

Simulink®

Getting Started Guide



MATLAB® & SIMULINK®

R2018b



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.
3 Apple Hill Drive
Natick, MA 01760-2098

Simulink® Getting Started Guide

© COPYRIGHT 1990–2018 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 for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

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)

Introduction

1

Simulink Product Description	1-2
Key Features	1-2
Model-Based Design with Simulink	1-3
Example Model-Based Design Workflow in Simulink	1-5
System Definition and Layout	1-8
Determine Modeling Objectives	1-9
Identify System Components and Interfaces	1-9
Model and Validate a System	1-16
Model the Components	1-16
Validate Components Using Simulation	1-21
Validate the Model	1-24
Design a System in Simulink	1-29
Identify Designed Components and Design Goals	1-29
Analyze System Behavior Using Simulation	1-30
Design Components and Verify Design	1-34
Documentation and Resources	1-40
Simulink Online Help	1-40
Simulink Examples	1-40
Website Resources	1-42

Modeling in Simulink

2

Simulink Block Diagrams	2-2
--------------------------------------	------------

Simple Simulink Model

3

Create a Simple Model	3-2
Open New Model	3-3
Open Simulink Library Browser	3-5
Add Blocks to a Model	3-7
Connect Blocks	3-9
Add Signal Viewer	3-12
Run Simulation	3-12
Refine Model	3-14

Navigate a Simulink Model

4

Navigate Model	4-2
Navigate Through Model Hierarchy	4-2
View Signal Attributes	4-4
Trace a Signal	4-7

Introduction

- “Simulink Product Description” on page 1-2
- “Model-Based Design with Simulink” on page 1-3
- “System Definition and Layout” on page 1-8
- “Model and Validate a System” on page 1-16
- “Design a System in Simulink” on page 1-29
- “Documentation and Resources” on page 1-40

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.

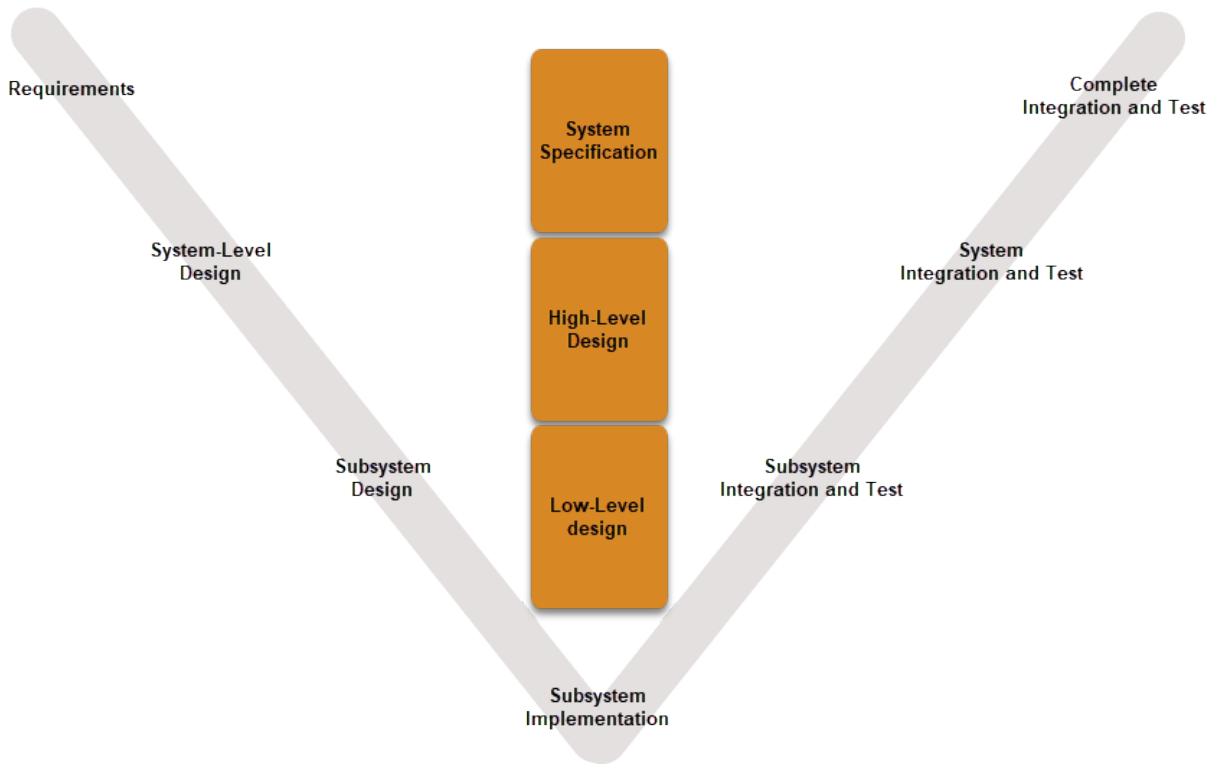
Modeling and simulation are especially valuable for testing conditions that are difficult to reproduce with hardware prototypes alone. This is especially true in the early phase of the design process when hardware is not yet available. Iterating between modeling and simulation can improve the quality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



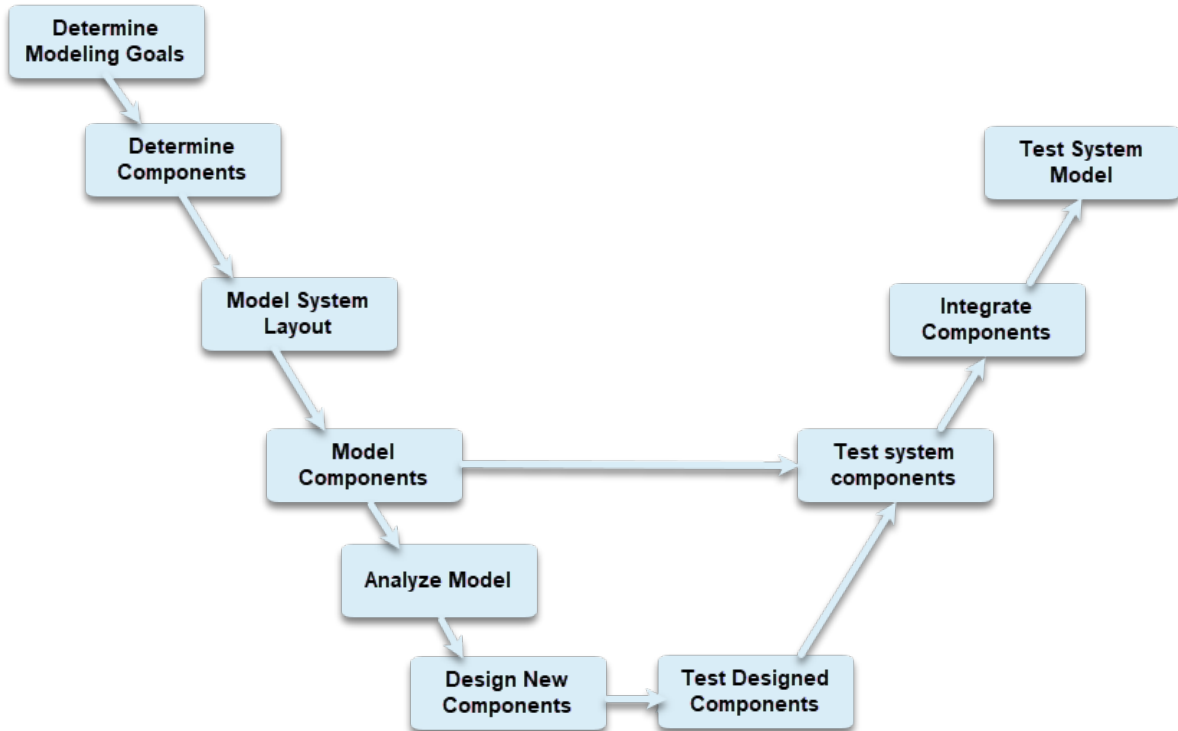
Modeling is a way to create a virtual representation of a real-world system. You can simulate this virtual representation under a wide range of conditions to see how it behaves.

Modeling and simulation are especially 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.

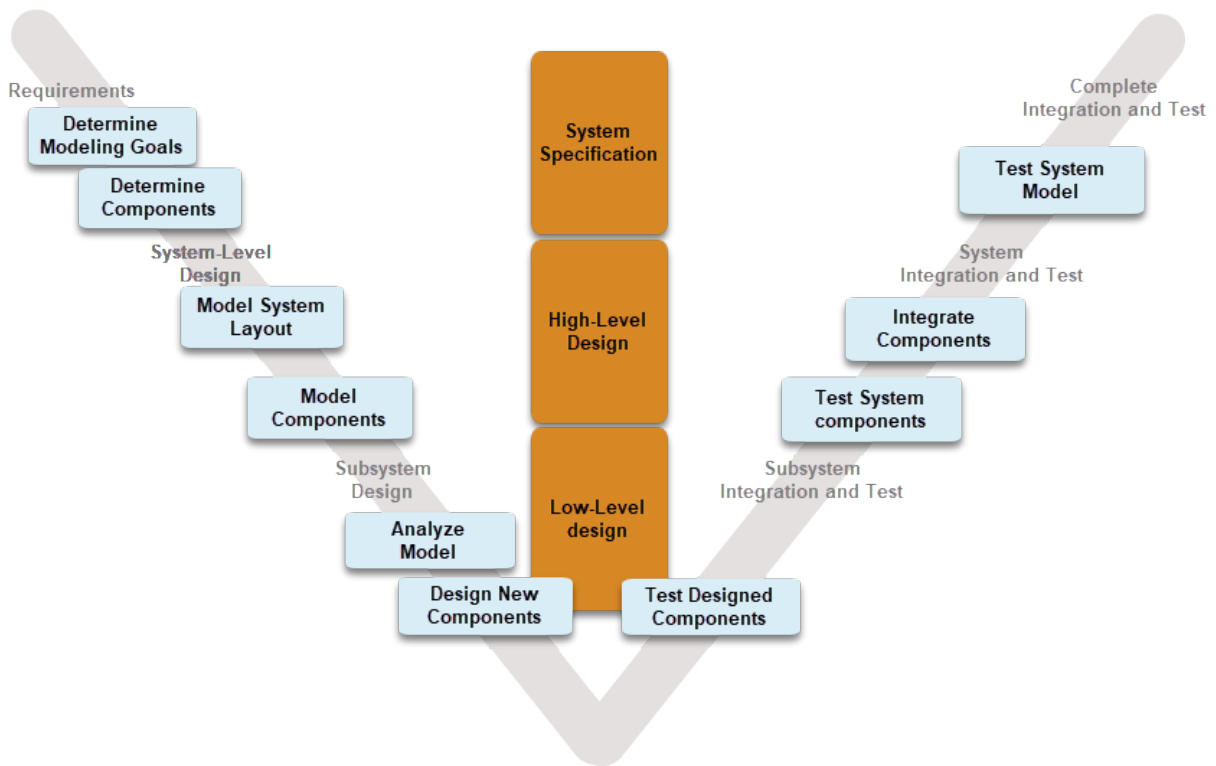
You can automatically generate code from a model and, when software and hardware implementation requirements are included, create test benches for system verification. Code generation saves time and prevents the introduction of manually coded errors.

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.



For an example workflow, see:

- “System Definition and Layout” on page 1-8 — Identify modeling goals, determine components, model system layout
- “Model and Validate a System” on page 1-16 — Model and test components, integrate components, test system
- “Design a System in Simulink” on page 1-29 — 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 to generate code and use the design with a real, physical system.

See Also

Related Examples

- “System Definition and Layout” on page 1-8
- “Model and Validate a System” on page 1-16
- “Design a System in Simulink” on page 1-29
- “Organize Large Modeling Projects”

External Websites

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

System Definition and Layout

In this section...

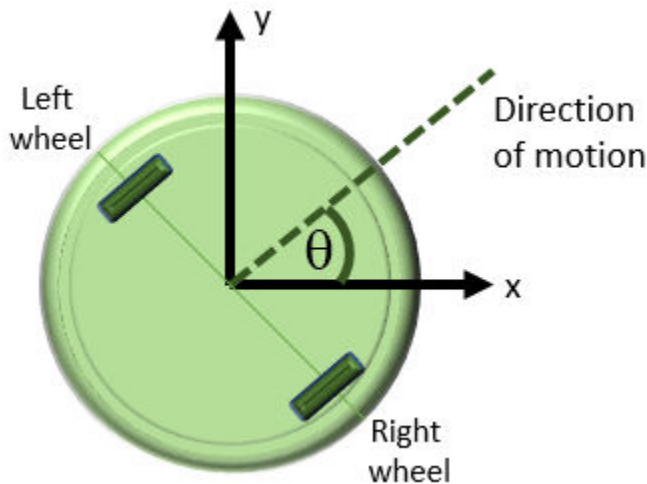
“Determine Modeling Objectives” on page 1-9

“Identify System Components and Interfaces” on page 1-9

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 is a flat robot that can move or rotate with the help of two wheels, similar to a home vacuuming robot. This tutorial 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, linear and rotational motion components can be modeled separately for this introductory tutorial.

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. For example, 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 for the goal of designing sensor parameters so that the robot stops in time when it detects an obstacle on its path. To achieve this goal, the model must enable you to:

- Determine how quickly the robot stops when the motors stop
- Provide a series of commands for linear and rotational motion so that it can move over 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 to understand, build, communicate, and test the system. This becomes more important when parts of the system are to be implemented in the process.
- What are the inputs and outputs for each component? Draw a picture showing the connections between components. This picture leads to signal flow within the model, and, in addition to the source and sink of each signal, it helps determine if all necessary components exist.

- What level of detail is necessary? Include major 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 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 gradually increase complexity where required.

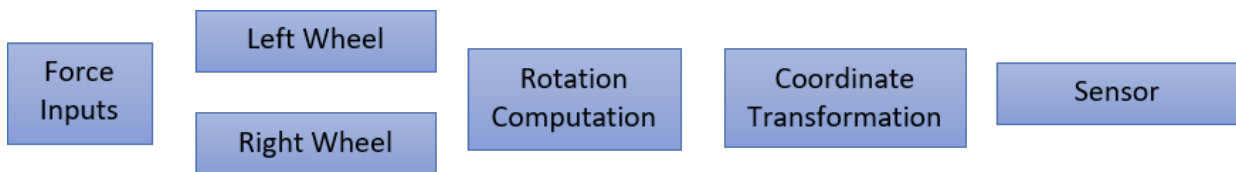
In addition, it is often beneficial to consider the following:

- 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:

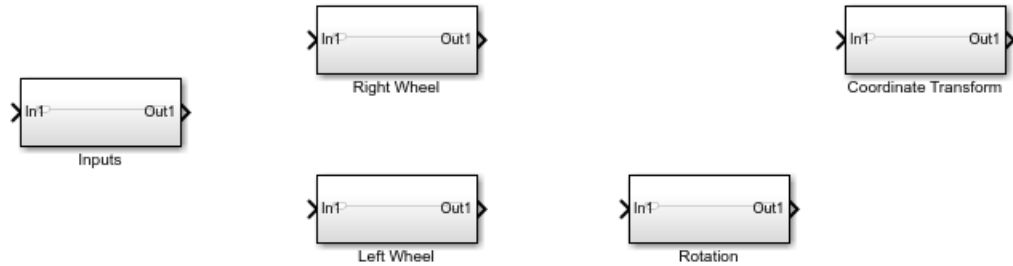
- Linear motion characteristics
- Rotational motion characteristics
- Transformations to determine the location of the system in two dimensions
- A sensor to measure the distance of the robot from an obstacle



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: “Open New Model” on page 3-3.
- 2 From the **Display** menu, clear the **Hide Automatic Names** check box.
- 3 Open the Library Browser: “Open Simulink Library Browser” on page 3-5
- 4 Add Subsystem blocks. Drag five Subsystem blocks from the Ports & Subsystems library to the new model.

Arrange and rename the Subsystem blocks as shown. Double-click a block name and type the new name.



Define Interfaces Between Components

Identify input and output connections (for example, signal lines) between subsystems. Input and output values change dynamically during a simulation. Lines connecting blocks represent data transfer. The table below shows the inputs and outputs for each component.

