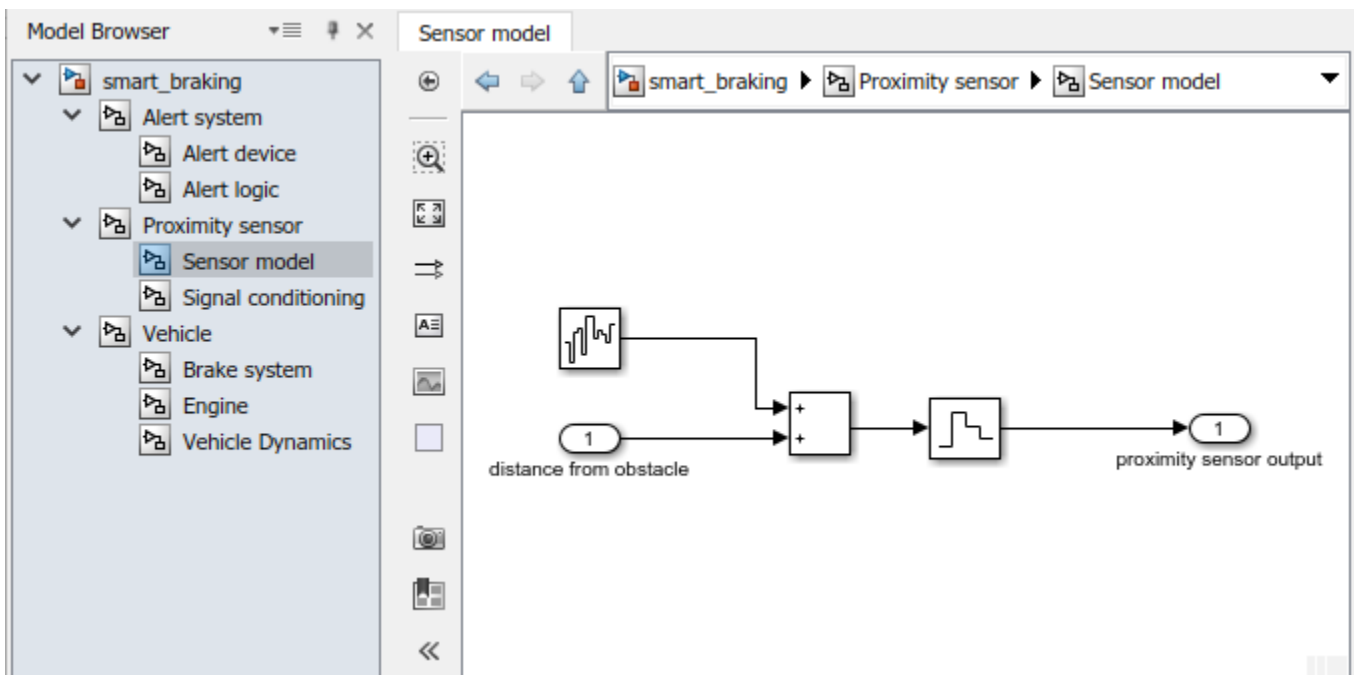
When you build a model, you connect blocks together to model complex components that represent system dynamics. In this model, Vehicle, Proximity sensor, and Alert system are all complex components with multiple blocks that exist in a hierarchy of subsystems. To view the contents of a subsystem, double-click the subsystem.

To view a representation of the complete model hierarchy, click the **Hide/Show Model Browser** button at the bottom left corner of the model window.





The Model Browser shows that all subsystems you view at the top level have subsystems of their own. Expand each subsystem node to see the subsystems it contains. You can navigate through the hierarchy in the Model Browser. For example, expand the Proximity sensor node and then select the Sensor model subsystem.



The address bar shows which subsystem you are viewing. To open the subsystem in a separate window, right-click the subsystem and select **Open In New Window**.

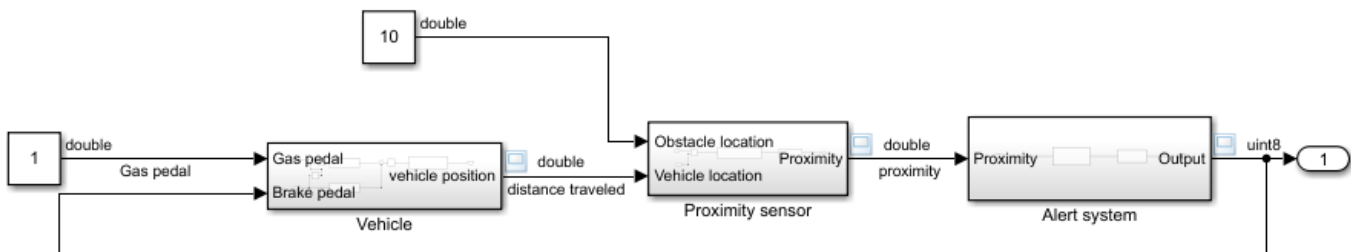
Every input or output port on a subsystem has a corresponding Inport or Outport block inside the subsystem. These blocks represent data transfer between a subsystem and its parent. When a system contains multiple input or output ports, the number on the Inport or Outport blocks indicates the position of the port on the subsystem interface.

