

Simscape™ Multibody™

User's Guide



MATLAB® & SIMULINK®

R2022b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simscape™ Multibody™ User's Guide

© COPYRIGHT 2002–2022 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

March 2012	Online only	New for Version 4.0 (Release R2012a)
September 2012	Online only	Revised for Version 4.1 (Release R2012b)
March 2013	Online only	Revised for Version 4.2 (Release R2013a)
September 2013	Online only	Revised for Version 4.3 (Release R2013b)
March 2014	Online only	Revised for Version 4.4 (Release R2014a)
October 2014	Online only	Revised for Version 4.5 (Release R2014b)
March 2015	Online only	Revised for Version 4.6 (Release R2015a)
September 2015	Online only	Revised for Version 4.7 (Release R2015b)
March 2016	Online only	Revised for Version 4.8 (Release R2016a) (Renamed from <i>SimMechanics™ User's Guide</i>)
September 2016	Online only	Revised for Version 4.9 (Release R2016b)
March 2017	Online only	Revised for Version 5.0 (Release R2017a)
September 2017	Online only	Revised for Version 5.1 (Release R2017b)
March 2018	Online only	Revised for Version 5.2 (Release R2018a)
September 2018	Online only	Revised for Version 6.0 (Release R2018b)
March 2019	Online only	Revised for Version 6.1 (Release R2019a)
September 2019	Online only	Revised for Version 7.0 (Release R2019b)
March 2020	Online only	Revised for Version 7.1 (Release R2020a)
September 2020	Online only	Revised for Version 7.2 (Release R2020b)
March 2021	Online only	Revised for Version 7.3 (Release R2021a)
September 2021	Online only	Revised for Version 7.4 (Release R2021b)
March 2022	Online only	Revised for Version 7.5 (Release R2022a)
September 2022	Online only	Revised for Version 7.6 (Release R2022b)

Multibody Modeling

1	Bodies
Bodies Workflow	1-2
Bodies in the Context of a Model	1-2
Step 1: Study the Bodies to Model	1-2
Step 2: Model the Solids in Each Body	1-2
Step 3: Connect the Solids Through Frames	1-3
Step 4: Verify the Body Subsystems	1-3
Modeling Bodies	1-4
Body Elements	1-4
Relevant Blocks	1-6
Body Visualization	1-7
See It: A Typical Body	1-7
Boundaries of Bodies	1-9
Bodies as Simulink Subsystems	1-12
Compounding Body Elements	1-15
Compounding as a Modeling Strategy	1-15
Try It: Create a Compound Geometry	1-15
Try It: Create a Compound Inertia	1-18
Overview of Flexible Beams	1-20
Flexible Beam Blocks	1-20
Beam Geometries	1-20
Connection Frames	1-20
Deformation Models	1-21
Material Properties	1-21
Damping Methods	1-21
Discretization	1-22
Simulation Performance	1-22
Deformation Under Gravity	1-22
Visualization	1-23
Working with Frames	1-24
Role of Frames	1-24
Custom Solid Frames	1-25
What Are Frame Transforms?	1-27
Visualizing Frame Transforms	1-27
Try It: Specify a Frame Transform	1-28

Creating Connection Frames	1-33
Frames as a Connection Points	1-33
Creating and Transforming Frames	1-33
See It: Frames in a Typical Body	1-34
Planning Connection Frames	1-35
Addressing Assembly Errors	1-36
Representing Solid Geometry	1-38
Geometry in a Model	1-38
Preset Solid Shapes	1-40
Imported Solid Shapes	1-41
Compound Solid Shapes	1-42
Modeling Extrusions and Revolutions	1-44
Extrusions and Revolutions	1-44
The Cross-Section Profiles	1-45
Cross-Sections with Holes	1-47
From Cross-Sections to Solids	1-49
Model an Excavator Dipper Arm as a Flexible Body	1-51
Visualize a Model and Its Components	1-58
Visualize a Complete Multibody Model	1-58
Visualize an Individual Solid Geometry	1-59
A Note on Imported Geometries	1-60
Representing Solid Inertia	1-62
Representing Inertias	1-62
Compounding Solids and Inertias	1-66
Specifying Custom Inertias	1-68
Key Inertia Conventions	1-68
Inertia Matrix Definitions	1-68
CAD as an Inertia Data Source	1-71
Automatic Inertia Calculation	1-74
Specifying Variable Inertias	1-76
Modeling Variable Inertias	1-76
Visualizing Variable Inertias	1-76
Modeling Body Interactions	1-77
Model a Variable-Mass Oscillator	1-77
Creating Custom Solid Frames	1-82
Solid Frames	1-82
Opening the Frame-Creation Interface	1-83
Geometry-Based Frame Placement	1-83
Primary and Secondary Axes	1-83
Try It: Create a Custom Solid Frame	1-84
Manipulate the Color of a Solid	1-90
Visual Property Parameterizations	1-90
RGB and RGBA Vectors	1-90
Simple Visual Properties	1-91
Advanced Visual Properties	1-92
Adjust Solid Opacity	1-92

Adjust Highlight Color	1-93
Adjust Shadow Color	1-93
Adjust Self-Illumination Color	1-94

Multibody Systems

2

Multibody Assembly Workflow	2-2
Study the Joints and Constraints to Model	2-2
Assemble Bodies Using Joints and Constraints	2-2
Guide Model Assembly	2-2
Verify Model Assembly	2-3
Modeling Joint Connections	2-4
Joint Degrees of Freedom	2-4
Joint Primitives	2-5
Joint Inertia	2-6
How Multibody Assembly Works	2-8
Model Assembly	2-8
Connecting Joints	2-8
Orienting Joints	2-9
Guiding Assembly	2-9
Verifying Model Assembly	2-10
Counting Degrees of Freedom	2-12
Model an Open-Loop Kinematic Chain	2-13
Model Overview	2-13
Build Model	2-13
Guide Model Assembly	2-14
Visualize Model and Check Assembly Status	2-14
Simulate Model	2-15
Open Reference Model	2-15
Model a Closed-Loop Kinematic Chain	2-16
Build Model	2-16
Specify Block Parameters	2-18
Guide Assembly and Visualize Model	2-19
Verify Model Assembly	2-19
Simulate Model	2-20
Troubleshoot an Assembly Error	2-21
Model Overview	2-21
Explore Model	2-21
Update Model	2-23
Troubleshoot Assembly Error	2-23
Correct Assembly Error	2-25
Simulate Model	2-25
Modeling Gear Constraints	2-27
Gear Constraints and Applications	2-27

Gear Assemblies as Kinematic Loops	2-28
Gear Assembly Restrictions	2-29
Gear Pitch Circles	2-30
Modeling Gear Geometries	2-30
Limitations of Gear Constraints	2-31
Assemble a Gear Model	2-32
Gear Examples	2-32
Bevel Gear	2-33
External Spur Gear	2-36
Internal Spur Gear	2-39
Rack and Pinion	2-41
Worm and Gear	2-44
Model a Compound Gear Train	2-47
Model Overview	2-47
Model Sun-Planet Gear Set	2-47
Constrain Sun-Planet Gear Motion	2-50
Add Ring Gear	2-51
Add Gear Carrier	2-54
Add More Planet Gears	2-57
Constrain a Point to a Curve	2-58
Open the Flap Assembly Model	2-58
Effect of Constraints on the Model	2-59
Create the Connection Frames	2-60
Connect the Constraint Blocks	2-61
Specify the Flap Constraint Curves	2-62
Simulate the Model	2-63

Internal Mechanics, Actuation and Sensing

3

Modeling and Sensing System Dynamics	3-2
Provide Joint Actuation Inputs	3-2
Specify Joint Internal Mechanics	3-2
Model Body Interactions and External Loads	3-2
Sense Dynamical Variables	3-3
Modeling Gravity	3-4
Gravity Models	3-4
Gravitational Force Magnitude	3-5
Force Position and Direction	3-5
Gravitational Torques	3-6
Model Gravity in a Planetary System	3-8
Model Overview	3-8
Step 1: Start a New Model	3-8
Step 2: Add the Solar System Bodies	3-9
Step 3: Add the Degrees of Freedom	3-12
Step 4: Add the Initial State Targets	3-13
Step 5: Add the Gravitational Fields	3-16