Block	Input	Output	Notes
Inputs	None	Force to right wheel Force to left wheel	
Right wheel	Force to right wheel	Right wheel velocity	Directional, negative means reverse direction
Left wheel	Force to left wheel	Left wheel velocity	Directional, negative means reverse direction
Rotation computation	Velocity difference between right and left wheels	Rotational angle	Measured counterclockwise
Coordinate transformation	Normal speed Rotational angle	Velocity in X Velocity in Y	

Block	Input	Output	Notes
Sensor	X coordinate Y coordinate	None	No block necessary for modeling. Sensor dynamics is part of the design task.

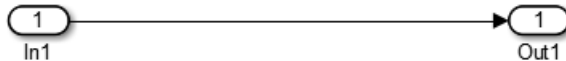
From the table, you can see that some block inputs do not exactly match block outputs. Therefore, in addition to the dynamics of the individual components, the model must compute the following:

- 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:

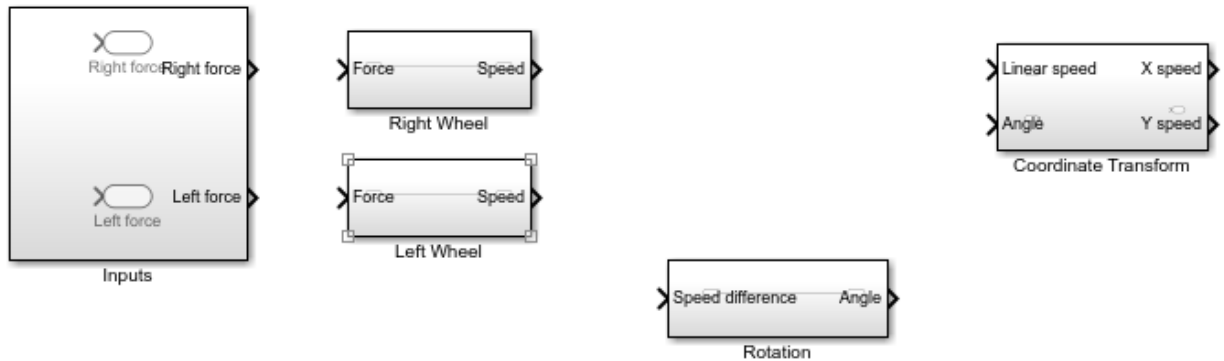
- 1 Add necessary input and output ports to each subsystem. Double-click a Subsystem block.



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.

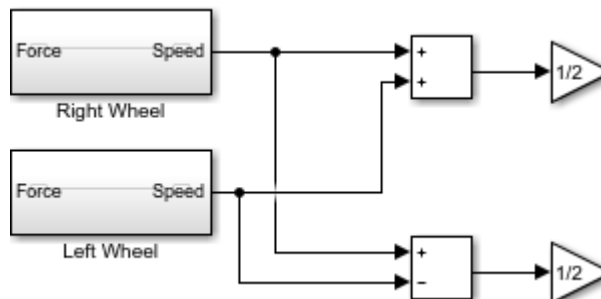
Each Inport block creates an input port on the Subsystem block, and each Outport block creates an output port. The model reflects the names of these blocks as the input/output port names. Add more blocks for additional input and output signals. On the Simulink Editor toolbar, click the **Up to Parent**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

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

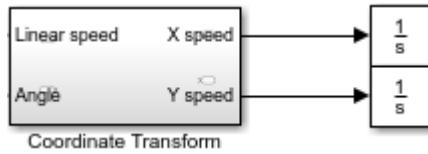


When copying an Inport block to create a new one, you must use the **Paste** option.

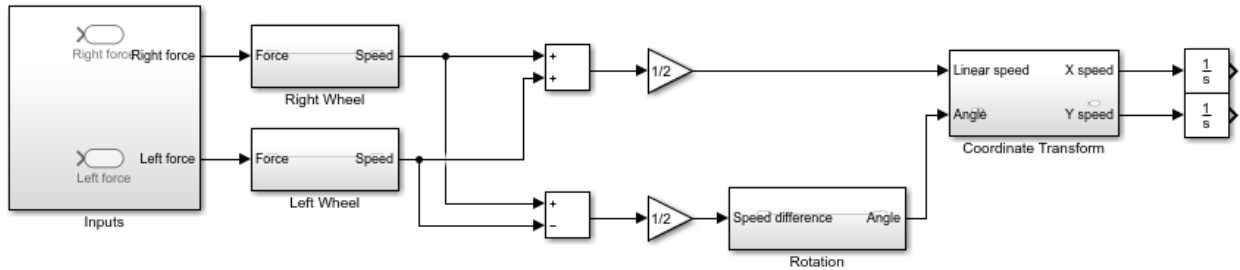
- 2 Compute required inputs from left wheel and right wheel velocities shown.
 - a Add an Add block from the Math Operations library and connect the outputs of the two-wheel components. Click the output port of the source block and then click the input port of the destination block. Add a Gain block and set the parameter to $1/2$. Compute the Linear speed input to the Coordinate Transform subsystem, connect the output of the Add block to this Gain block.
 - b Add a Subtract block from the Math Operations library and connect the outputs of the two-wheel components. Add a Gain block and set the parameter to $1/2$. Compute the Speed difference input to the Rotation subsystem, connect the output of the Subtract block to this Gain block.



- 3 Compute 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 to the Integrator blocks as θ .



4 Complete the connections for the system as shown.



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.

Parameter	Block	Symbol	Value/Unit	Notes
Mass	Left/right wheel	m	2.5 kg	Variable
Rolling resistance	Left/right wheel	k_drag	30 Ns ² /m	Variable
Robot radius	Rotation computation	r	0.15 m	Variable
Initial angle	Rotation computation	None.	0	Fixed
Initial velocities	Left/right wheel	None.	(0,0)	Fixed
Initial coordinates	Integrators	None	(0,0)	Fixed

Simulink uses 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 1-16
- “Design a System in Simulink” on page 1-29

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.

A big-picture view of the whole system layout is useful when modeling individual components. Start by loading the layout model:

```
open_system(fullfile(matlabroot,...  
'help', 'toolbox', 'simulink', 'examples', 'system_layout'))
```

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. An example would be the heat a TV set dissipates. You can use 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. The switching logic of a thermostat is an example. You can model most functional relationships as Simulink blocks and subsystems.

This tutorial models physical and functional components with explicit input/output relationships:

- 1 Use the system equations to create a Simulink model.
- 2 Add Simulink blocks in the Simulink Editor. Blocks represent coefficients and variables from the equations. Connect blocks.
- 3 Build the model for each component separately. The most effective way to build a model of a system is to 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 and the values that do not change unless you change them?
- What are the variables for each component and the values that 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 second-order differential equation provides 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 following difference equation gives 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 — This force F acts in the direction of velocity change, and is an input to the wheel subsystem.
- Drag force — This force F_{drag} acts against the direction of velocity change, and is a function of the velocity itself:

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

The acceleration is proportional to the sum of these forces:

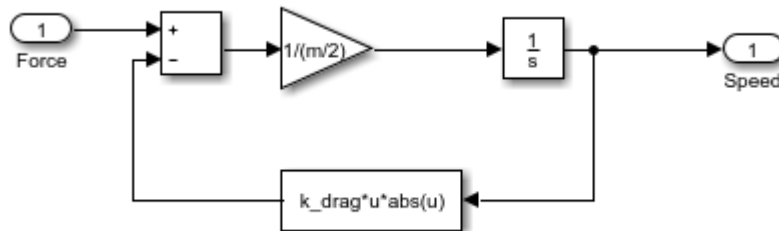
$$\begin{aligned}(m/2)\dot{V} &= F - F_{drag} \\ (m/2)\dot{V} &= F - k_{drag}V|V| \\ \dot{V} &= \frac{F - k_{drag}V|V|}{(m/2)}\end{aligned}$$


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 layout model, double-click the Right Wheel subsystem to display the empty subsystem. Delete the connection between the Inport and the Output blocks.
- 2** Model velocity and acceleration. Add an Integrator block. Leave the initial condition as \emptyset . The output of this block is the velocity, V , and the input is the acceleration, $Vdot$.
- 3** Model the drag force. Add an Fcn block from the User-Defined Functions library. Set the expression to $k_drag*u*abs(u)$. You can resize the block to see the expression on its icon. The Fcn block provides a quick way to type simple mathematical expressions of one input variable, u .

- 4 Subtract the drag force from the motor force with Subtract block, and complete the force-acceleration equation with a Gain block with parameter $1/(2*m)$.
- 5 To reverse the direction of the Fcn block, right-click the block and select **Rotate & Flip > Flip Block**. Make the connections between blocks as shown.



- 6 View the top level of the model: Click the **Up to Parent** button . Make a copy of the subsystem you modeled as the dynamics for both wheels are the same.

Rotational Motion

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


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

Build the Rotation Dynamics model:

- 1 In the layout model, double-click the Rotation subsystem to display the empty subsystem. Delete the connection between the Inport and the Output.
- 2 Model angular speed and angle: Add an Integrator block. Leave the initial condition as θ . The output of this block is the angle, θ , and the input is the angular speed, θ_{dot} .
- 3 Compute angular speed from tangential speed. Add a Gain with parameter $1/(2*r)$.

- 4 Make the connections between blocks as follows.



- 5 View the top level: Click the **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

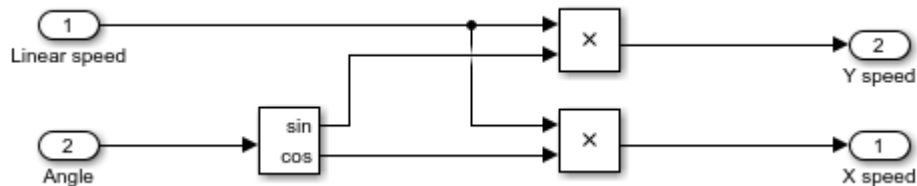
The velocity of the robot in the X and Y coordinates, V_x and V_y , are related to the linear speed, V_n , and the angle as follows:


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

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

Build coordinate transformation model:

- 1 In the 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 Make connections between the blocks as shown.



- 5 View the top level: Click the **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

The model uses these parameters:

Parameter	Symbol	Value/Unit
Mass	m	2.5 kg
Rolling resistance	k_drag	30 Ns ² /m
Robot radius	r	0.15 m

Simulink uses 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 the following behavior:

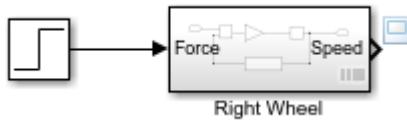
- When a force is applied continuously to a wheel, the velocity increases until it reaches a steady-state velocity.
- When the wheels are turning in opposite directions, the angle increases steadily.


Validate Wheel Component

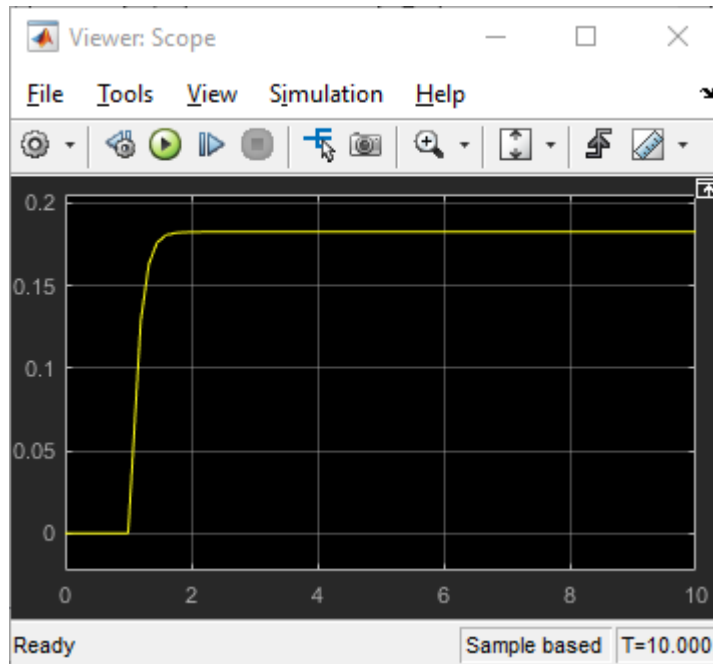
Create and run a test model for the wheel component:

- 1 Create a model. Click  and copy the Right Wheel block into the new model.
- 2 Create a test input in the new model. Add a Step block from the Sources library. Connect it to the input of the Right Wheel block.

- 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. Click .




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. Besides validation, this simulation also gives information on the maximum speed of the wheel with the given force.

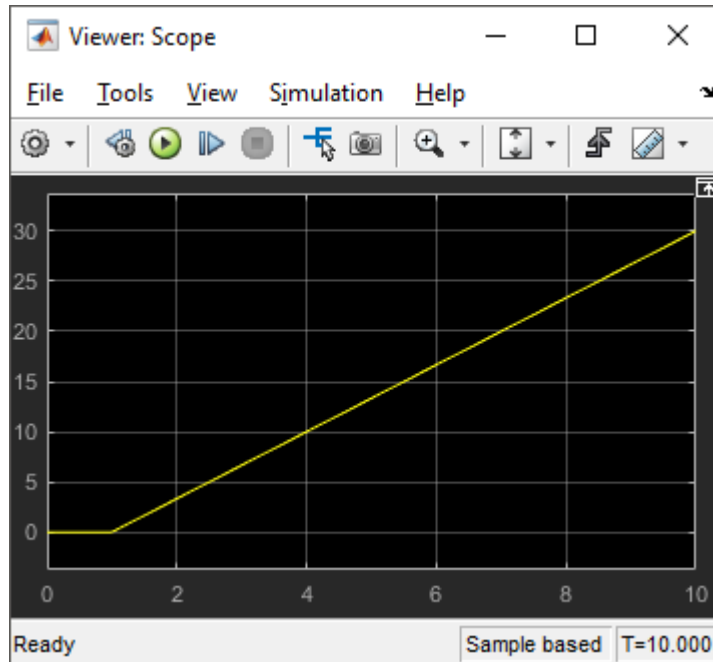
Validate Rotation Component

Create and run a test model for the rotation model:

- 1 Create a 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. 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. Click .



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:

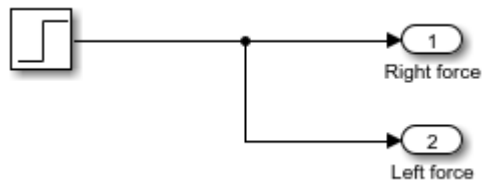
- You can convert the output in radians to degrees. Add a Gain block with a gain of $180/\pi$.
- You can display the degrees output in cycles of 360 degrees. Add a Math Function block with function `mod`.

MATLAB trigonometric functions take inputs in radians.

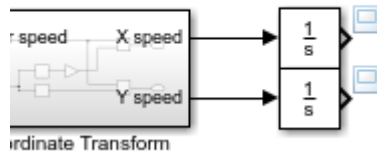
Validate the Model

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

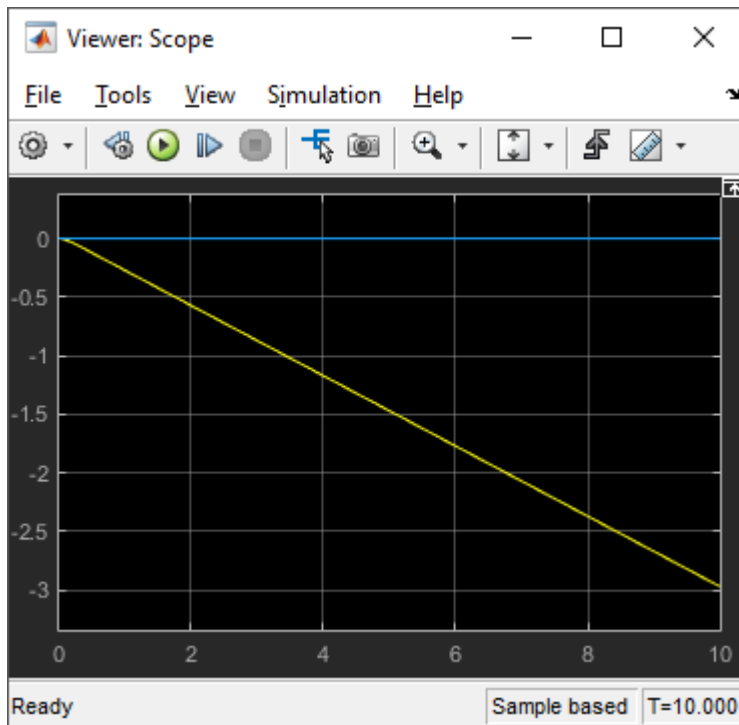
- When the same force is applied to both wheels in the same direction, the vehicle moves in a line.
 - When the same force is applied to both wheels in the same direction, the vehicle moves turns around itself.
- 1 In the layout model, double-click the Inputs subsystem to display the empty subsystem.
 - 2 Create a test input by adding a Step block. Connect it to both Output blocks.