## View Signal Attributes

Signal lines in Simulink indicate data transfer from block to block. Signals have properties corresponding to their function in the model:

- Dimensions — Scalar, vector, or matrix
- Data type — String, double, unsigned integer, etc.
- Sample time — A fixed time interval at which the signal has an updated value, or continuous sampling

To show the data type of all signals in a model, in the **Debug** tab, under **Information Overlays**, click **Base Data Types**.



The model displays data types along the signal lines. Most signals are double, except the output of the Alert system. Double-click the subsystem to investigate.



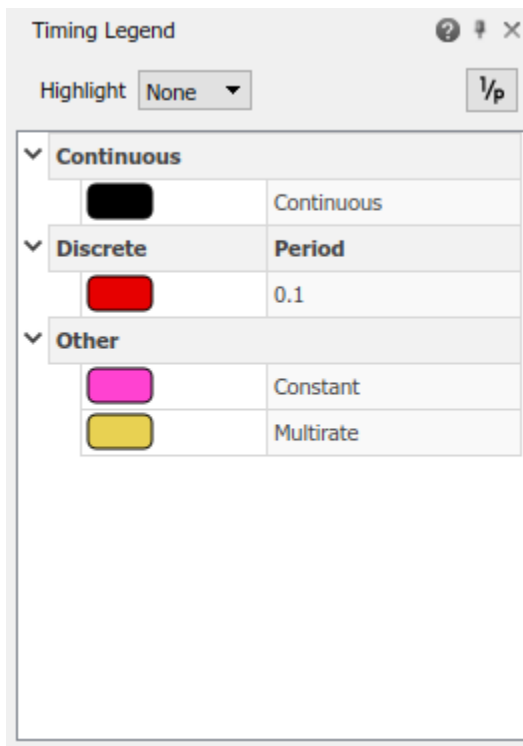
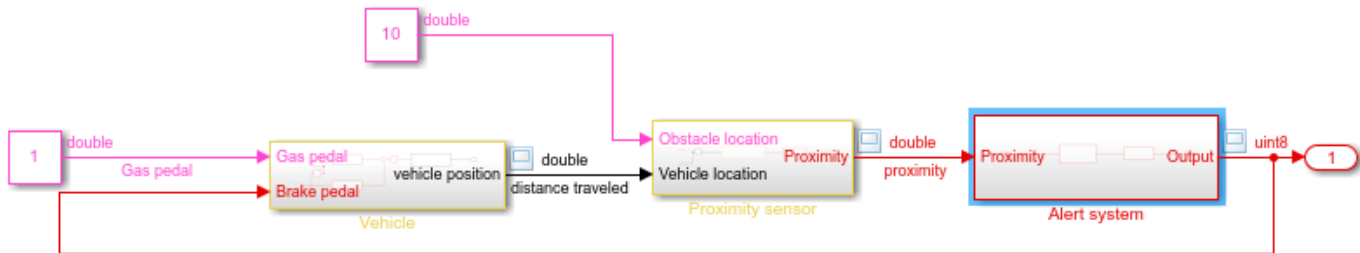
The data type labels in this subsystem show that data type change occurs in the Alert device subsystem. Double-click the subsystem to investigate.



The Alert device component converts the `Alert index` signal from a double to an integer. You can set the data type at sources, or use a Data Type Conversion block from the Signal Attributes library. Double, the default data type, provides the best numerical precision and is supported in all blocks.

The double data type also uses the most memory and computing power. Other numerical data types can be used to model embedded systems where memory and computing power are limited.

To show sample times, in the **Debug** tab, under **Information Overlays**, click **Colors** from the Sample Time section. The model updates to show different colors for each sample time in the model, along with a legend.




- A block or signal with continuous dynamics is black. Signals with continuous sample time update as often as Simulink requires to make the computations as close to the physical world as possible.
- A block or signal that is constant is magenta. They remain unchanged through simulation.
- A discrete block or signal that updates at the lowest fixed interval is red. Signals with discrete sample time update at a fixed interval. If the model contains components with different fixed sample times, each discrete sample time has a different color.
- Multirate subsystems, which contain a mix of discrete and continuous signals, are yellow.