Step 6: Configure and Run the Simulation	3-17
Open an Example Model	3-18
Specifying Joint Actuation Inputs	3-19
Actuation Modes	3-19
Motion Input	3-21
Input Handling	3-22
Assembly and Simulation	3-23
Specifying Motion Input Derivatives	3-24
Joint Actuation Limitations	3-26
Closed Loop Restriction	3-26
Motion Actuation Not Available in Spherical Primitives	3-26
Redundant Actuation Mode Not Supported	3-26
Model Report and Mechanics Explorer Restrictions	3-26
Motion-Controlled DOF Restriction	3-27
Actuating and Sensing with Physical Signals	3-28
Exposing Physical Signal Ports	3-28
Converting Actuation Inputs	3-28
Obtaining Sensing Signals	3-29
Sensing	3-31
Sensing Overview	3-31
Variables You Can Sense	3-31
Blocks with Sensing Capability	3-32
Sensing Output Format	3-32
Force and Torque Sensing	3-33
Blocks with Force and Torque Sensing	3-33
Joint Forces and Torques You can Sense	3-33
Force and Torque Measurement Direction	3-34
Modeling Contact Force Between Two Solids	3-36
Spatial Contact Force Block Forces	3-37
Sensing	3-37
Connect to Solid Blocks	3-38
Considerations for Contact Modeling	3-38
Use Contact Proxies to Simulate Contact	3-40
How to Use Contact Proxies	3-40
Examples	3-41
Model Wheel Contact in a Car	3-49
Model a Rolling Wheel	3-49
Model a Bumper Car	3-52
Motion Sensing	3-56
Sensing Spatial Relationships Between Joint Frames	3-56
Sensing Spatial Relationships Between Arbitrary Frames	3-57
Rotational Measurements	3-60
Rotation Sensing Overview	3-60
Measuring Rotation	3-60
Axis-Angle Measurements	3-60

Quaternion Measurements	3-61
Transform Measurements	3-61
Rotation Sequence Measurements	3-62
Translational Measurements	3-65
Translation Sensing Overview	3-65
Measuring Translation	3-65
Cartesian Measurements	3-66
Cylindrical Measurements	3-66
Spherical Measurements	3-67
Selecting a Measurement Frame	3-69
Measurement Frame	3-69
Example	3-70
Sense Motion Using a Transform Sensor Block	3-79
Model Overview	3-79
Modeling Approach	3-79
Build Model	3-80
Guide Model Assembly	3-81
Simulate Model	3-81
Save Model	3-83
Specify Joint Actuation Torque	3-84
Model Overview	3-84
Four-Bar Linkages	3-84
Modeling Approach	3-86
Build Model	3-87
Simulate Model	3-90
Analyze Motion at Various Parameter Values	3-94
Model Overview	3-94
Build Model	3-94
Specify Block Parameters	3-96
Create Simulation Script	3-97
Run Simulation Script	3-97
Measure Forces and Torques Acting at Joints	3-99
Open Model	3-100
Sense Actuation Torque	3-100
Sense Constraint Forces	3-102
Sense Total Torques	3-103
Sense Constraint Forces	3-105
Model Overview	3-105
Add Weld Joint Block to Model	3-106
Add Constraint Force Sensing	3-106
Add Damping to Joints	3-107
Simulate Model	3-107
Plot Constraint Forces	3-108
Specify Joint Motion Profile	3-110
Build Model	3-110
Simulate Model	3-112

Specify Joint Motion in Planar Manipulator Model	3-114
Add Virtual Joint	3-114
Prescribe Motion Inputs	3-115
Sense Joint Actuation Torques	3-118
Simulate Model	3-119

Simulation and Analysis

Simulation

4

Update and Simulate a Model	4-2
Create or Open a Model	4-2
Update the Block Diagram	4-2
Examine the Model Assembly	4-3
Configure the Solver Settings	4-3
Run Simulation and Analyze Results	4-3
Multibody Simulation Issues	4-4

Visualization and Animation

5

Enable Mechanics Explorer	5-2
Working with Animation	5-3
Animation Playback	5-3
Looping Playback	5-3
Changing Playback Speed	5-3
Jumping to Playback Time	5-3
Manipulate the Visualization Viewpoint	5-4
Model Visualization	5-4
Select a Standard View	5-4
Set View Convention	5-5
Rotate, Roll, Pan, and Zoom	5-6
Split Model View	5-7
Visualization Cameras	5-8
Camera Types	5-8
Global Camera	5-9
Dynamic Cameras	5-9
Camera Trajectory Modes	5-9
Dynamic Camera Selection	5-10
Dynamic Camera Reuse	5-11

Create a Dynamic Camera	5-12
Start a New Camera Definition	5-12
Define a Keyframes Camera	5-12
Define a Tracking Camera	5-13
Select a Dynamic Camera	5-14
Selective Model Visualization	5-15
What Is Visualization Filtering?	5-15
Changing Component Visibility	5-15
Visualization Filtering Options	5-16
Components You Can Filter	5-16
Model Hierarchy and Tree Nodes	5-17
Filtering Hierarchical Subsystems	5-17
Updating Models with Hidden Nodes	5-18
Alternative Ways to Enhance Visibility	5-18
Selectively Show and Hide Model Components	5-20
Visualization Filtering	5-20
Open Example Model	5-20
Update Example Model	5-21
Hide Half-Cylinder Subsystem	5-21
Show Solid in Hidden Subsystem	5-22
Show Only Piston Subsystem	5-22
Show Everything	5-23
Visualize Simscape Multibody Frames	5-24
What Are Frames?	5-24
Show All Frames	5-24
Highlight Specific Frames	5-25
Visualize Frames via Graphical Markers	5-26
Go to a Block from Mechanics Explorer	5-27
Create a Model Animation Video	5-28
UI and Command-Line Tools	5-28
Before Creating a Video	5-28
Create a Video Using Video Creator	5-28
Create a Video Using smwritevideo	5-29

Multibody Model Import

CAD and URDF Model Import

6

CAD Translation	6-2
Translating a CAD Model	6-2
What's in a Translated Model?	6-2
What's in a Data File?	6-5
Exporting a CAD Model	6-5
Importing a CAD Model	6-6

Simplifying Model Topology	6-7
Updating an Existing Data File	6-8
Install the Simscape Multibody Link Plugin	6-9
Software Requirements	6-9
Step 1: Get the Installation Files	6-9
Step 2: Run the Installation Function	6-10
Step 3: Register MATLAB as an Automation Server	6-10
Step 4: Enable the Simscape Multibody Link Plugin in a CAD Application	6-10
Importing CAD Files from Applications Not Supported by Simscape Multibody Link	6-10
Import a CAD Assembly Model	6-12
Before You Begin	6-12
Example Files	6-12
Import a Model	6-12
After Model Import	6-13
Import a Robotic Arm CAD Model	6-14
Example Overview	6-14
Example Files	6-14
Import the Model	6-14
Visualize the Model	6-15
Build on the Model	6-16
Onshape Import	6-17
What Is Onshape?	6-17
What's in an Onshape Model?	6-17
Preparing a Model for Import	6-18
Importing an Onshape Model	6-18
What Can You Import?	6-19
User Authentication and Account Permissions	6-19
Mapping to Simscape Multibody Blocks	6-19
Onshape Import Warnings and Errors	6-20
Physical Units	6-20
Obtaining Onshape Models to Import	6-21
Import an Onshape Humanoid Model	6-22
Onshape Import	6-22
Example Overview	6-22
Export the Model	6-22
Import the Model	6-23
URDF Primer	6-25
URDF Elements and Attributes	6-25
XML Hierarchies and Kinematic Trees	6-26
Required and Optional URDF Entities	6-28
Create a Simple URDF Model	6-29
Obtaining URDF Models to Import	6-31
Import URDF Models	6-33
Supported URDF Elements and Attributes	6-33
Mapping to Simscape Multibody Blocks	6-34
Mesh Geometries	6-35

Physical Units	6-35
URDF Import Limitations	6-36
Differences from CAD Import	6-36
Import a Simple URDF Model	6-36
Import a URDF Humanoid Model	6-40
URDF Import	6-40
Example Overview	6-40
Import the Model	6-40

Deployment

	Code Generation
7	
Code Generation Applications	7-2
Code Generation Overview	7-2
Simulation Acceleration	7-2
Model Deployment	7-3
Code Generation Setup	7-4
Before You Begin	7-4
Solver Selection	7-4
Target Selection	7-4
Run-Time Parameters	7-4
Compiler Optimization	7-5
Generate Code for a Multibody Model	7-6

Examples

	Simscape Multibody Examples
8	
Elevator	8-2
Cable Driven Space Manipulator	8-5
Using the Spatial Contact Force Block - Bumper Car	8-9
Full Vehicle on Four Post Testrig	8-10
Single Pendulum in Simulink and Simscape Multibody	8-11