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

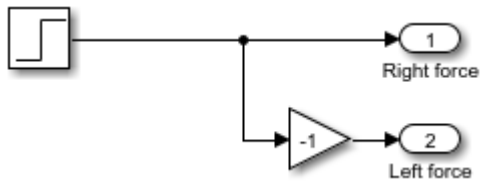


- 4 Run the model.

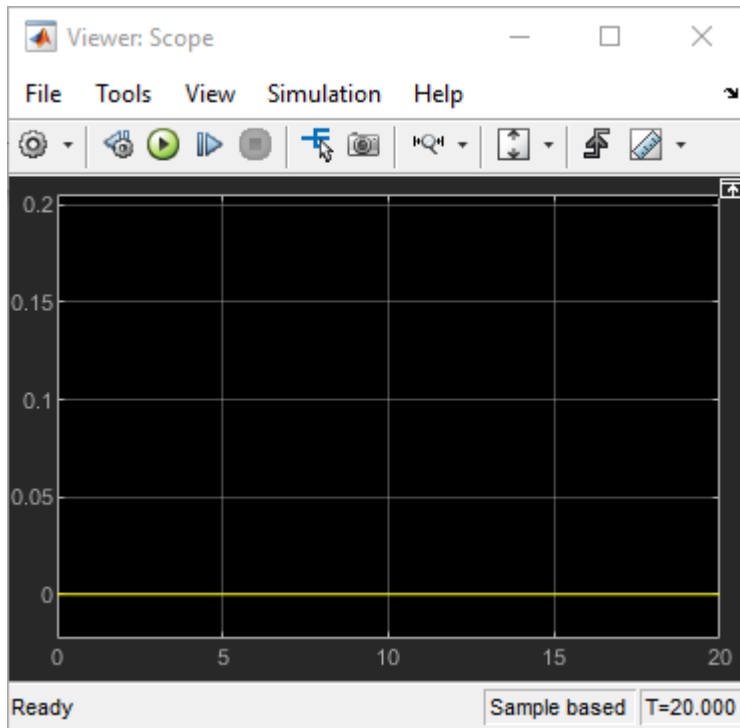


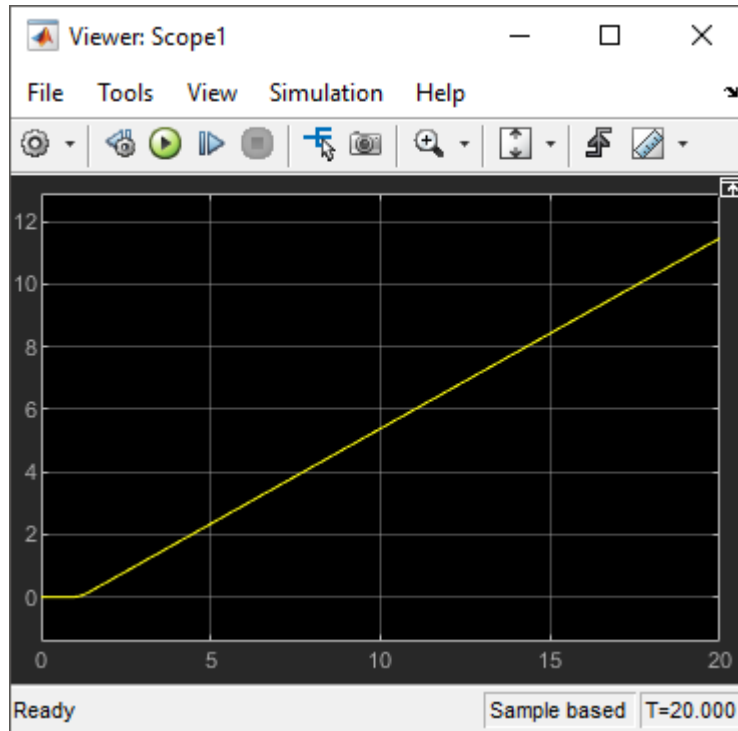
In this figure, the yellow line is the X direction and the blue line is the Y direction. Since the angle is zero and is not changing, the vehicle moves only in the X direction, as expected.

- 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, the drag coefficient is lower?

See Also

Related Examples

- “System Definition and Layout” on page 1-8
- “Design a System in Simulink” on page 1-29

Design a System in Simulink

In this section...

“Identify Designed Components and Design Goals” on page 1-29

“Analyze System Behavior Using Simulation” on page 1-30

“Design Components and Verify Design” on page 1-34

Model-Based Design paradigm is centered around 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.

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 the code at the MATLAB command line.

```
open_system(fullfile(matlabroot,...  
'help', 'toolbox', 'simulink', 'examples', 'system_model'))
```

This tutorial analyzes this system and adds functionality to it.

Identify Designed Components and Design Goals

Proper specification of the objective is a critical first step to the design task. Even with a simple systems, there could be multiple, and even competing design goals. Consider these 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 a 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 an 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, and generates an alert in time for the robot to come to a stop.

Analyze System Behavior Using Simulation

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

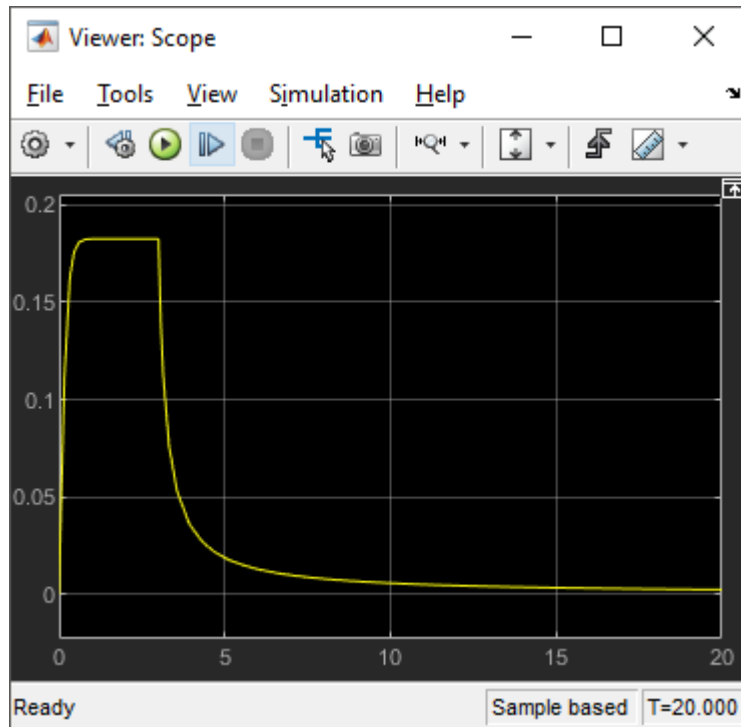
- How far the robot can travel at the top speed if power to the wheels is cut
- The robot's top speed

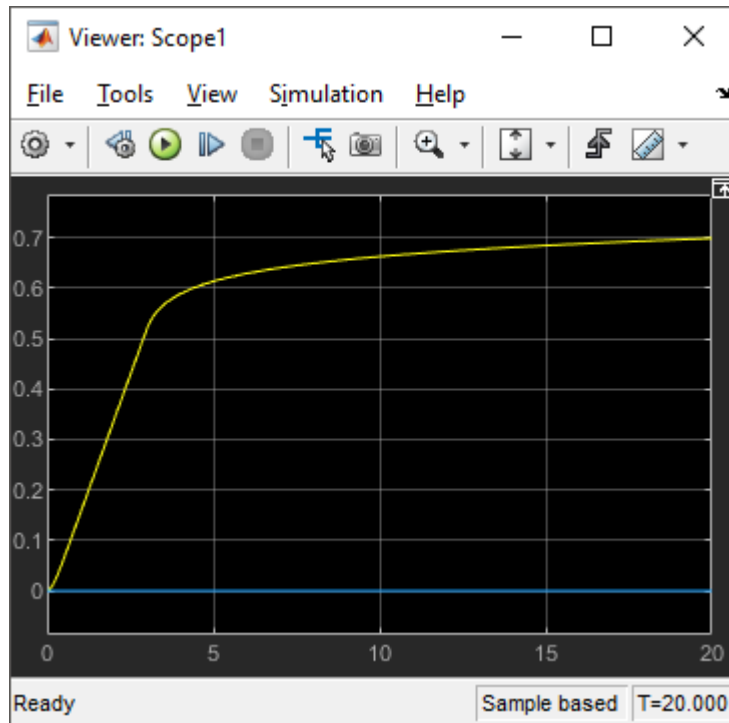
Run the model with a force input that starts motion, waits until the robot reaches a steady velocity, and then sets the force to zero:

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

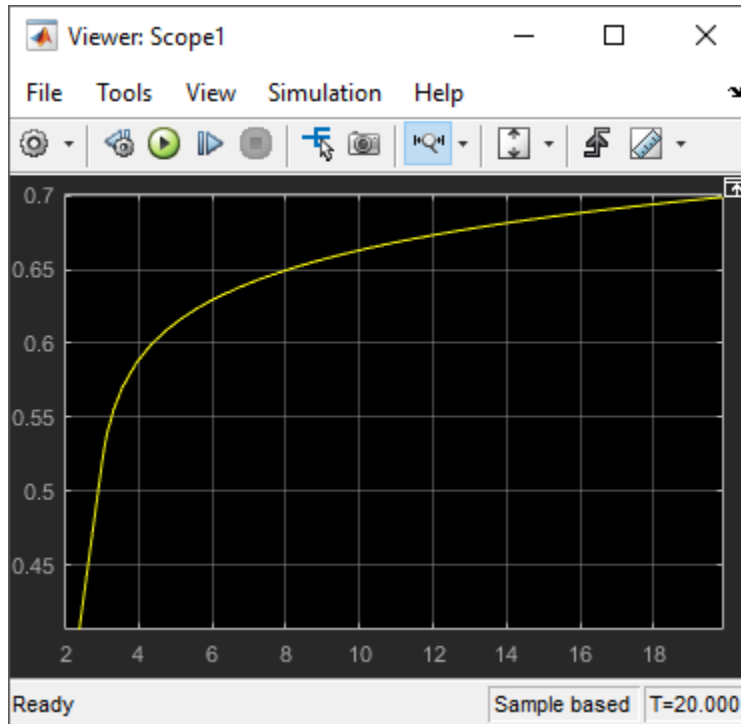
These parameters are designed to ensure that the top speed is reached. You can change parameters to see their effect.

- 4 Run the model for 20 sec.






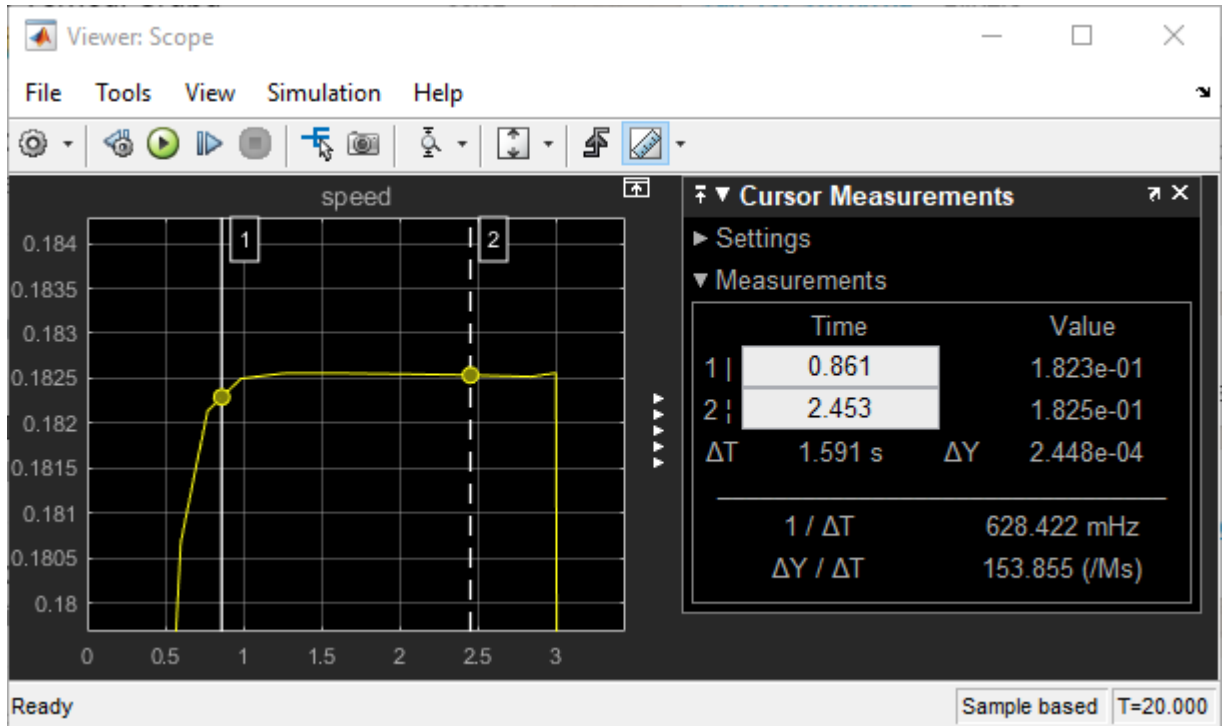
The first scope shows that the speed rapidly starts decreasing when the power is cut at time 3, and then asymptotically approaches zero but does not quite reach it. This is a limitation of modeling — the dynamics at low speeds without external force may require a more complex representation. For the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



At time 3, the position of the robot is at about 0.55 m, and when the simulation ends, it 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 Cursor Measurements button .
- 3 Set the second cursor to the region where the line is horizontal.



The **Value** column in Cursor Measurements indicate that the top speed of the robot is 0.183 m/s. Divide 0.05 by this speed to obtain the time it takes the robot to travel 0.05 m — 0.27 s.

Design Components and Verify Design

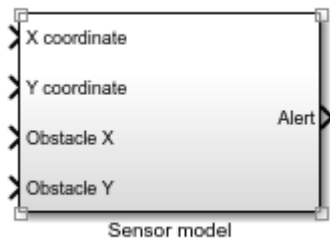
Sensor design consists of these components:

- Measurement of the distance between the robot and the obstacle — This example assumes that the measurement is perfect.
- The interval at which the sensor system measures the distance: To keep the measurement error below 0.05 m, this interval should 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, but the actual alert distance must also take the measurement error, 0.05, into account.