## Trace a Signal

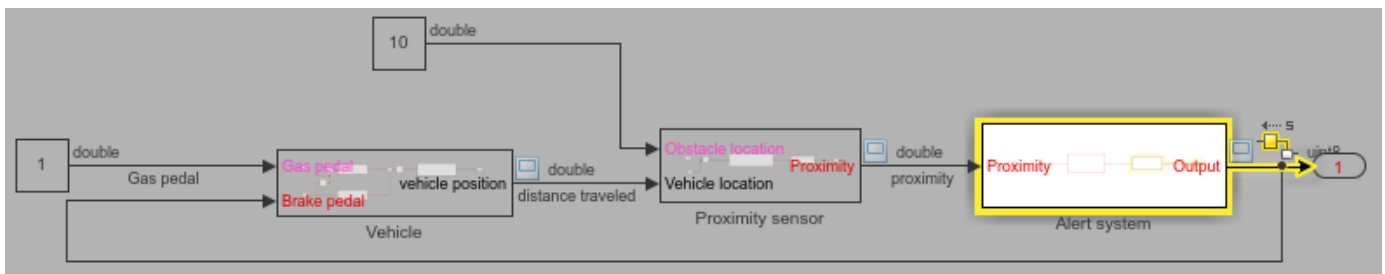
This model has a constant input and a discrete output. To determine where the sampling scheme changes, trace the output signal through blocks.

1

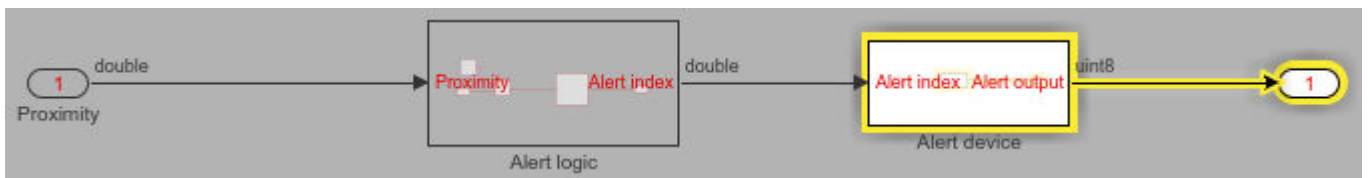
To open the Model Browser, click the **Hide/Show Model Browser** button .

2 To highlight the output signal, select the signal and, in the **Signal** tab, click the **Trace to Source** button .

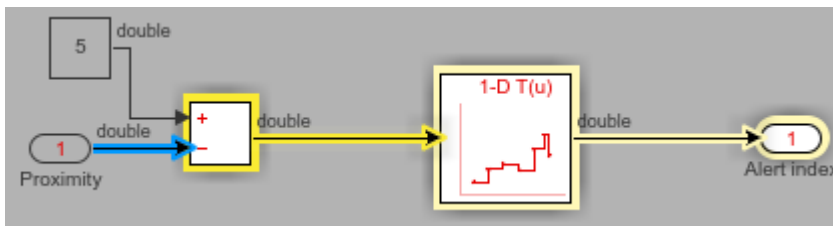
The editor is now in highlight mode. Click the editor to continue.



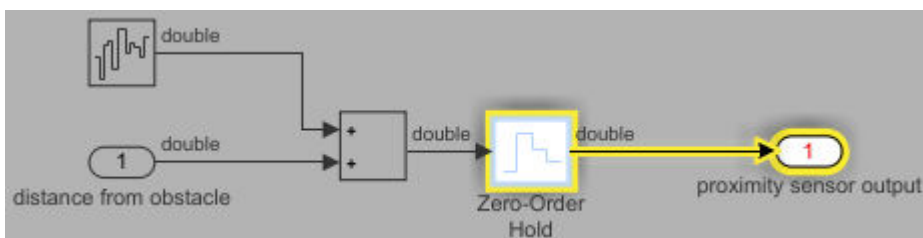
3 To continue tracing the signal to its source, press the left arrow key.



4 Keep tracing the signal to its source until you reach the Alert logic subsystem. You see that the Subtract block has two inputs. Choose the signal path from the Inport by pressing the down arrow key.



5 To find the source of the discretization, keep pressing the left arrow and note the colors of port names that reflect the sample time.



The Zero-Order Hold block in the Sensor model subsystem converts the signal from continuous to discrete.

## **See Also**

### **Related Examples**

- “Create a Simple Model” on page 3-2
- “Model-Based Design with Simulink” on page 1-3