Double Pendulum in Simulink and Simscape Multibody	8-14
Creating Frames Using Rigid Transforms	8-18
Creating a Simple Part	8-19
Creating a Complex Part	8-20
Assembling Parts into a Double Pendulum	8-21
Assembling Parts into a Four Bar Mechanism	8-22
How to Build a Model	8-23
Using the Lead Screw Joint Block - Linear Actuator	8-46
Modeling Constant Velocity Joints - Power Take-Off Shaft	8-47
Using the Common Gear Block - Cardan Gear Mechanism	8-48
Using the Rack-Pinion Block - Windshield Wiper Mechanism	8-49
Using the Worm and Gear Constraint Block - Solar Tracker	8-50
Using the Point-On-Curve Block : Flapping Wing Mechanism	8-51
Computing Actuator Torques Using Inverse Dynamics	8-52
Sensing Composite Forces and Torques in Joints - Potter's Wheel	8-54
Modeling Self-Locking Worm and Gear Constraints - Worm Jack ..	8-55
Rotational Interface : Electrically Operated Bread Slicer	8-56
Translational Interface : Radial Engine with Gas Force Model	8-57
Hydraulic Interface - Dump Trailer with Hydraulic Cylinder	8-58
Stewart Platform with Controller	8-60
Four Bar Mechanism Imported from a CAD Assembly	8-61
Modeling A Robot Using STEP Files	8-62
Humanoid Robot	8-63
Configuring Dynamic Cameras - Vehicle Slalom	8-64
Airplane Wing Landing Gear	8-65
Inverted Double Pendulum on a Sliding Cart	8-66
Stewart Platform	8-67

Pendulum Waves	8-68
Double Wishbone Suspension	8-69
Independent Suspension System Templates	8-70
Backhoe	8-71
3-Roll Robotic Wrist Mechanism	8-73
Fairground Carousel Ride	8-74
Pick and Place Robot Using Forward and Inverse Kinematics	8-75
Cable Robot	8-80
Cable-Driven XY Table with Cross Base	8-81
Pulley Mechanism Right Angle Drive	8-87
Tower Crane With Trolley and Hoist	8-90
Block and Tackle with Four Pulleys	8-95
Using the Common Gear Block	8-98
Lead Screw with Friction	8-104
Forklift	8-109
Flexible Dipper Arm	8-113
Train Humanoid Walker	8-114
Cartesian 3D Printer	8-121
Ratchet Lifter	8-124
Modeling and Control of a Mars Rover	8-129
Creating a Mobile Robot using a MATLAB App	8-147
Creating a Robotic Gripper Multibody in MATLAB	8-150
Creating a Four Bar Multibody Mechanism in MATLAB	8-152
Creating a Simple Pendulum in MATLAB	8-155
Creating a Multibody with different joints in MATLAB	8-157
Perform Forward and Inverse Kinematics on a Five-Bar Robot ..	8-162
Package Delivery Quadcopter	8-173

How to Build a Multibody System in MATLAB	8-178
Vehicle Dynamics - Car with Heave and Roll	8-195
Contact Modeling - Ball on Grid Surface	8-196

Multibody Modeling

Bodies

- “Bodies Workflow” on page 1-2
- “Modeling Bodies” on page 1-4
- “Compounding Body Elements” on page 1-15
- “Overview of Flexible Beams” on page 1-20
- “Working with Frames” on page 1-24
- “Creating Connection Frames” on page 1-33
- “Representing Solid Geometry” on page 1-38
- “Modeling Extrusions and Revolutions” on page 1-44
- “Model an Excavator Dipper Arm as a Flexible Body” on page 1-51
- “Visualize a Model and Its Components” on page 1-58
- “Representing Solid Inertia” on page 1-62
- “Specifying Custom Inertias” on page 1-68
- “Specifying Variable Inertias” on page 1-76
- “Creating Custom Solid Frames” on page 1-82
- “Manipulate the Color of a Solid” on page 1-90

Bodies Workflow

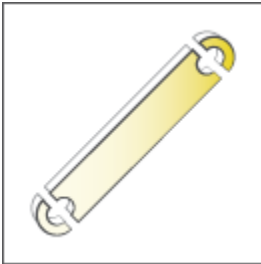
In this section...

“Bodies in the Context of a Model” on page 1-2
“Step 1: Study the Bodies to Model” on page 1-2
“Step 2: Model the Solids in Each Body” on page 1-2
“Step 3: Connect the Solids Through Frames” on page 1-3
“Step 4: Verify the Body Subsystems” on page 1-3

Bodies in the Context of a Model

Bodies are the core constituents of a model. In order to assemble bodies into a mechanism or machine, you must first model those bodies. Here are the recommended steps to follow when performing this task.

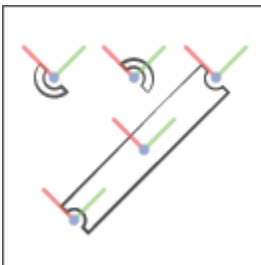
Step 1: Study the Bodies to Model



Bodies are collections of solids and occasionally other body elements. Start your model by conceptually breaking down each body into shapes that you can specify using the different solid blocks. Obtain the dimensions, inertia, and color of each solid. For more information on bodies and their nature as (normally rigid) collections of body elements, see:

- “Modeling Bodies” on page 1-4
- “Compounding Body Elements” on page 1-15

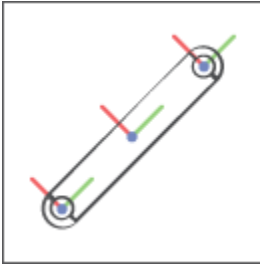
Step 2: Model the Solids in Each Body



Body elements are defined by their material attributes. Specify these attributes using solid blocks, the inertia block, and other blocks from the Body Elements library. For more information on body elements and their attributes, see:

- “Representing Solid Geometry” on page 1-38
- “Representing Solid Inertia” on page 1-62
- “Manipulate the Color of a Solid” on page 1-90

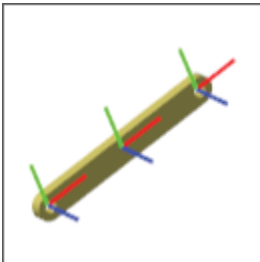
Step 3: Connect the Solids Through Frames



The placement of a body element in a body depends on how its frames are defined. Create and connect the frames required to assemble the various body elements into a complete body. For more information on frames and frame connections, see:

- “Working with Frames” on page 1-24
- “Creating Connection Frames” on page 1-33
- “Creating Custom Solid Frames” on page 1-82

Step 4: Verify the Body Subsystems



Visualize each body and verify its geometry, color, and frames. You can use the block visualization pane to visualize an individual solid or Mechanics Explorer to visualize a complete body. For more information about visualization, see:

- “Visualize a Model and Its Components” on page 1-58
- “Selective Model Visualization” on page 5-15
- “Manipulate the Visualization Viewpoint” on page 5-4

Modeling Bodies

In this section...
“Body Elements” on page 1-4
“Relevant Blocks” on page 1-6
“Body Visualization” on page 1-7
“See It: A Typical Body” on page 1-7
“Boundaries of Bodies” on page 1-9
“Bodies as Simulink Subsystems” on page 1-12

Bodies are the basic constituents of a multibody model. They are the solid components that you connect when assembling a model and the subjects upon which all forces and torques in that model ultimately act. The planets and the sun serve as bodies in a model of the solar system, for example, and wings do too in a model of a flapping mechanism.



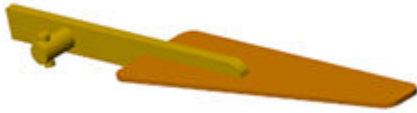
A Flapping Mechanism as an Assembly of Bodies

If you are familiar with CAD modeling, you can loosely think of bodies as equivalents of CAD parts, each a modular component with geometry and material. The modeling approach may differ—for example, there are no sketches to draw—but, conceptually, the end result is the same: something that you can connect, constrain, act upon, and visualize.

Body Elements

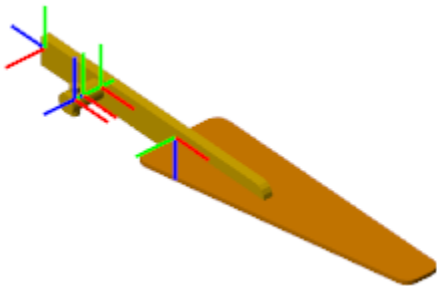
A body is a (normally rigid) collection of simpler body elements: solids with geometry and inertia and, less often, plain inertias (without assigned geometries) and plain geometries (without associated inertias). The body elements that you model and the ways in which you connect them determine the overall attributes of the body.

Bodies with simple shapes often require a single solid. Those with complex shapes or inertias may require several solids and the occasional plain inertia or plain geometry. Like bricks in a modular build set, the properties of the various body elements can differ from each other. It is up to you to combine them in a way that produces the desired body.



A Wing Body as a Collection of Solids

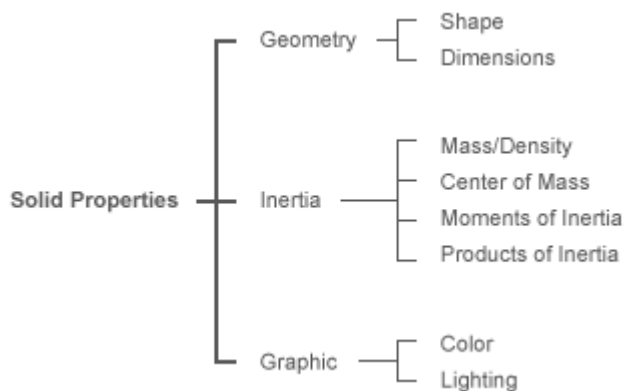
Each body element comprises one or more frames and a set of material attributes. The frames determine the placement of the body elements relative to each other and provide the attachment points for joints and constraints. The attributes factor in the dynamic behavior of the body elements and, in those with geometry, help determine their visual appearance.