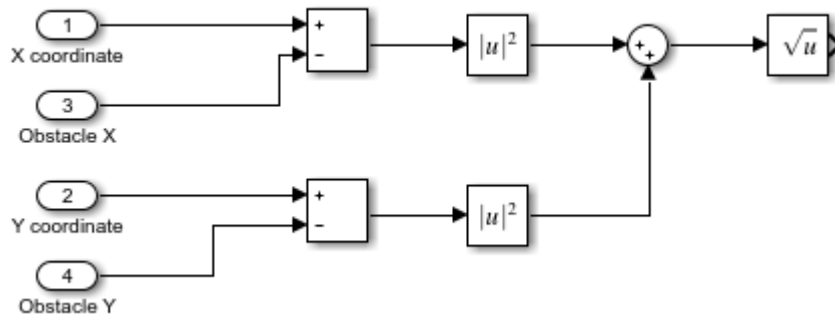
Add Designed Component

Build the sensor:

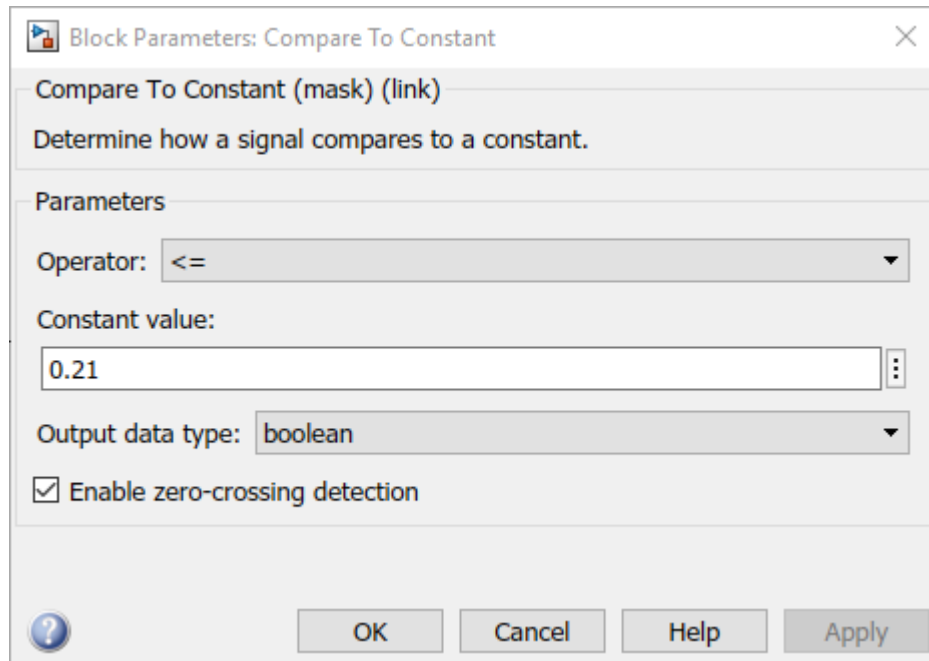
- 1 Create a subsystem with the ports as shown.



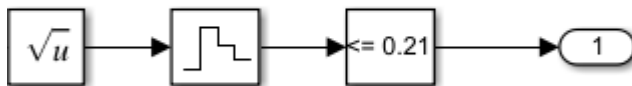
- 2 Construct the distance measurement. In the sensor model block, use Subtract, Math Function with $|u|^2$ function, Sum, and Sqrt blocks as shown. Note the reordering of the input ports.



- 3 Model sampling. Add a Zero-Order Hold block from the Discrete library to the subsystem and set its **Sample Time** parameter to 0.25.
- 4 Model the alert logic. Use the Compare to Constant from Math Operations and set its parameters:
 - **Operator:** \leq
 - **Constant Value:** 0.21



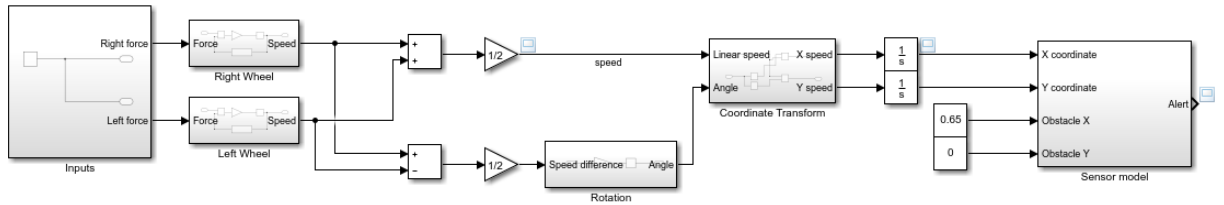
This logical block sets its output to 1 when its input is less than 0.21.



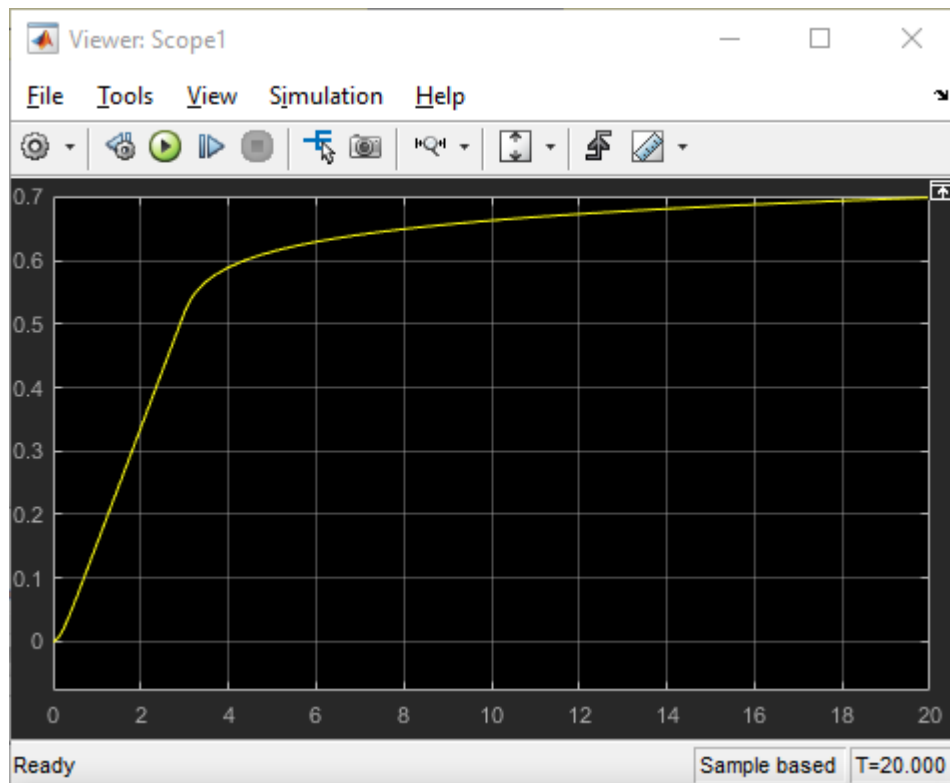
Verify Design

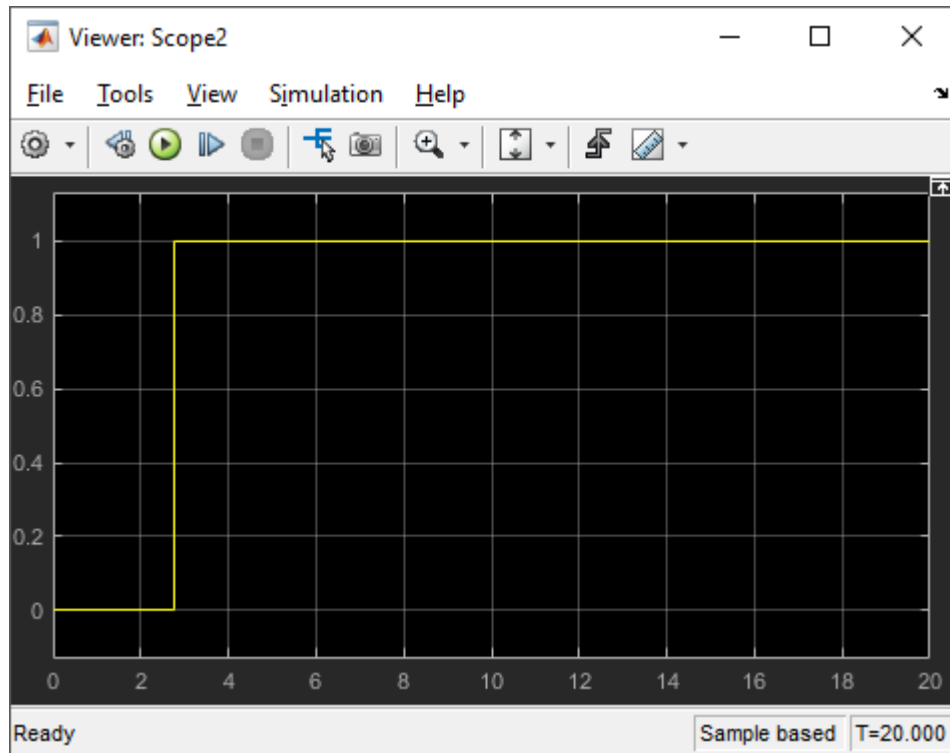
Test the design with an obstacle location of $X=0.65$, $Y=0$, using Constant blocks as input. This test verifies functionality in the X direction, you can create similar tests for different paths. This model only generates an alert. It does not control the robot.

- 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 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 1-8
- “Model and Validate a System” on page 1-16


Documentation and Resources

In this section...
“Simulink Online Help” on page 1-40
“Simulink Examples” on page 1-40
“Website Resources” on page 1-42

Simulink Online Help

Simulink software provides comprehensive online help describing features, blocks, and functions with detailed procedures for common tasks.

Access online help from **Help** menus and context-sensitive block labels.

- From the Simulink Library Browser toolbar, select the **Help** button .
- From the Simulink Editor menu, select **Help > Simulink > Simulink Help**.
- Right-click a Simulink block, and then select **Help**.
- From the model Configuration Parameters dialog box or a block parameters dialog box, right-click a parameter label, then select **What's This?**

Simulink Examples

Simulink provides example models that illustrate key modeling concepts and Simulink features. To view a list of examples:

- From the Simulink Editor menu, select **Help > Simulink > Examples**.
- From the Help browser, open the Simulink product page, and then click **Examples** at the top right.

Simulink

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.

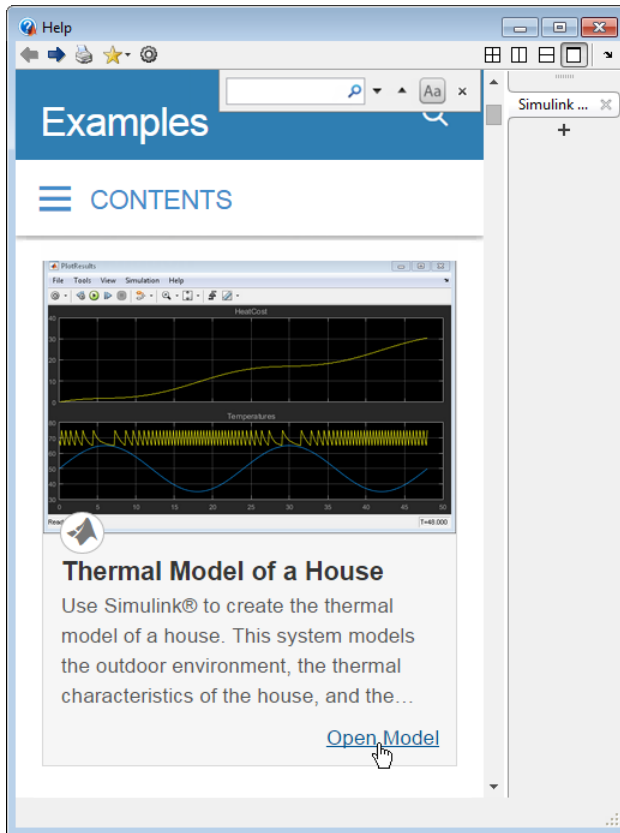
[Examples](#)

[Blocks and Other Reference](#)

[Release Notes](#)

[PDF Documentation](#)

To open the Simulink model for an example, click the **Open Model** button.



Website Resources

You can access additional Simulink resources on the MathWorks website, including a description of capabilities, technical articles, tutorials, and hardware support.

<https://www.mathworks.com/products/simulink>

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:

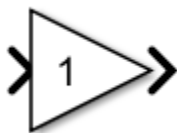


A block can represent a physical component, a small system, or a function; an input/output relationship fully characterizes the block. Consider these examples:

- A faucet fills a bucket: Water goes into the bucket at a certain flow rate, and the bucket gets heavier. Here, a block represents the bucket, with flow rate as its input and its weight as the output.
- You use a megaphone to make your voice heard: 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: Here the cart can be the block, the force you apply is the input and cart position is the output.

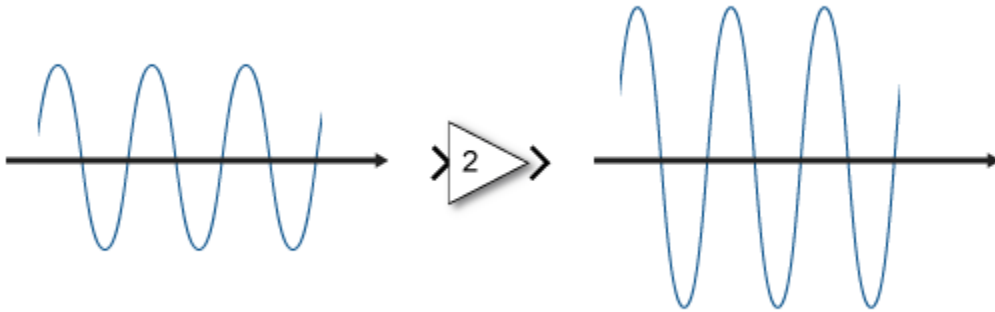
The definition of a block is only complete with its inputs and outputs and this task relates to the goal of the model. For example, the cart velocity may be a natural choice as an output if the modeling goal does not involve its location.

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



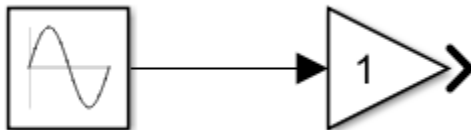
A sound wave goes into the megaphone, as its input, and a louder version of the same wave comes out as its output.

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



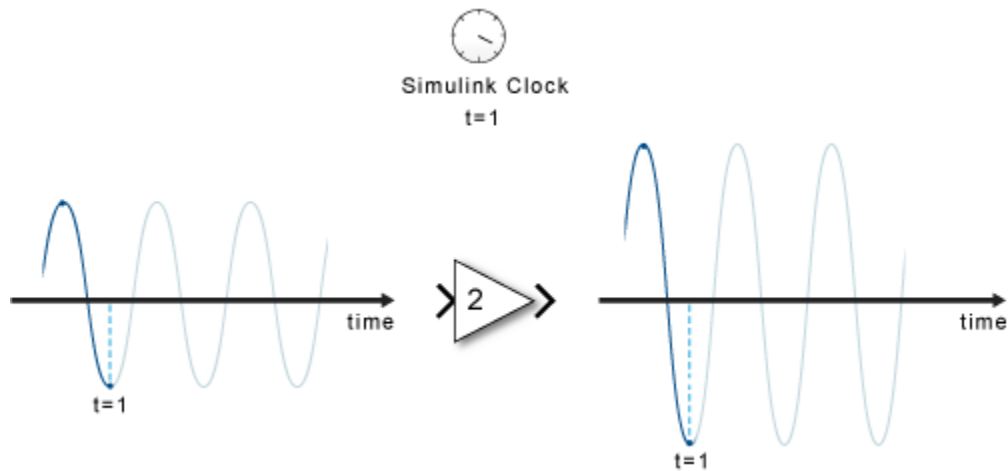
You can connect blocks to other blocks to represent more complex functionality and form systems. An audio player, for example, turns a digital file into sound: A digital representation is read from storage, gets interpreted mathematically, and is turned into sound physically. The software that processes 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 also a block in its own right.

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

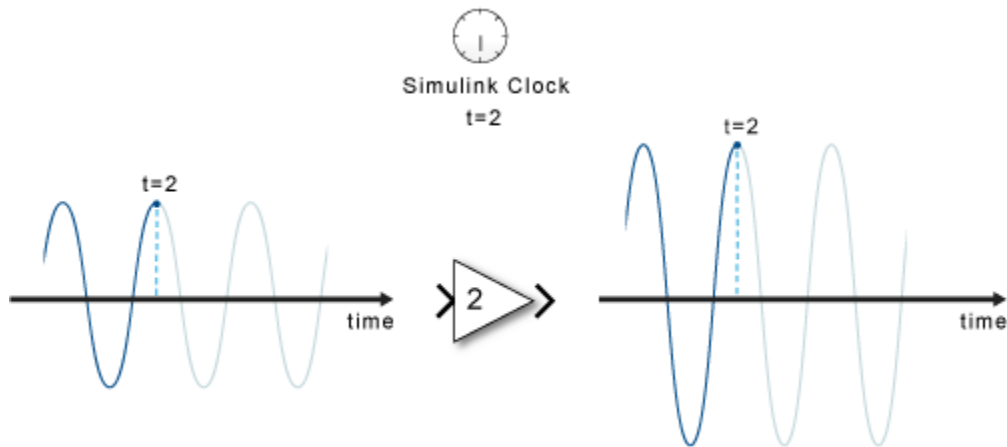


The primary function of Simulink is to simulate behavior of system components over time. In its simplest form, this task involves keeping a clock, determining the order in which the

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 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.

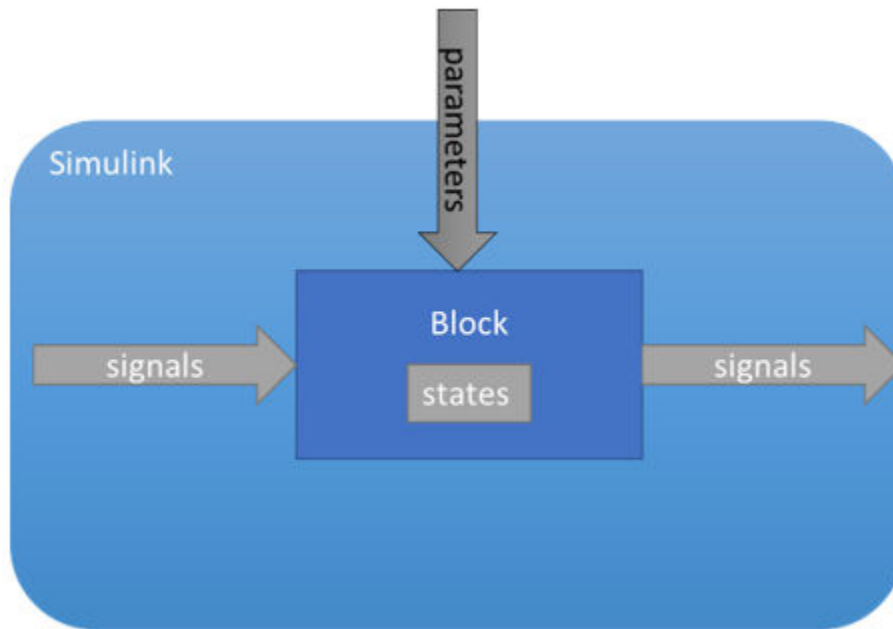


In simulation, time progresses differently from a real clock. Each time step takes as much time as it takes to finish the computations for that time step, whether that time step represents a fraction of a second or a few years.

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, and 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.

Simple Simulink Model

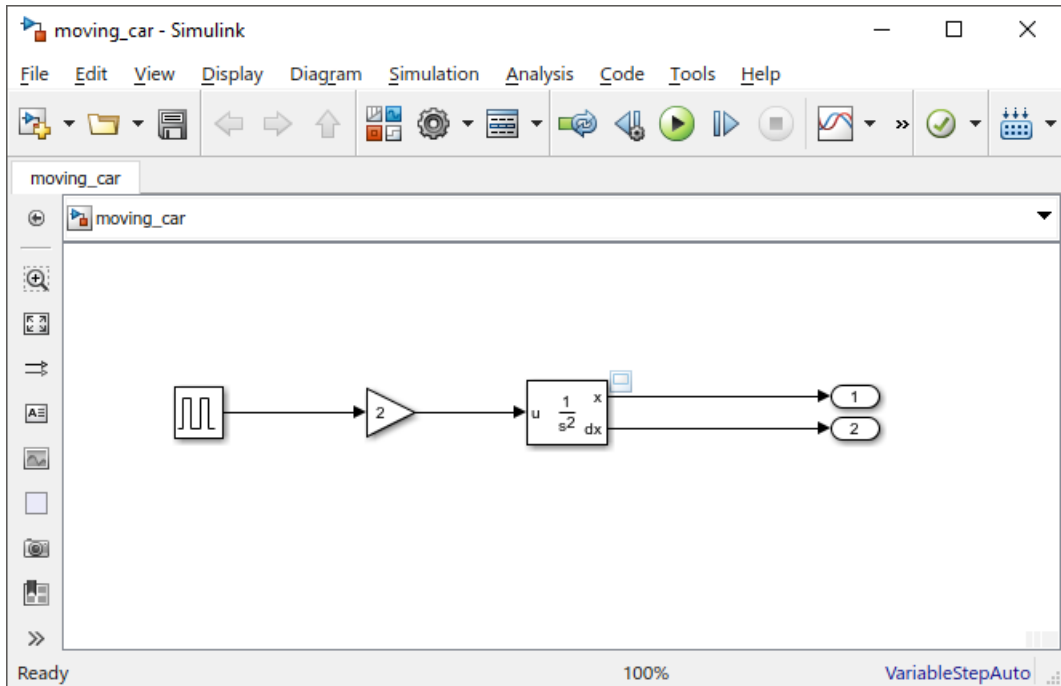
Create a Simple Model

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 3-9
"Add Signal Viewer" on page 3-12
"Run Simulation" on page 3-12
"Refine Model" on page 3-14

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 pedal is pressed. It idles AFTER and comes to a stop. after a brief press of the accelerator pedal.

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 factor	Calculate how pressing the accelerator affects the car acceleration
Integrator, Second-Order	Integrate input signal twice	Obtain position from acceleration
Output	Designate a signal as an output from the model	Designate the position as an output from the model

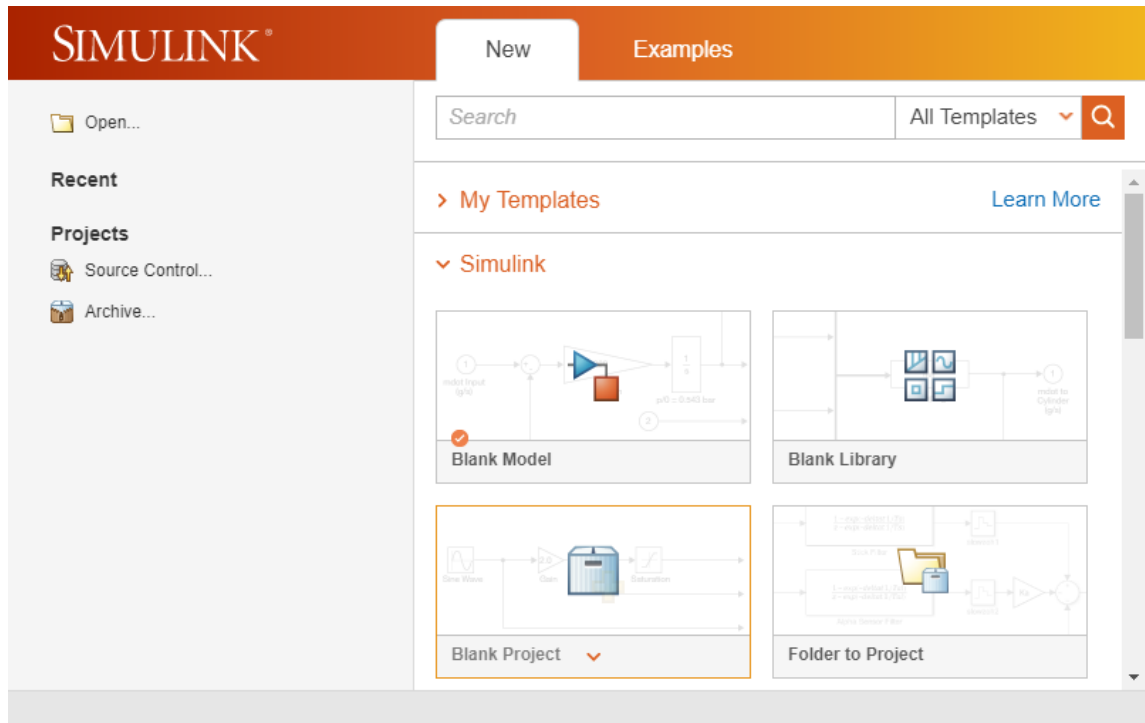


Simulating this model integrates a brief pulse twice to get a ramp. The results display in a Scope window. The input pulse represents a press of the gas pedal — 1 when the pedal is pressed and 0 when it is not. The output ramp is the increasing distance from the starting point.

Open New Model

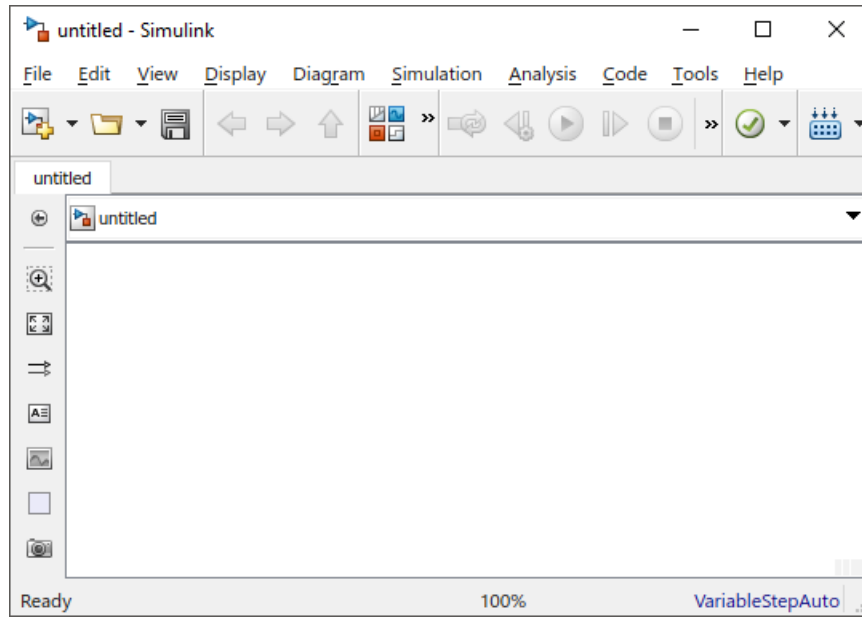
Use the Simulink Editor to build your models.

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



- 2 Click the **Blank Model** template.

The Simulink Editor opens.




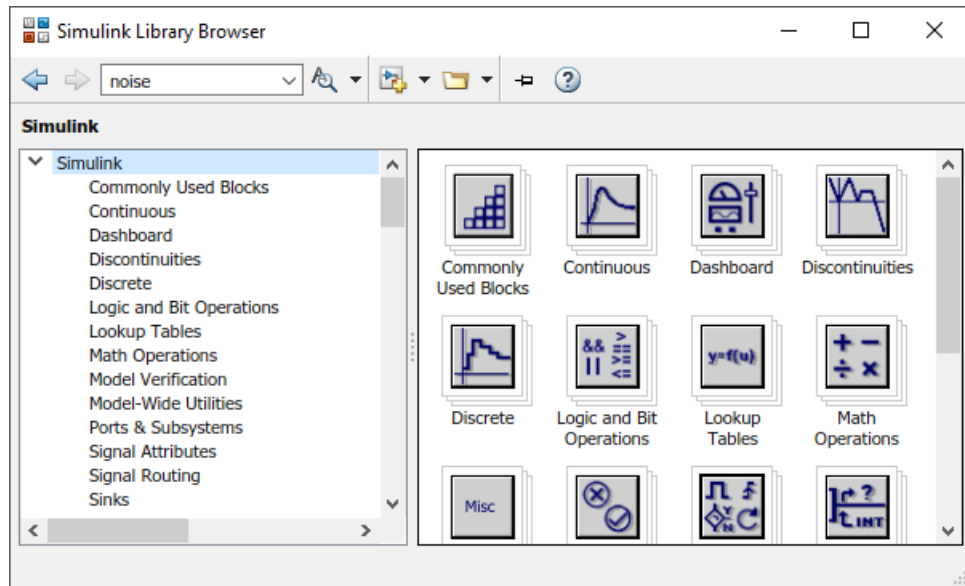
- 3 From the **File** menu, select **Save as**. In the **File name** text box, enter a name for your model, For example, `simple_model`. Click **Save**. The model is saved with the file extension `.slx`.


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

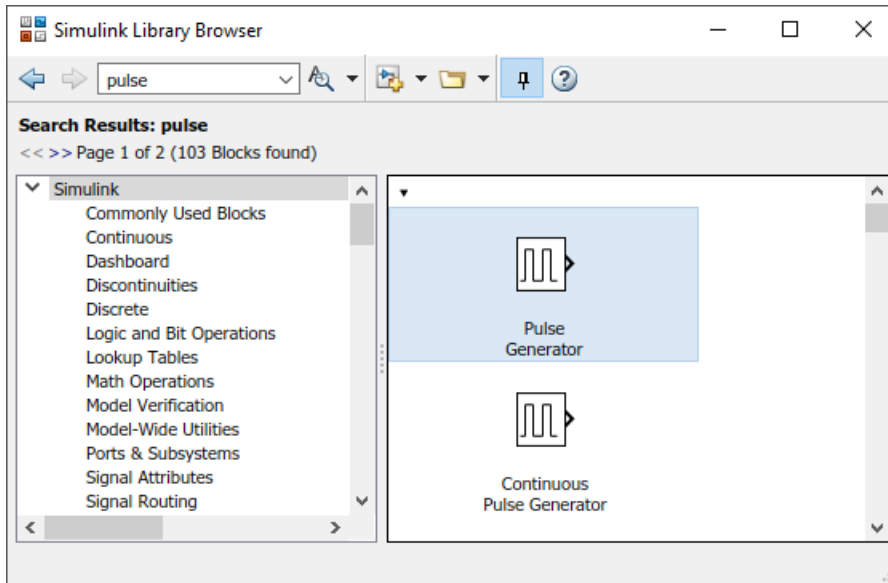
- 1 From the Simulink Editor toolbar, click the **Library Browser** button .



- 2 Set the Library Browser to stay on top of the other desktop windows. On the 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 the Enter key. Simulink searches the libraries for blocks with `pulse` in their name or description, and then displays the blocks.



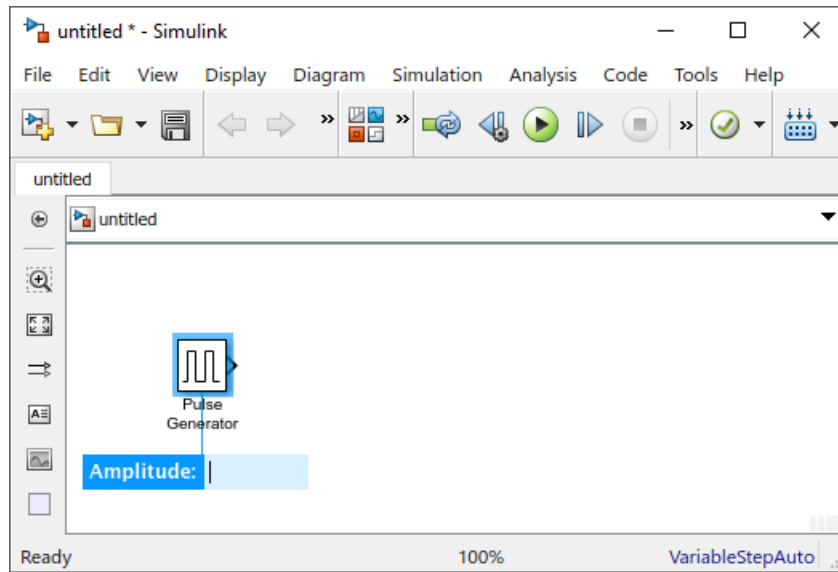
Get detailed information about a block. Right-click a block, and then select **Help for the Pulse Generator block**. The Help browser opens with the reference page for the block.

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

Add Blocks to a Model

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

- 1 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.

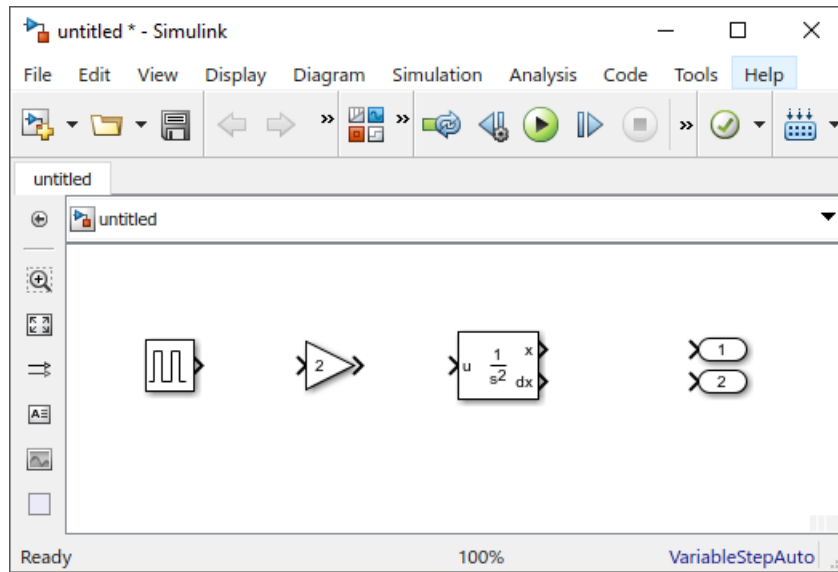
- 2 Add the following blocks to your model using the same approach.