Frames on a Wing Body

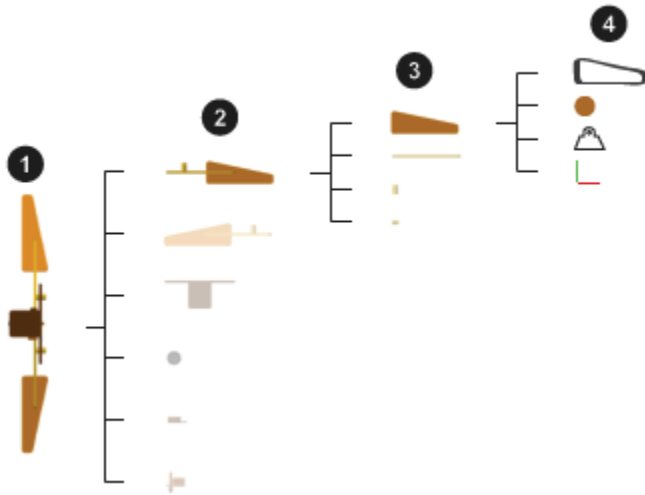
Material Attributes

The attributes of solids include inertia and color. Geometry enables the automatic calculation of inertia parameters and, in conjunction with color, visualization. Inertia quantifies the resistance to changes in motion and factor into the calculation of the forces and torques required to induce an acceleration.



The attributes of other body elements are more limited in scope. Those of plain inertias include only inertia, with parameters that in the general case encompass mass, center of mass, moments of inertia, and products of inertia.

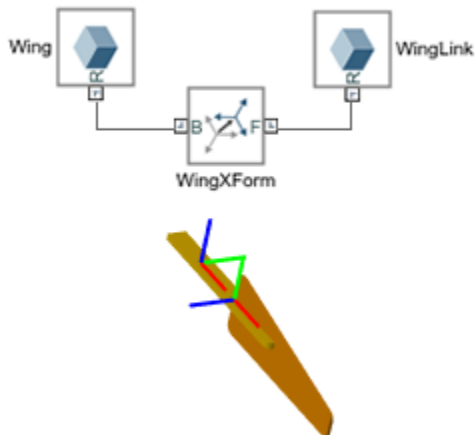
The figure summarizes the structure of a typical body (here a wing) in the context of a typical multibody model (here a flapping wing mechanism). Multibody assemblies (1) comprise bodies (2), bodies comprise body elements (3)—often all solids—and body elements comprise frames and any material attributes relevant to them (4).



Relevant Blocks

You can model most bodies using the solid blocks and Rigid Transform block. The solid blocks represent their namesake—a solid element of a certain type. The block parameters set the attributes of the solid and a frame port, labeled **R**, provides a reference frame for connection to a model. You can create additional, custom, frames and position them using a variety of solid features.

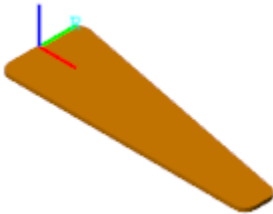
The Rigid Transform block represents a fixed spatial relationship between two frames. Whenever you add a Rigid Transform block to a frame connection line, you replace the coincidence relationship originally set by that line with a rotational and translational offset that you specify in the block dialog box.



Only rarely do bodies contain other blocks. Those that remain in the Body Elements library—Graphic, Inertia, and all in the Variable Mass sublibrary—serve special cases. Use them, for example, to add graphic markers, adjust inertia through compounding, or allow inertia to vary during simulation.

Body Visualization

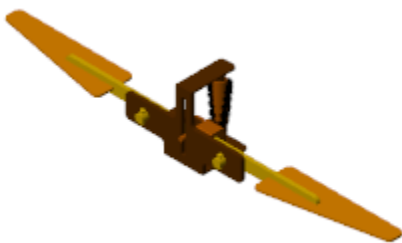
You can visualize individual solids and complete models. The right visualization tool to use depends on which of the two you want to visualize. You can visualize individual solids directly through their respective solid blocks. In its dialog box, The solid blocks provide a visualization pane that shows the geometry, frames, and color of the solid that it represents. The visualization works even if the block diagram is incomplete or invalid.



A Typical Solid Visualization

You can visualize a model, and all the bodies within it, using the Simscape Multibody visualization utility, Mechanics Explorer. The model must be free of kinematic conflicts, such as those due to mutually incompatible joints and constraints. It must also contain one Solver Configuration block for every topologically distinct multibody network—each a group of Simscape Multibody blocks that connect without breaks in their frame connection lines.

Mechanics Explorer opens a static model visualization in its initial configuration whenever you do a diagram update (in the **Debug** tab, click **Update Model**). The initial configuration is the aggregate result of all initial joint positions and angles, whose values you can specify using joint state targets. The visualization becomes dynamic when you run a simulation, although this is a task you are unlikely to perform while still modeling bodies.



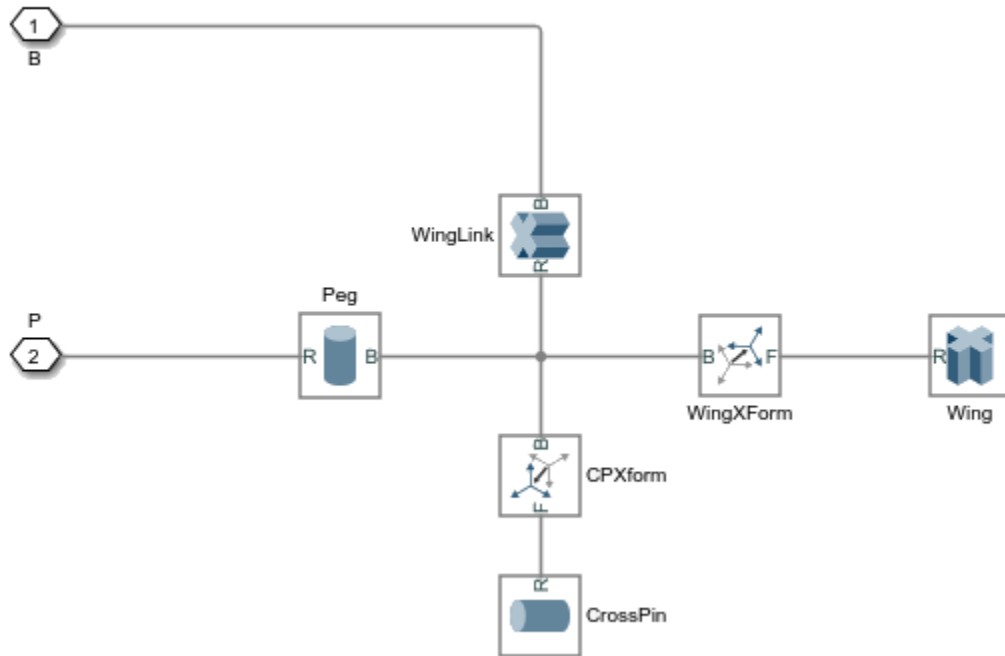
A Typical Model Visualization

For more information about visualization, see “Visualize a Model and Its Components” on page 1-58.

See It: A Typical Body

At the MATLAB command prompt, enter `sm_cam_flapping_wing`. A model of a flapping wing mechanism opens up. Look inside the mask of the subsystem block named `RightWing`. You can do

this by clicking the down arrow at the bottom left corner of the block. The blocks inside the subsystem are a typical representation of a body.



The solid blocks each represent a section of the wing body. The frame connection lines between the solid blocks, and the Rigid Transform blocks that some of the connection lines contain, define the spatial relationships that exist between the solid sections. Open the dialog box of the Cylindrical Solid block named **CrossPin** and explore its attributes:

- The **Geometry** parameters are by default expanded. The Cylindrical Solid block uses a shape with its relevant dimensions (**Radius**, **Length**) parameterized in terms of MATLAB variables (R_p , L_p). All variables are defined numerically in the subsystem mask.

Geometry			
Radius	$R_p/2$	cm	▼
Length	$3 \cdot R_p$	cm	▼

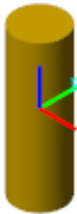
- Expand the **Inertia** parameters. The block is configured to calculate the bulk of the inertia parameters from geometry and a mass parameter (**Density**). This parameter too is parameterized in terms of a MATLAB variable (ρ).