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

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

Your model now has the blocks you need.

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

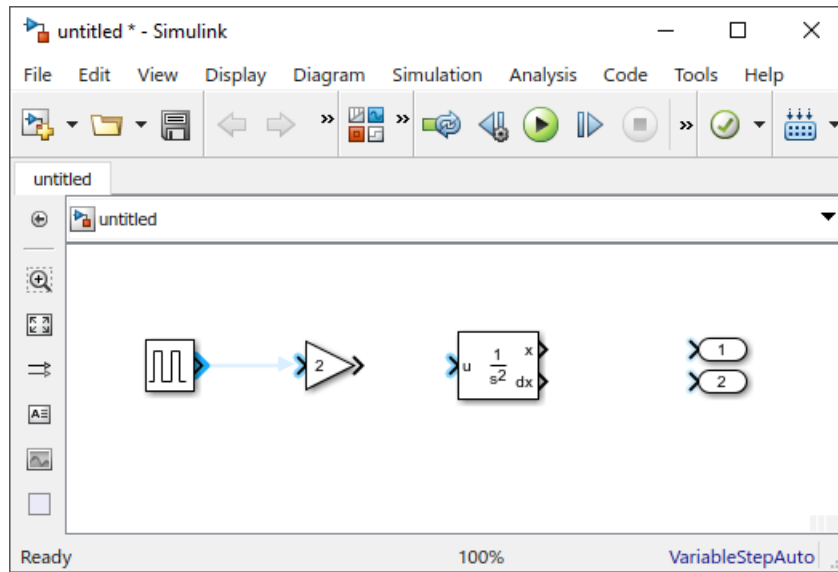


Connect Blocks

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

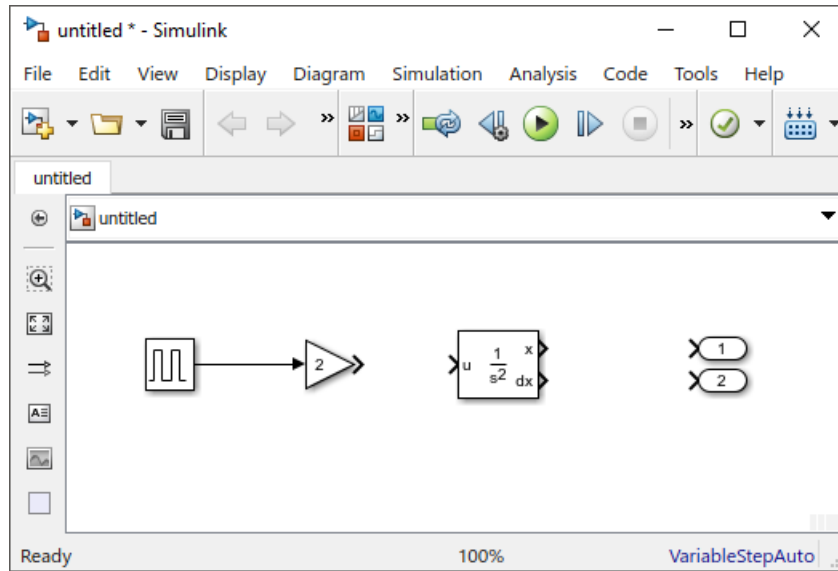
- 1 Click the output port on the right side of the Pulse Generator block.

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

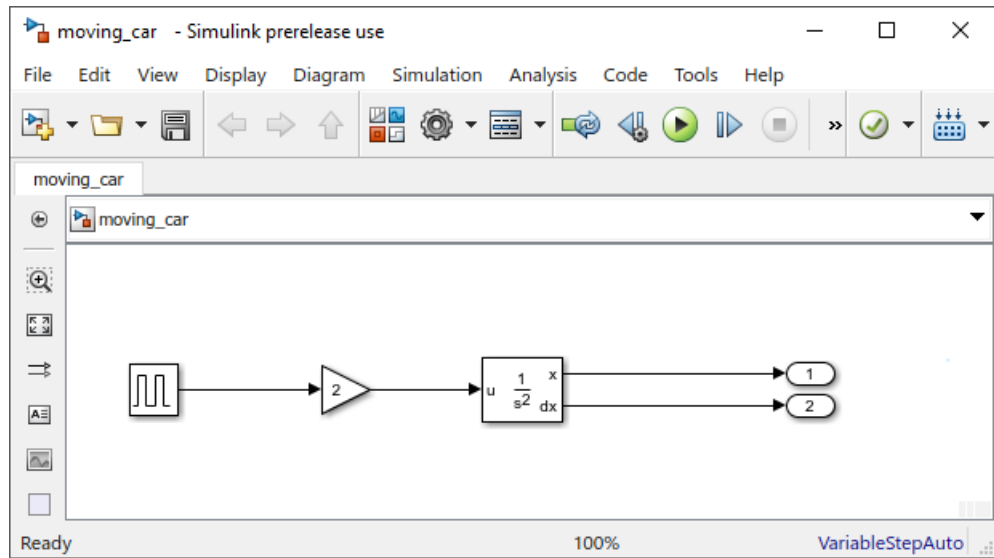


- 2 Click the input port of the Gain block.

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



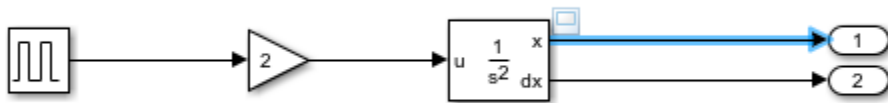
- 3 Connect the output port of the Gain block to the input port on the Integrator, Second Order block.
- 4 Connect the two outputs of the Integrator, Second Order block to the two Output blocks.
- 5 Save your model. Select **File > Save** and provide a name.



Add Signal Viewer

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

Access the context menu by right-clicking the signal. Select **Create & Connect Viewer > Simulink > Scope**. A viewer icon appears on the signal and a scope window opens.

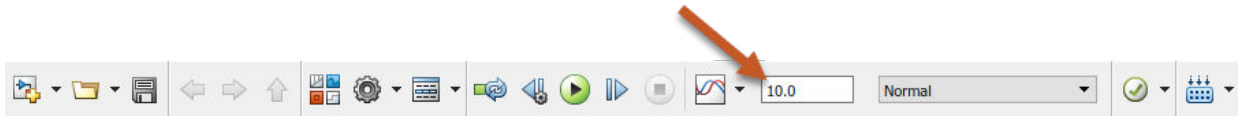


You can open the scope at any time by double-clicking the icon.


Run Simulation

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

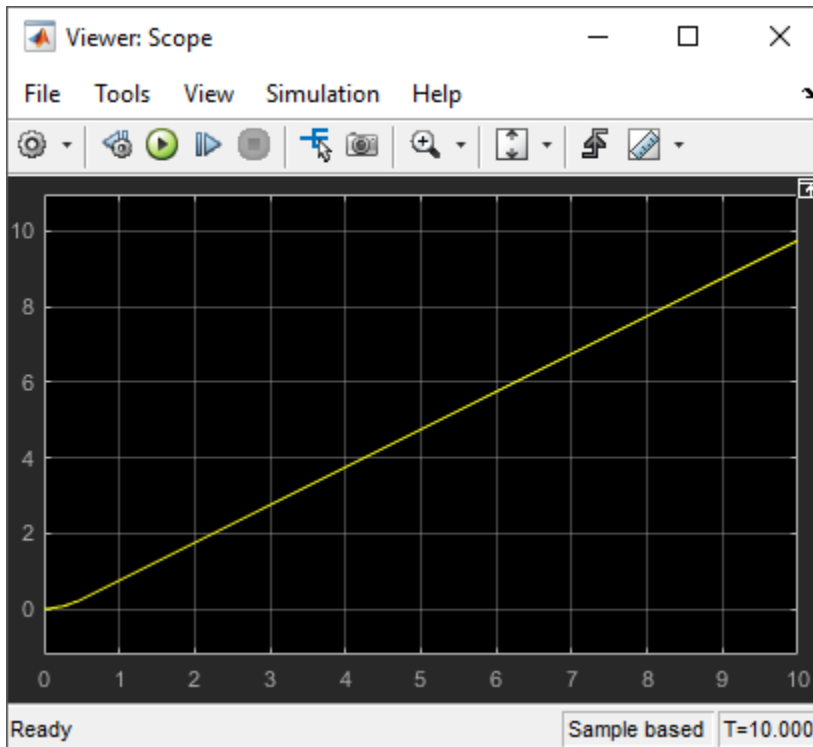
- 1 On the model window, set the simulation stop time by changing the value at the toolbar.



The default stop time of 10.0 is appropriate for this model. This time value has no unit. 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.

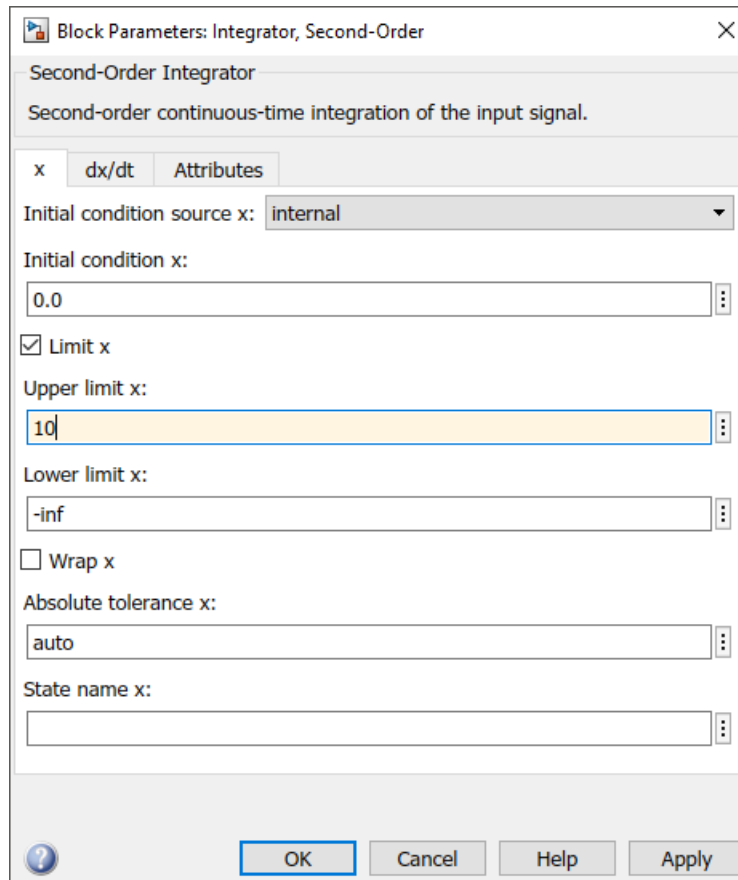
Change Block Parameters

To start, open the `moving_car` model. In the MATLAB Command Window, enter

```
open_system(fullfile(matlabroot,...  
'help', 'toolbox', 'simulink', 'examples', 'moving_car'))
```

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.

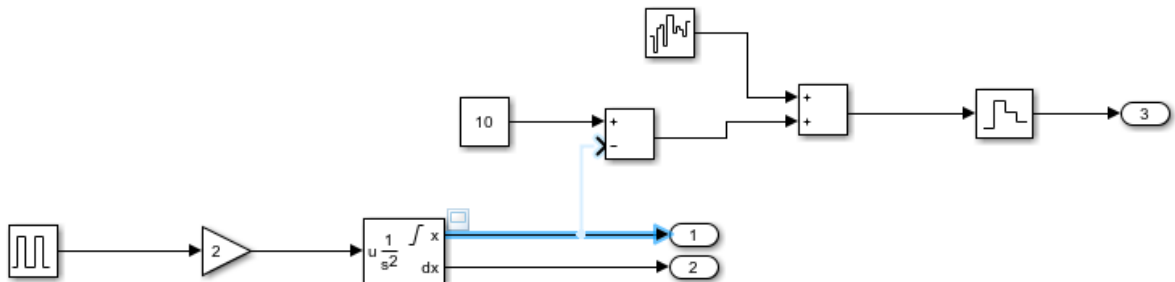
- 3 Click **OK** to apply the changes and close the dialog box.

Add New Blocks and Connections

Add a sensor that measures the distance from the obstacle.

- 1 Modify the model. Extend 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. Also add the Constant block 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 the digital sensor that fires every 0.1 seconds. In Simulink, sampling of a signal at a given interval requires a sample and hold, implemented by a zero-order 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 Outputport to connect to the sensor output. Leave the **Port number** parameter as default.
- 2 Connect the new blocks. Note that 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.



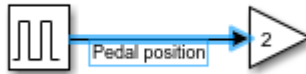
Annotate signals

Add signal names to the model to make it easier to understand.

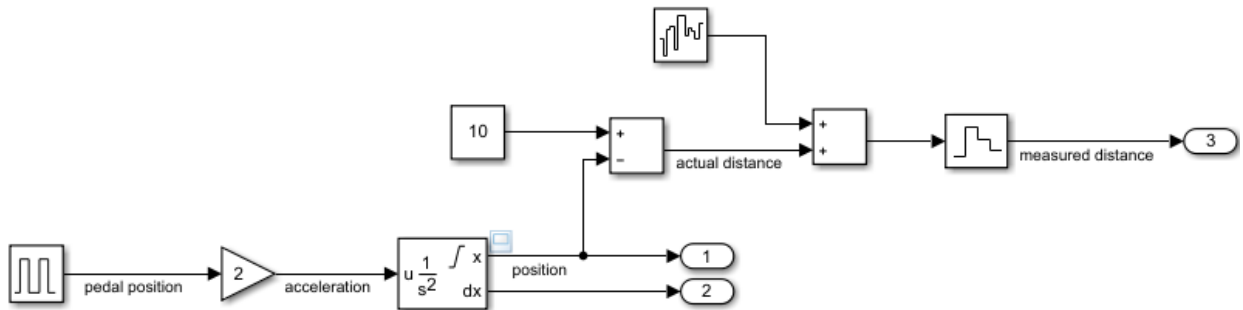
- 1 Double-click the signal. An editable textbox appears.



- 2 Type the signal name.



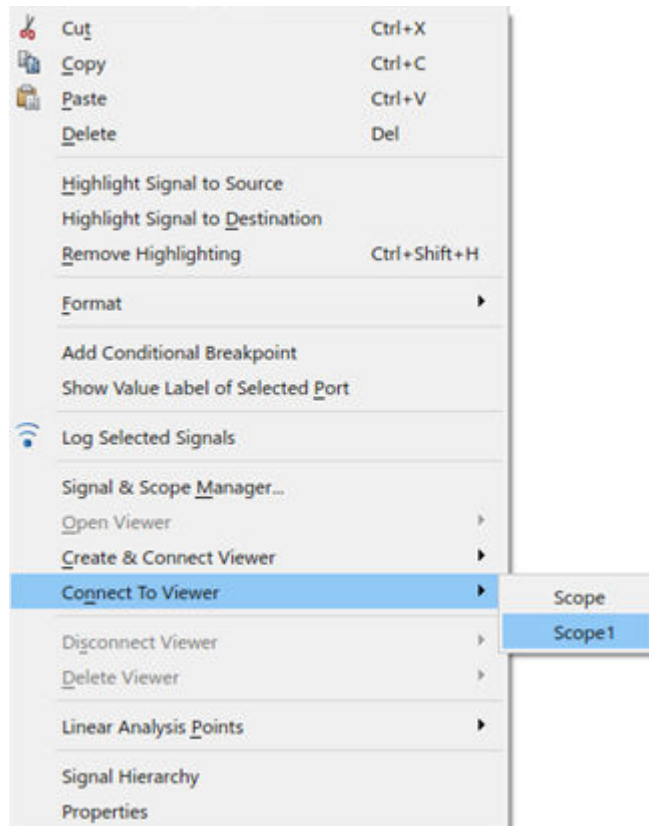
- 3 To finish, click away from the textbox.
- 4 Repeat these steps to add the names as shown.



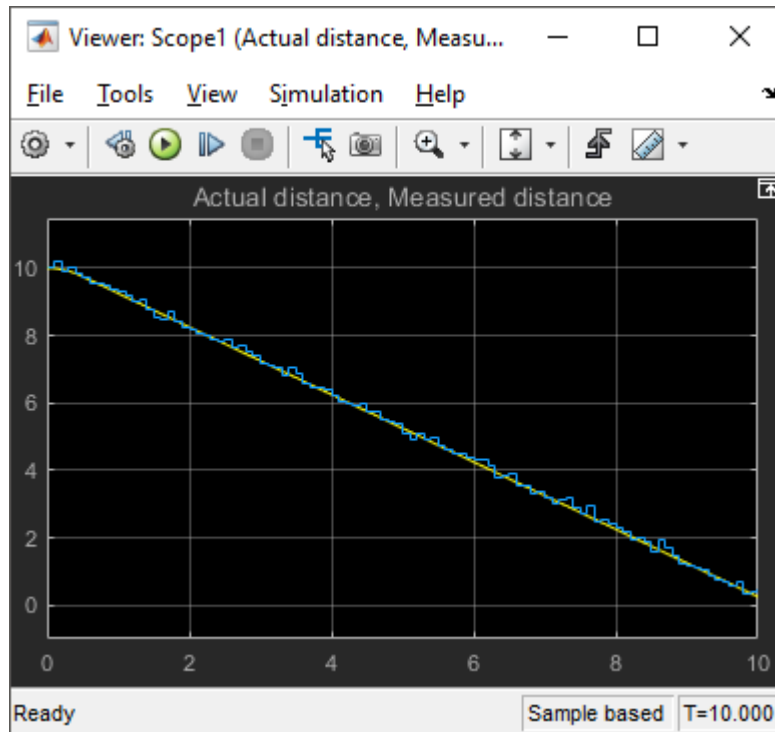
Compare Multiple Signals


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

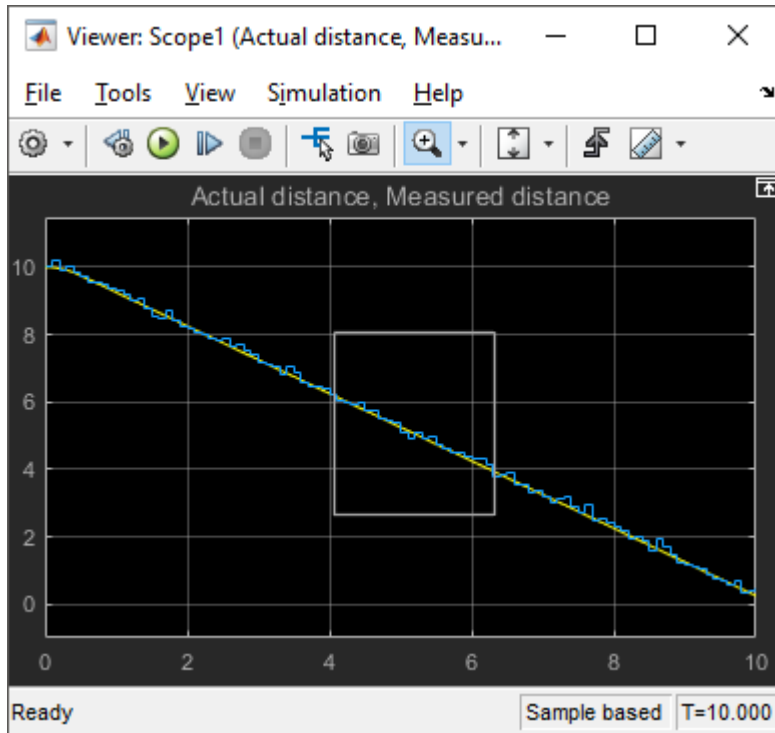
- 1 Create and connect a Scope to the *actual distance*. Note that the name of the signal appears in the viewer title.
- 2 Add the *measured distance* signal to the same viewer. Right-click the signal, and select **Connect to Viewer > Scope1**. Make sure you are connecting to the viewer you created in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



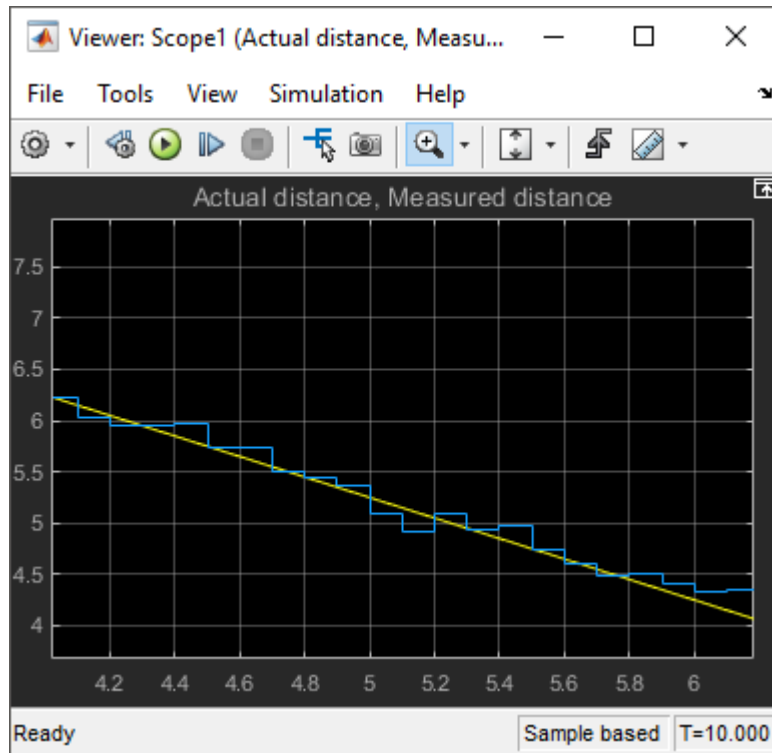
- 3 Run the model. The Viewer shows the two signals, *actual distance* in yellow and *measured distance* in blue.



- 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.



You can repeatedly zoom in to observe the details.



From the plot, you can see 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, Second-Order Integrator Limited | Zero-Order Hold

Related Examples

- “Model and Validate a System” on page 1-16

Navigate a Simulink Model

Navigate Model

In this section...

“Navigate Through Model Hierarchy” on page 4-2

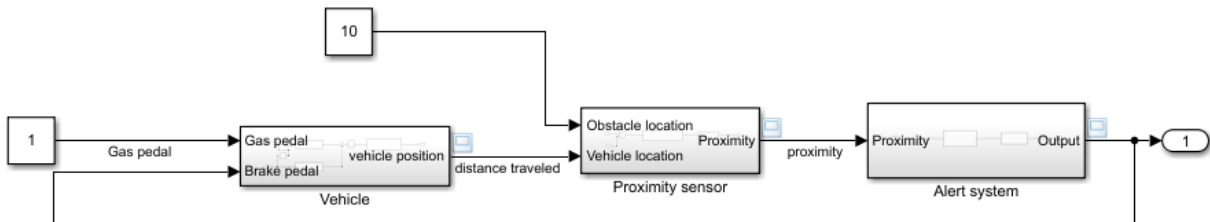
“View Signal Attributes” on page 4-4

“Trace a Signal” on page 4-7

Simulink models are hierarchical, so you can build models using both top-down and bottom-up approaches. You can view the system at a high level, then drill down to see increasing levels of model detail. This approach provides insight into how a model is organized and how parts interact.

To start, open the `smart_braking` model. In the MATLAB Command Window, enter

```
open_system(fullfile(matlabroot, ...  
'help', 'toolbox', 'simulink', 'examples', 'smart_braking'))
```




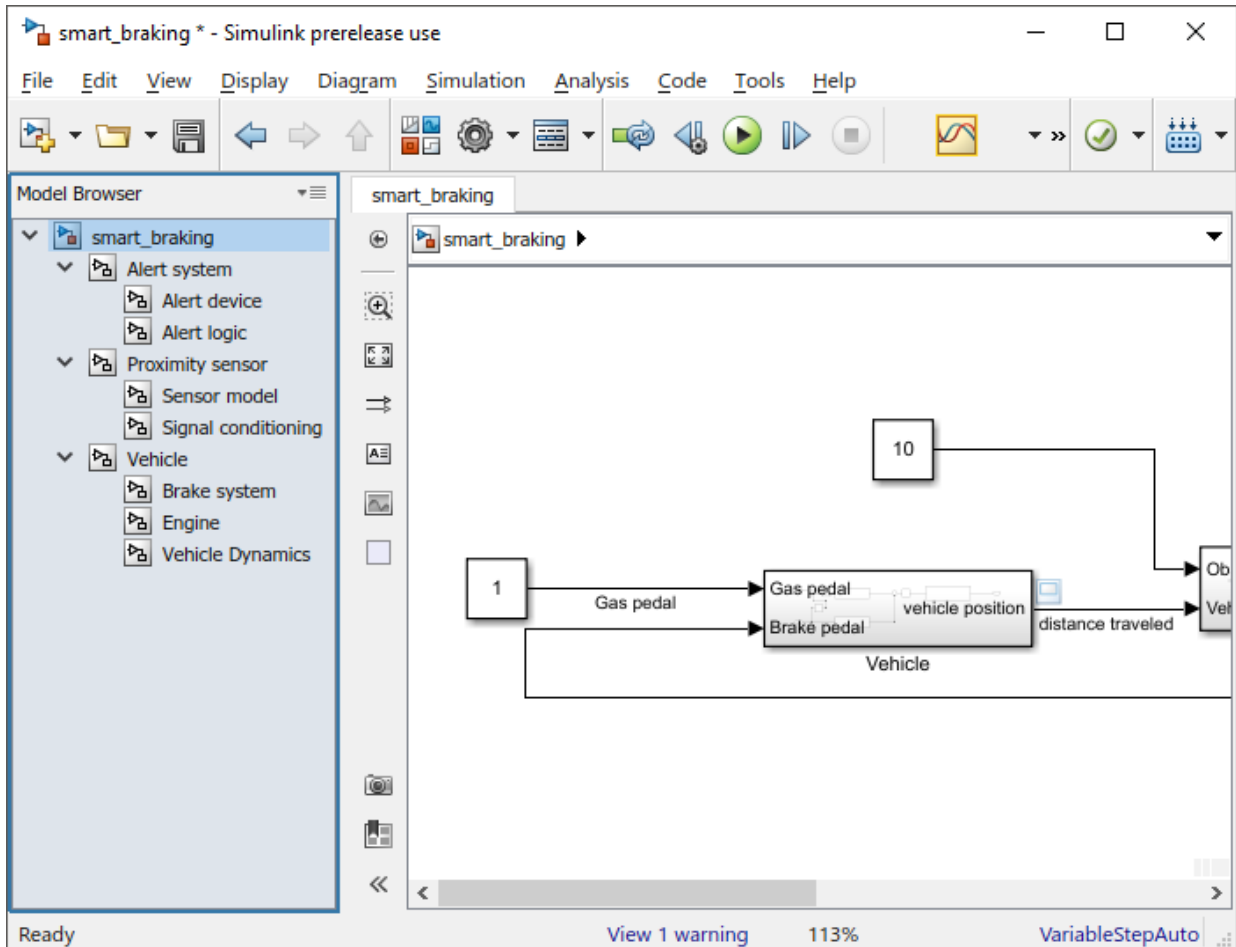
This model includes the following components and data flow:

- A vehicle moves as the gas pedal is pressed.
- A proximity sensor measures its distance from an obstacle.
- An alert system generates an alarm based on that proximity.
- The alarm automatically controls the brake to avoid hitting the obstacle.

Navigate Through Model Hierarchy

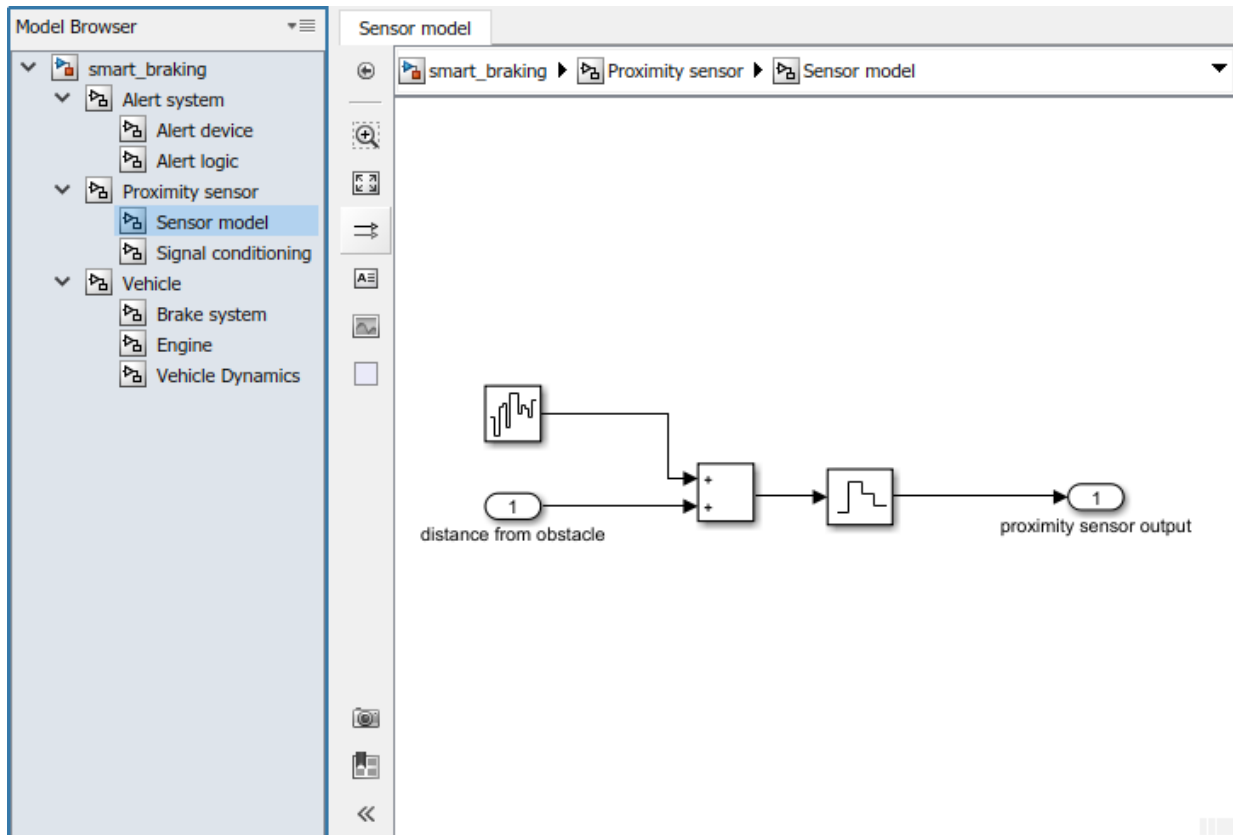
You connect blocks together to model complex components. In this model, `Vehicle`, `Proximity sensor`, and `Alert system` are all complex components with multiple blocks, and they exist in a hierarchy of subsystems. To view its contents, double-click any subsystem:

To view the complete tree, 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 also have subsystems of their own. Click > icons to see the subsystems. You can navigate through the hierarchy in the Model Browser. For example, click the Sensor model subsystem:

4 Navigate a Simulink Model



Observe that the subsystem is highlighted in the Model Browser. The address bar also shows which subsystem you are viewing. To open the subsystem in a separate window instead, right-click the subsystem name and select **Open In New Window**.