Inertia			
Type	Calculate from Geometry		▼
Based on	Density		▼
Density	ρ	kg/(m ³)	▼ Compile-time ▼

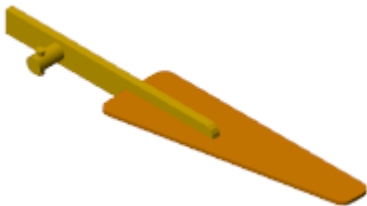
- Expand the **Graphic** parameters. The block uses a **Simple** color model with the visual properties of the solid (**Color**, **Opacity**) parameterized in terms of MATLAB variables (`lclr`) or specified numerically (`1.0`).

Graphic	
Type	From Geometry ▼
Visual Properties	Simple ▼
Color	<code>lclr</code> Compile-time ▼
Opacity	<code>1.0</code> Compile-time ▼

- In the visualization toolbar, click the frame button. The visualization pane shows the frames associated with the solid. This solid has a single frame, the local reference frame that by default every solid block has. The placement of this frame relative to the geometry impacts the placement of the geometry in the context of the model.



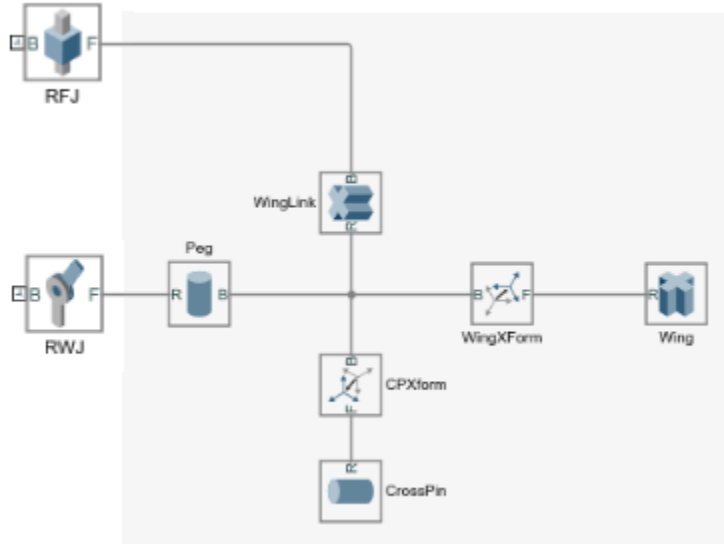
- Update the block diagram. Mechanics Explorer opens with a static visualization of the flapping wing model in its initial configuration. In the tree view pane (located on the left side of Mechanics Explorer), right-click the **RightWing** node and select **Show Only This**. The visualization pane updates to show only the body elements that compose the selected component.



Boundaries of Bodies

Direct connection lines and Rigid Transform blocks unite body elements into a single body. Such connections are treated as internal to the bodies they belong to. Joint and constraint blocks in turn separate solids into different bodies. These blocks identify the boundaries of the bodies that they connect. This distinction has practical consequences in models with Gravitational Field blocks.

The figure shows a flattened portion of the `sm_cam_flapping_wing` block diagram. The joint blocks named `RFJ` and `RWJ` separate the blocks that compose the body to their right (identified by the shaded area) from the neighboring bodies to which it connects (not shown).

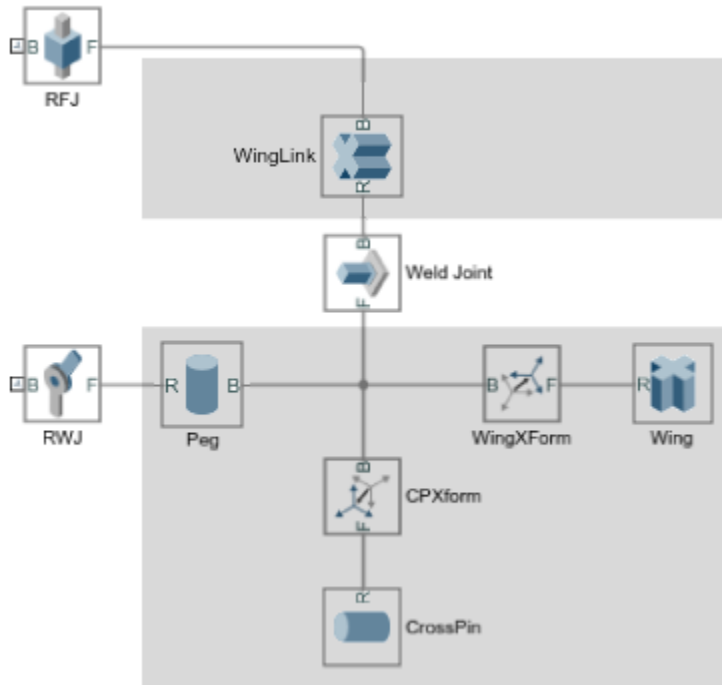


Impact on Gravitational Fields

By design, the Gravitational Field block exerts a force on the center of mass of a body. The center of mass is determined from the aggregate of all Body Elements blocks that comprise the body. If two solid blocks connect through a Rigid Transform block, they belong to the same body. A single gravitational force then acts at the center of mass of that body.

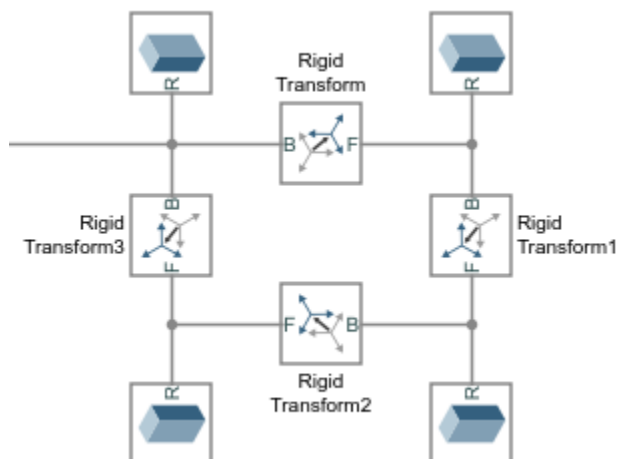
If, however, the Cylindrical Solid blocks connect through a Revolute Joint block, they belong to separate bodies. Two gravitational forces then apply at the individual centers of mass of those bodies. The same is true even if you replace the Revolute Joint block with a Weld Joint block. Regardless of its type, every joint block separates the body elements that it connects into separate bodies.

The figure shows the effect of adding a Weld Joint block to a block network that originally composed a single body. This joint block divides the body into two bodies, one comprising only the Cylindrical Solid block named *WingLink*, the other comprising the remaining Cylindrical Solid and Rigid Transform blocks. Gravitational Field blocks in your model, if any, would in this case exert a force at the calculated center of mass of each body.



A Note on Rigid Loops

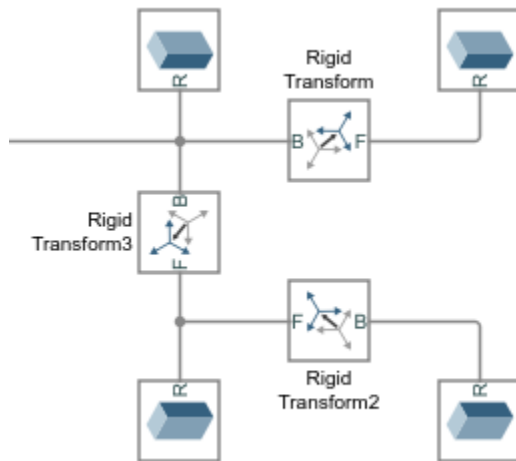
The blocks that comprise a body connect rigidly and, in the ideal case, in series. The result is a tree structure in which the path between any two frames is unique. It is technically possible, however, to rigidly connect blocks so that they form a rigid loop, a closed structure formed by connecting the open ends of a branch or tree.



A Rigid Transform Loop

Rigid transform loops are disallowed in a model. They contain redundant (and unnecessary) rigid connections and these could lead to preventable numerical errors if included in a model. If your

model contains a rigid kinematic loop, you must break that loop by removing of its redundant rigid connections. The figure shows an example.



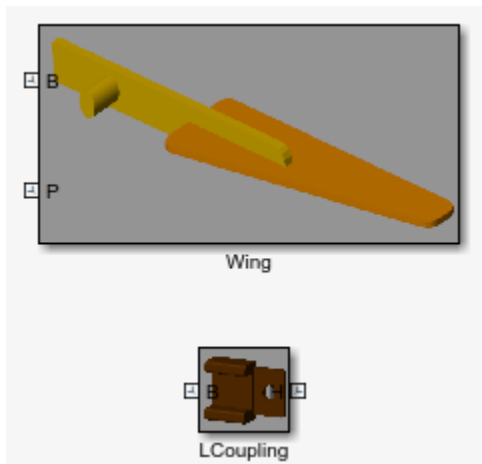
Bodies as Simulink Subsystems

It is common practice to enclose the blocks that belong to a body inside a Simulink Subsystem block. You can save the resulting Subsystem block in a Simulink library for later use in different models. The mask of the Subsystem block provides a place to define the variables and parameters that are common to the enclosed blocks—often some length, density, and color—without cluttering the workspace of a model.