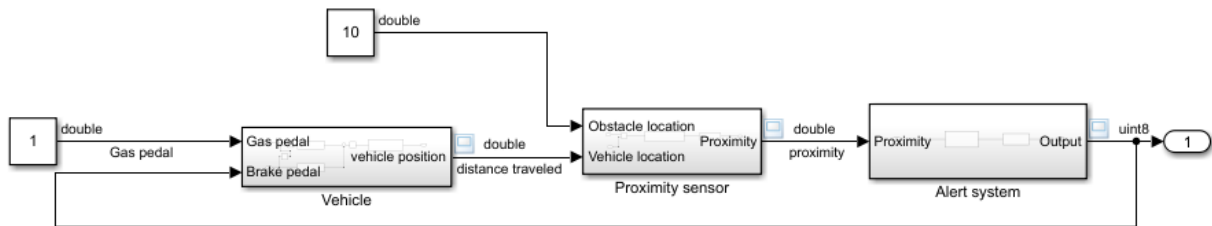
Every input or output port on a subsystem has a corresponding Inport or Outport block inside the subsystem. These blocks indicate data transfer between a subsystem and its parent. In the case of multiple inputs or outputs, the number on the block designates which port it connects to on the subsystem.

View Signal Attributes

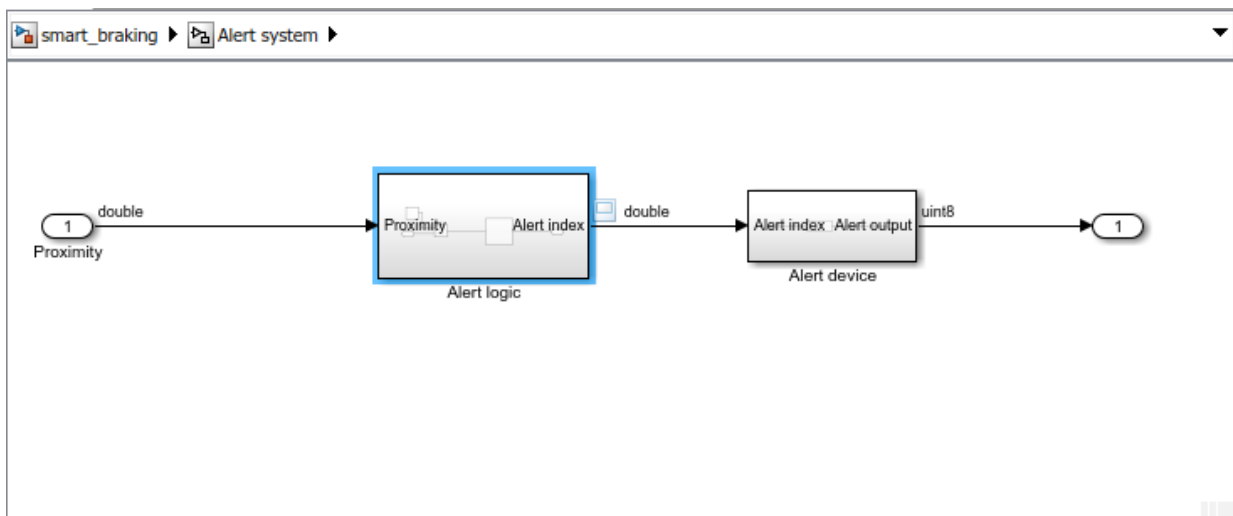
The signal lines in Simulink indicate data transfer from block to block. These signals have attributes essential to the function of the model:

- Size: Scalar, vector, or matrix
- Data type: String, double, unsigned integer, etc.
- Sample time: The fixed time interval at which this signal has an updated value, or continuous sampling

To show the data type of all signals on a model, select **Display > Signals & Ports > Port Data Types**:

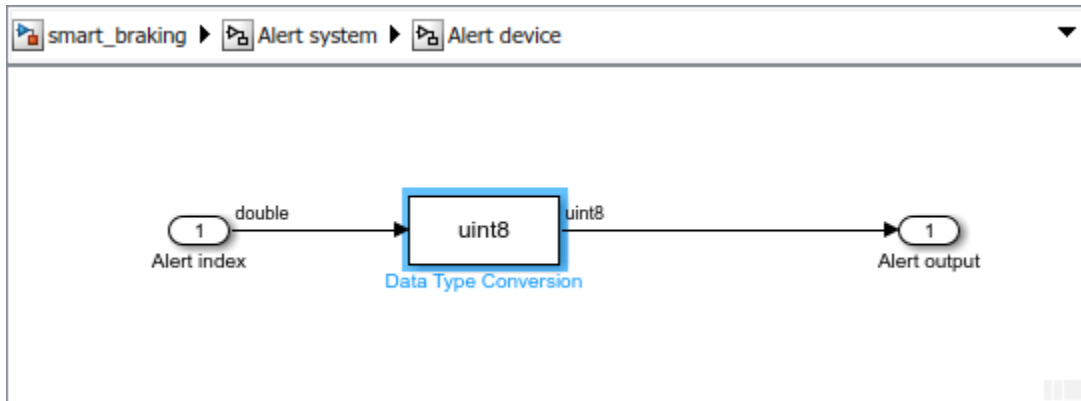


The model displays data types along the signal types. Observe that most signals are double, except the output of the Alert subsystem. Double-click this subsystem to investigate why:



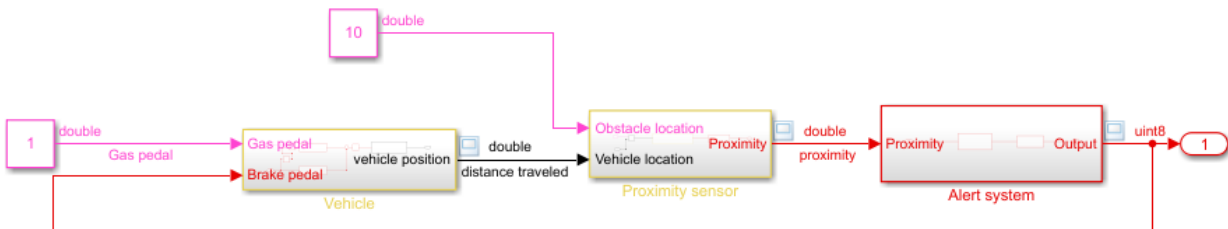
The labels in this subsystem suggests that data type change occurs in the Alert device subsystem, double-click to investigate:

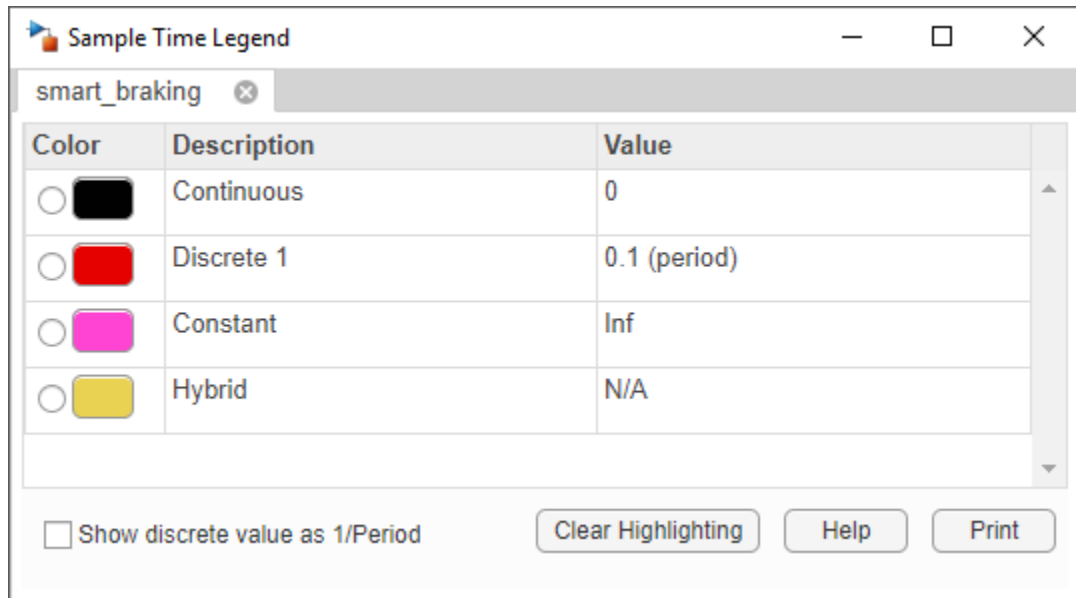
4 Navigate a Simulink Model



This shows that 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. The double data type, the default, provides the best numerical precision and is supported in all blocks. It also uses the most memory and computing power. Other numerical data types serve to model typical embedded systems where memory and computing power are limited.

To show sample times, select **Display > Sample Time > Colors**. This updates the model to show different colors for different sample times, and also displays a legend:





- A block or signal with continuous dynamics is black. They 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: They update only at fixed intervals. If the model contains components with different fixed sample times, each sample time has a different color.
- A subsystem that contains continuous and discrete components are yellow: They are hybrid systems.

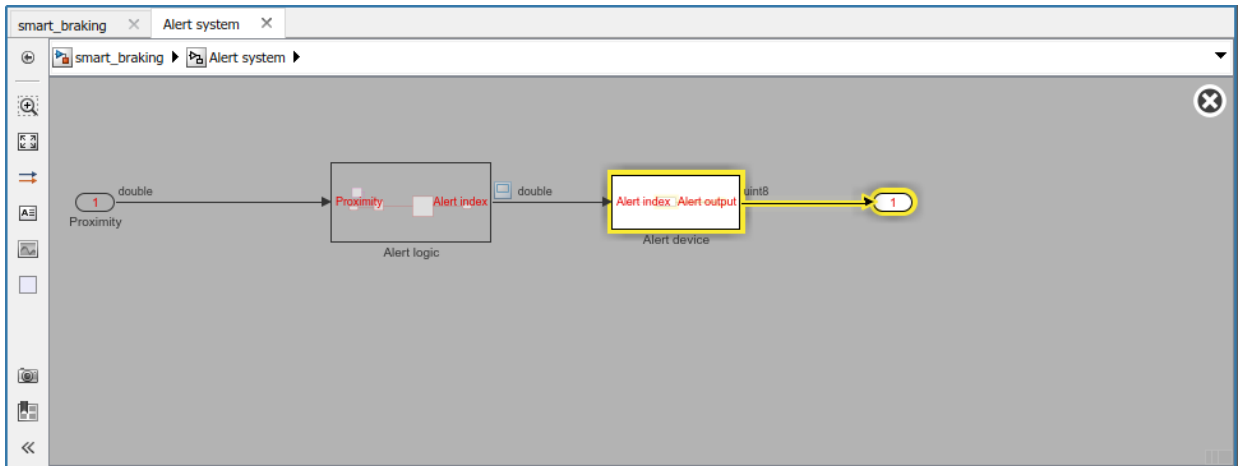
Trace a Signal

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

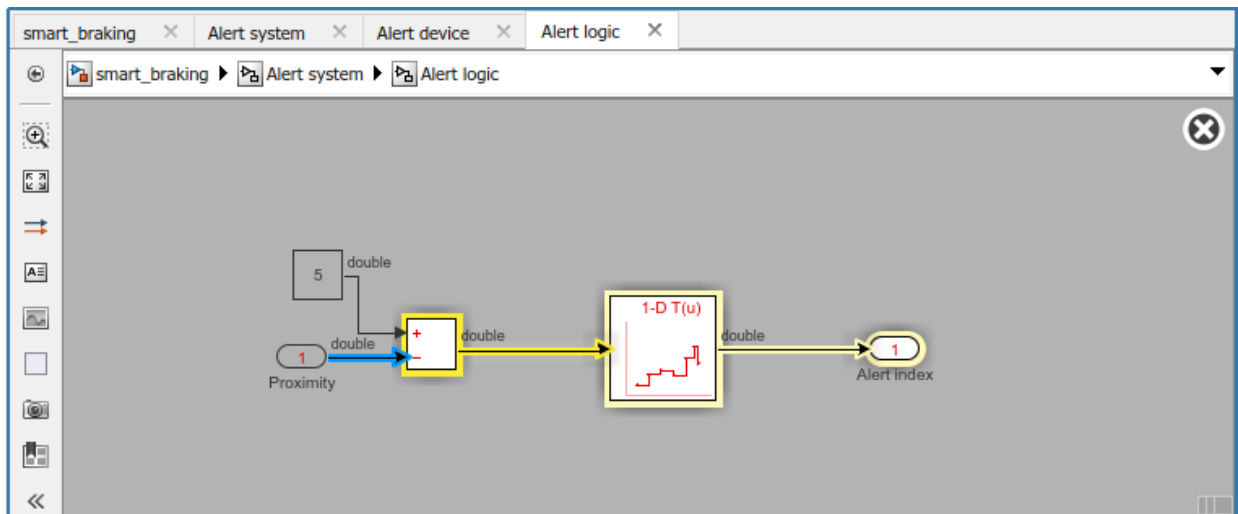
1

Open the Model Browser: Click the **Hide/Show Model Browser** button .

2 Highlight the source of the output signal: Right-click the signal and select **Highlight Signal to Source**.

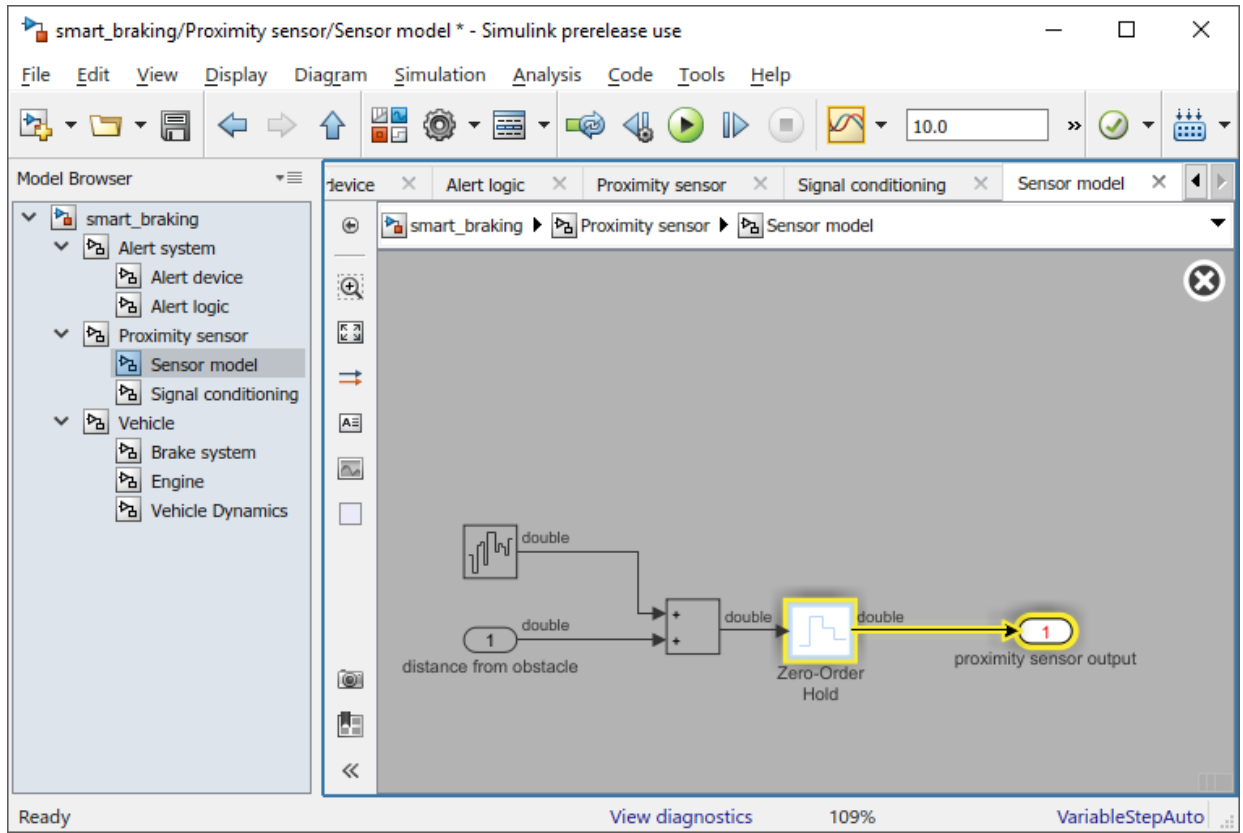


- 4 Keep tracing the signal to its source until you reach the Alert logic subsystem. You see that the Subtract block has two inputs. Choose the signal path from the Inport by pressing the down arrow key.



- 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.

4 Navigate a Simulink Model



You find that the Zero-Order Hold block in the Sensor model subsystem does the conversion from continuous to discrete.