You can model a body and convert it to a Simulink subsystem at any time prior to its use in a multibody model. For example, when modeling a piston engine, you can defer work on a piston body until you are ready to connect the piston to the engine housing or to the connecting rod. However, given that the shape and size of one body often depend on those of another, you should at least consider those attributes prior to focusing on assembly.

See It: A Typical Body Subsystem

At the MATLAB command prompt, enter `sm_cam_flapping_wing_lib`. A Simulink library opens with subsystem blocks representing two of the bodies used in the `sm_cam_flapping_wing` model. Each of the blocks has frame ports for connection to a model.



Double-click the subsystem block named **Peg**. A custom dialog box opens with the key parameters required to completely define this body. The values specified here are used in the Cylindrical Solid and Rigid Transform blocks that compose the subsystem block.

Dimensions	Wing	Material Properties
W - Link Width (cm):		
<input type="text" value="1"/>		<input type="button" value="⋮"/>
T - Link Thickness (cm):		
<input type="text" value="3"/>		<input type="button" value="⋮"/>
L - Link Length (cm):		
<input type="text" value="30"/>		<input type="button" value="⋮"/>
Dp - Peg distance (cm):		

Right-click the subsystem block and select **Mask > View Mask**. The Simulink Mask Editor opens with the parameters and code relevant to the wing body. The parameters and the MATLAB variables associated with them are defined in the **Parameters & Dialog** tab. The code used to generate the wing and spar shape profiles is defined in the **Initialization** tab.

<input type="text" value="311"/> #5	Lp - Peg length (cm):	Lp
<input type="text" value="311"/> #6	Rp - Peg Radius (cm):	Rp
<input type="text" value="311"/> #7	Rc - Chamfer radius (cm):	Rc

Right-click the subsystem block and select **Mask > Look Under Mask**. The block diagram corresponding to this subsystem opens up. Open the dialog boxes of some of the blocks and note the MATLAB variables used to define many of their parameters. These are the variables defined in the subsystem mask and specified in the subsystem block dialog box.

Geometry			
Radius	$Rp/2$	cm	▼
Length	$3*Rp$	cm	▼

See Also

More About

- “Compounding Body Elements” on page 1-15
- “Creating Connection Frames” on page 1-33
- “Representing Solid Geometry” on page 1-38
- “Representing Solid Inertia” on page 1-62

Compounding Body Elements

Compounding as a Modeling Strategy

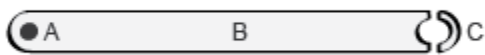
It is often simpler to specify the attributes of several simple solids than those of a single complex body. Compounding is a modeling strategy whereby you can model a body as a combination of simpler body elements. You can use compounding to obtain complex geometries and inertias that you cannot otherwise (or easily) specify. The `RightWing` body in the `sm_cam_flapping_wing` model, for example, is a product of this modeling strategy.



An Example of a Compound Body

Try It: Create a Compound Geometry

Combine three general Extruded Solid shapes using Rigid Transform blocks to specify the fixed spatial relationships shared by the solid reference frames. The result of this example is not merely a compound solid geometry—it is a compound body. You must use the deliberately incomplete model `smdoc_compound_link` that by default comes with your Simscape Multibody installation.



Explore the Compound Body Model

Start by exploring the `smdoc_compound_link` model and the geometry variables defined in its workspace:

- 1 At the MATLAB command prompt, enter the example model name, `smdoc_compound_link`. The model opens. In it are six unconnected blocks—three solid blocks, two Rigid Transform, and one Solver Configuration.

The solid blocks represent the elementary sections of a binary link and the Rigid Transform blocks the spatial relationships between the solid reference frames. The Solver Configuration block is required only for visualization in Mechanics Explorer.

- 2 In the **Modeling** tab, click **Model Explorer**. Model Explorer is a Simulink tool that you can use to explore your model workspace. All the relevant solid dimensions, including the general Extruded Solid cross-sections, are defined there.
- 3 In the **Model Hierarchy** pane, located on the left, expand the node named after your model and select the **Model Workspace** subnode. The **Model Workspace** pane opens on the right prefilled with several lines of MATLAB code.

```

% Body Geometry Parameters
l = 20;      % Hole-to-hole distance
w = 2;      % Link width
d = 1.2;    % Hole diameter
t = 1;      % Link thickness

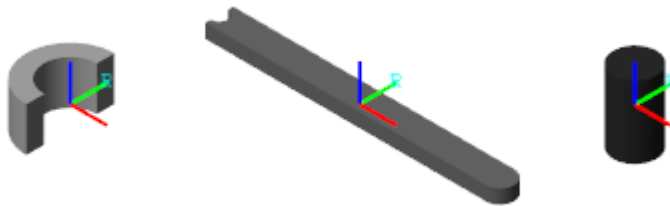
% Main Solid Cross-section:
A = linspace(-pi/2,pi/2)';
B = linspace(pi/2,-pi/2)';
csRight = [l/2+w/2*cos(A) w/2*sin(A)];
csLeft = [-l/2 w/2; -l/2 + d/2*cos(B) d/2*sin(B); -l/2 -w/2];
csMain = [csRight; csLeft];


% Hole Solid Cross-section:
C = linspace(pi/2,3*pi/2)';
D = linspace(3*pi/2,pi/2)';
csHole = [w/2*cos(C) w/2*sin(C);
d/2*cos(D) d/2*sin(D)];

```

This code defines the $[x, y]$ coordinates of the General Extruded Solid cross-sections. The cross-sections are parameterized in terms of the relevant solid dimensions, namely length, width, and hole diameter. Note the link dimensions specified in the code. The distance between the link holes (variable l), is 20 in what will later be units of cm. The link width (w) is 2 and the hole diameter (d) 1.2 in the same units.

- 4 Open each solid block dialog box. The visualization pane shows the solid geometry, derived partly from the code in the model workspace, corresponding to the respective block. Two of the solids are general Extruded Solid blocks, and one is a Cylindrical Solid block.

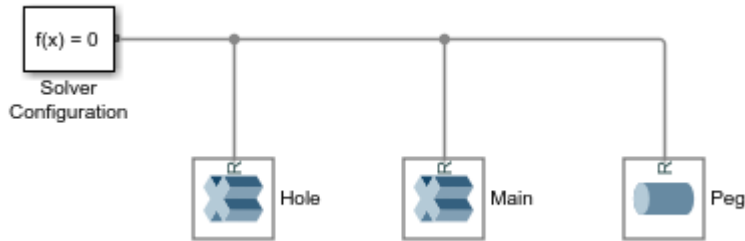


On the toolbar of the visualization pane, click the **Toggle visibility of frames**  button. The visualization pane displays the solid reference frame. The placement of a reference frame relative to a solid geometry becomes important when considering the rigid transforms that you must apply between the various solid reference frames.

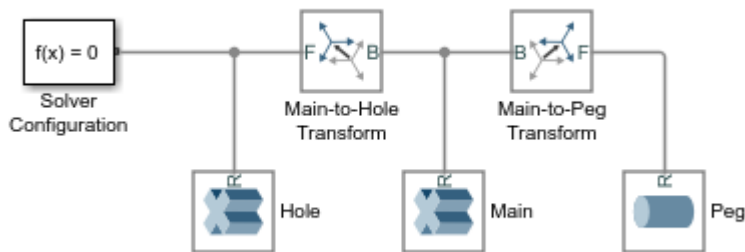
Combine the Solids Through Rigid Transforms

Complete the model by rigidly connecting the solids and specifying their spatial relationships:

- 1 Connect the solid blocks as shown in the figure. The solid reference frames are, for the moment only, coincident with each other.



- Drop the Rigid Transform blocks on the connection lines as shown in the figure. Simulink automatically connects the frame ports to the connection lines.



Pay special attention to the port positions—the **B** ports should both face the Extruded Solid block named **Main**. Flipping the port connections would change the relative placement of the solids in the final body.

- In the dialog box of the Rigid Transform block named **Main-to-Hole Transform**, specify the **Translation** parameters listed below. These parameters describe a translation of half the binary link length along the $-x$ axis of the base (**B**) frame—in this model held coincident with the reference (**B**) frame of the solid named **Main**.
 - Method:** Standard Axis
 - Axis:** $-X$
 - Offset:** $l/2$, units of cm
- In the dialog box of the Rigid Transform block named **Main-to-Peg Transform**, specify the **Translation** parameters listed below. These parameters describe a translation of half the binary link length along the $+x$ -axis and a translation equal to the binary link thickness along the $+z$ -axis of the base (**B**) frame.
 - Method:** Cartesian
 - Offset:** $[l/2 \ 0 \ t]$, units of m
- In the **Modeling** tab, select **Update Model**. Mechanics Explorer opens with a visualization of the binary link model. The body is compound—it comprises multiple solids—and can therefore be visualized in its entirety using Mechanics Explorer only. For emphasis, the solids are shown in different shades of gray.



A More Detailed Cross-Section Example

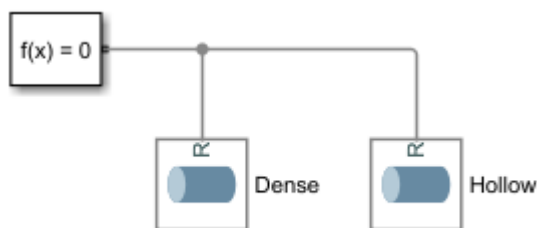
For an example showing how to specify a General Extruded Solid cross-section, see “Try It: Define a Simple Cross-Section” on page 1-46. The cross-section in that example is based on a similar, though not identical, model of a binary link. That link is treated as a simple body—one modeled as a single solid—with neither pegs nor holes. However, the strategy demonstrated there applies to other General Extruded Solid cross-sections as well. For an extension of that example showing how to include holes in a cross-section, see “Try It: Define a Cross-Section with Two Holes” on page 1-48.

Try It: Create a Compound Inertia

While it is more commonly used to represent complex geometries, compounding serves also to represent complex inertias. In particular, you can combine the inertia of a positive mass with the inertia of a negative mass, effectively subtracting one from the other.

Use this strategy to subtract the inertia associated with a bore from a cylindrical solid originally modeled without one. Represent the dense and hollow regions using Cylindrical Solid blocks. Set the cylinder length to 1 m and the radius to 0.25 m:

- 1 At the MATLAB command prompt, enter `smnew`. A new model based on the Simscape Multibody template opens up. The model contains commonly used blocks and is configured with suitable solver settings for multibody models.
- 2 Add two of the Cylindrical Solid blocks and connect them to the Solver Configuration blocks. The frame connection line between the blocks make their reference frames coincident in space. You can delete the remaining blocks.



- 3 In the dialog box of the leftmost Cylindrical Solid block, set the **Radius** parameter to 0.25 m, and the **Length** parameter to 1 m. Name this block **Dense**.
- 4 In the dialog box of the rightmost Cylindrical Solid block, set the **Radius** parameter to 0.20 m, and the **Length** parameter to 1 m. Name this block **Hollow**.
- 5 Set the **Inertia > Density** parameter of the **Hollow** block to the negative of the value used in the **Dense** block: -1000 kg/m^3 . The compound body represented by the Cylindrical Solid blocks now has the inertia of a hollow cylinder with a bore 0.2 m in radius.

- 6 Expand the **Inertia > Derived Values** node and click the **Update** button to display the inertia parameters of the Hollow solid. Do the same for the Dense solid. The mass and moments of inertia have opposite signs, as expected from the density inputs.

Derived Values		Update
Mass	-1.256637e+02	kg
Center of Mass	[+0.000000e+00, +0.000000e+00, +0.000000e...	m
Moments of Ine...	[-1.172861e+01, -1.172861e+01, -2.513274e+0...	kg*m^2
Products of Iner...	[-0.000000e+00, -0.000000e+00, -0.000000e+0...	kg*m^2

See Also

More About

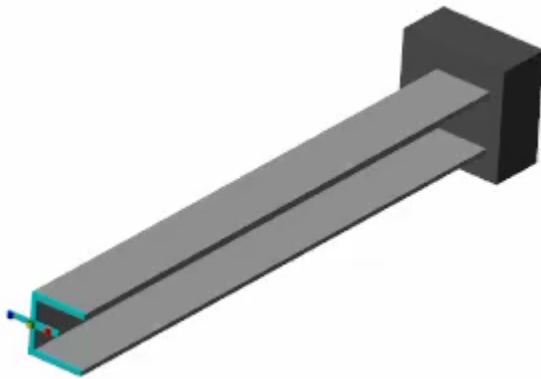
- “Modeling Bodies” on page 1-4
- “Representing Solid Geometry” on page 1-38
- “Representing Solid Inertia” on page 1-62

Overview of Flexible Beams

Flexible Beam Blocks

You can use flexible beam blocks in the Simscape Multibody to model slender bodies with constant cross-sections that can have small and linear elastic deformations. These deformations include extension, bending, and torsion. To use these blocks, in the **Library Browser**, click **Simscape > Multibody > Body Elements > Flexible Bodies > Beams**.

The following figure shows a flexible channel beam model. In this example, the beam undergoes both bending and torsion under an applied transverse point load. The degree to which the beam bends and twists varies with the point of application of the force in the plane of the cross-section. Enter `smdoc_flexible_cantilever_channel` at the MATLAB® command prompt to open the model.



Beam Geometries

The geometry of a beam is an extrusion of its cross-section. The general cross-sections, with or without holes, are supported by the General Flexible Beam block. Additionally, the beam cross-section can take many standard shapes, such as channel, angle, and hollow cylindrical. For beams with standard cross-sectional shapes, use the following flexible beam blocks:

- Flexible Angle Beam
- Flexible Channel Beam
- Flexible Cylindrical Beam (both solid and hollow)
- Flexible I Beam
- Flexible T Beam
- Flexible Rectangular Beam (both solid and hollow)

Connection Frames

Each beam has two connection frames labeled **A** and **B**. Each connection frame has a frame port on the block that can connect to another block. The connection frames are located at the ends of the

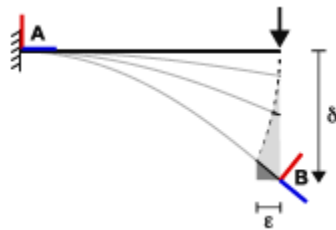
beam and fall on the z -axis of the local reference frame labeled **R**. The reference frame serves merely as an internal reference for the beam and has no frame port.

Deformation Models

In Simscape Multibody, all the flexible beams can have elastic bending, axial, and torsional deformations. The beams are assumed to be slender bodies whose length must far exceed its overall cross-sectional dimensions, and all the deformations should be linear and small.

The bending and axial deformations of a beam follow classical (Euler-Bernoulli) beam theory. The bending can be about any axis in the cross-sectional plane (xy -plane) of the beam. Cross-sectional slices are assumed to be rigid in-plane, to stay planar during deformation, and to always be perpendicular to the deformed neutral axis of the beam. The twisting of a beam derives from Saint-Venant torsion theory, and the cross-sectional slices are rigid in-plane but free to warp out-of-plane.

When one or more of these assumptions are not met, the result may be inaccurate. For example, in the figure, a cantilevered beam that is subjected to a transverse point load will get an inaccurate result when the bending deformation, δ , is large. During the bending, the free end of the beam moves downward perpendicularly instead of following the true physical path, which is indicated by the dotted trajectory. The discrepancy, ε , increases as the δ increases.



Material Properties

Flexible beams in Simscape Multibody are assumed to be made of a homogeneous, isotropic, and linearly elastic material. You can specify the material properties, such as density and Young's modulus in the **Stiffness and Inertia** section of the block dialog box. The beam cross-sectional properties, such as the axial, flexural, and torsional rigidities, are automatically calculated by the block using the material and geometry properties that you specify. To see the computed values, in the beam block dialog box, open **Stiffness and Inertia > Derived Values** and click the **Update** button.

Damping Methods

The beam blocks support two damping methods: uniform modal damping and proportional damping. The uniform modal damping method applies identical damping ratios to all the vibration modes of the beam. In the proportional damping method, the damping matrix $[C]$ is a linear combination of the mass matrix $[M]$ and the stiffness matrix $[K]$:

$$[C] = \alpha[M] + \beta[K],$$

where α and β are scalar coefficients.

Discretization

The **Number of Elements** parameter in the **Discretization** section of the beam block dialog box specifies the number of finite elements used to discretize the beam. You can select its value to obtain a good compromise between simulation accuracy, which may require more elements, and simulation speed, which requires fewer elements. Use the fewest elements needed to satisfy your accuracy requirements.

For bending deformations, the beam blocks use the cubic Hermite interpolation method to compute the displacement distributions throughout each element. The distributions of axial displacement and torsional rotation are obtained by linear interpolation method.

Simulation Performance

When using beam blocks in a model, several factors impact the accuracy and speed of the simulation performance. This section discusses the impact of the three most important factors: flexible beam usage, solver selection, and damping settings.

Even though using flexible beams can increase the accuracy of a multibody simulation, the flexible beams tend to slow it down by increasing the numerical stiffness and the number of degrees of freedom of the system. To speed up the simulation, you should use a rigid body whenever the deformation of the body is negligible. Moreover, the **Number of Elements** parameter in the **Discretization** section heavily impacts the performance of the simulation. For more information, see the “Discretization” on page 1-22 section.

The solver is critical to the performance of a multibody simulation. The stiff solvers, such as ode15s, ode23t, or daessc, tend to work better for systems with flexible beams due to the stiff nature of these systems. Additionally, solver tolerances and maximum order also impact the accuracy and speed of the simulation. For more information, see “Choose a Solver”.

Note All the solvers, except ode23t, provide some level of numerical dissipation, which can be helpful for modeling flexible multibody systems.

When modeling a flexible beam with little or no damping, undesirable high-frequency modes in the response can slow down the simulation if the solver does not already provide adequate numerical dissipation. In that case, adding a small amount of damping can improve the speed of the simulation without significantly affecting the accuracy of the model.


Deformation Under Gravity

Flexible beams in Simscape Multibody respond to gravity, but only that specified in the Mechanism Configuration block. The force due to a Gravitational Field block is ignored. If the frame network of which the flexible beam block is a part contains a Gravitational Field block, the body behaves as though in zero gravity. Using flexible body and Gravitational Field blocks in the same frame network causes **Diagnostic Viewer** to issue a compilation warning.

Note Modeling gravity with both the Mechanism Configuration and Gravitational Field blocks results in a compilation error.

Visualization

The dialog box of each flexible beam block contains a collapsible visualization pane. This pane provides instant visual feedback on the beam you are modeling. Use it to find and fix any issues with the cross-section, length, and color of the beam. You can examine the beam from different views by selecting a standard view or by rotating, panning, and zooming.

In the toolbar of the visualization pane, click the **Update Visualization** button  to view the latest changes to the beam. Click **Apply** or **OK** to commit any changes to the model.

Note You can point to any button to see its function.

Additionally, you can right-click the visualization pane for a context-sensitive menu. This menu provides additional options to change the background color, modify the view convention setting, and split the visualization pane into multiple windows that display different views of the beam.

Working with Frames

In this section...

“Role of Frames” on page 1-24

“Custom Solid Frames” on page 1-25

“What Are Frame Transforms?” on page 1-27

“Visualizing Frame Transforms” on page 1-27

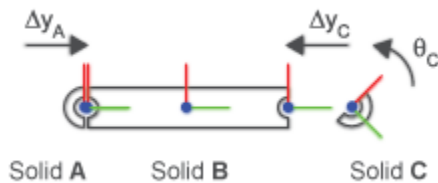
“Try It: Specify a Frame Transform” on page 1-28

Frames are axis triads that encode position and orientation data in a 3-D multibody model. Each triad consists of three perpendicular axes that intersect at an origin. The origin determines the frame position and the axes determine the frame orientation. The axes are color-coded, with the x-axis in red, the y-axis in green, and the z-axis in blue.



Role of Frames

Every solid component has one or more local frames to which it is rigidly attached. By positioning and orienting the component frames, you position and orient the components themselves. This is the role of frames in a model—to enable you to specify the spatial relationships between components.

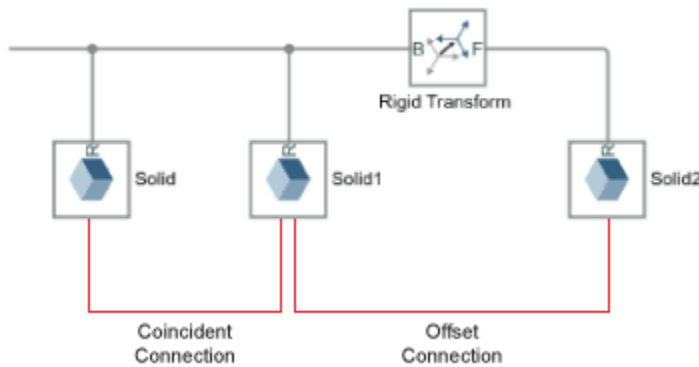


Working with Frames

A frame port identifies a local frame on a component. For example, the R frame port of a solid block identifies the local reference frame of a solid. Every block has one or more frame ports that you connect in order to locate the associated components in space. The figure shows the reference frame ports on several of the **Body Elements** blocks.

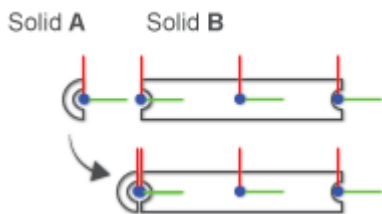


The connections between frame ports determine the spatial relationships between their frames. A direct frame connection line makes the connected frames coincident in space. A Rigid Transform block sets the rotational and translational offsets between the frames. The figure shows examples of coincident and offset frame connections.



A coincident relationship between solid frames does not, by itself, constitute a coincident relationship between solid geometries. The spatial arrangement of two solid geometries depends not only on the spatial arrangement of the respective reference frames, but also on how the geometries are defined relative to those frames.

If two geometries differ from each other, or if their positions and orientations relative to their reference frames differ from each other, then making the reference frames coincident will cause the solid geometries to be offset. In the figure, connecting the frame of Solid A to the left frame of Solid B joins the solids such that their geometries are offset from each other.



Custom Solid Frames

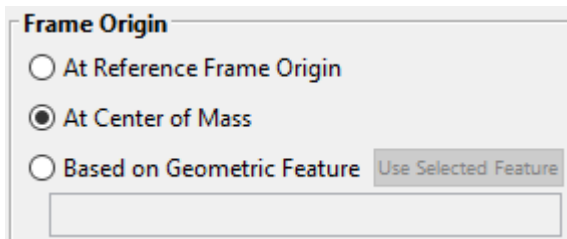
The Solid block provides a frame creation interface that you can use to create new, *custom*, frames. You can position and orient a custom frame using geometry features such as vertices, edges, and faces. More conveniently from an inertia standpoint, you can do the same using the center of mass and three principal axes of the solid.

Try It: Create a Custom Solid Frame

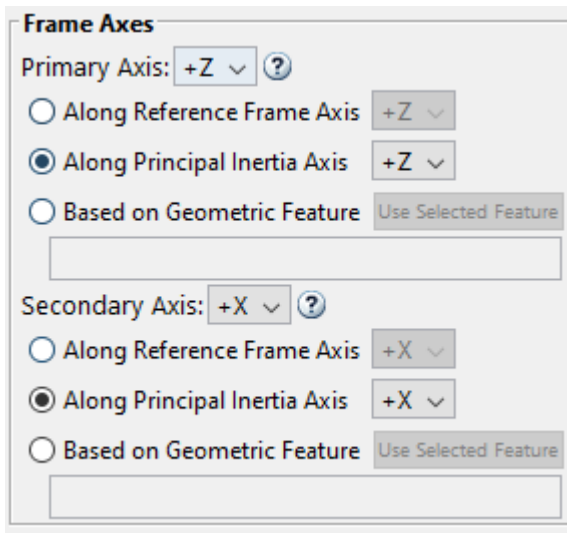
Create a custom frame using the frame creation interface of the File Solid block. Then, place the frame origin at the center of mass and align the frame axes with the principal axes of inertia. The result is a frame that coincides with the principal reference frame—one in which the inertia matrix is diagonal and the products of inertia are zero.

- 1 At the MATLAB command prompt, enter `smdoc_lbeam_inertia`. A model opens with a solid possessing the shape of an L-beam.

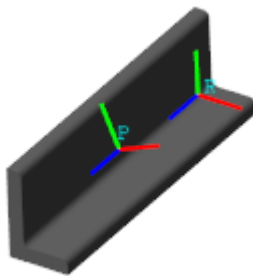
- 2 In the File Solid block dialog box, click the **Create Frame** button. The File Solid block dialog box switches to a frame creation view.
- 3 Change the **Frame Name** parameter to P (for “Principal Frame”). The visualization pane and the frame port use this label to identify your new frame.
- 4 Under **Frame Origin**, select the radio button labeled **At Center of Mass**.



- 5 Under **Frame Axes > Primary Axis** and **Frame Axes > Secondary Axis**, select the radio button labeled **Along Principal Inertia Axis**. Accept the default axis options (+Z and +X, respectively) and click **Save**. The block dialog box switches back to the main (parameters) view.

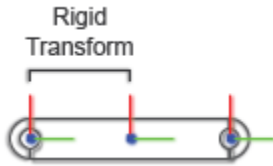


- 6 In the visualization toolbar, click the **Toggle visibility of frames** button. The visualization pane shows the frames of the solid, including your new custom frame, **P**.



What Are Frame Transforms?

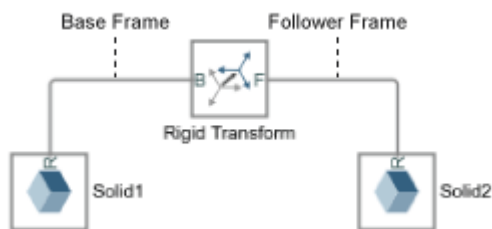
The rotational and translational offsets between frames are called transforms. If the transforms are constant through time, they are called rigid. Rigid transforms enable you to fix the relative positions and orientations of components in space, e.g., to assemble solids into bodies.



Working with Frame Transforms

You use the Rigid Transform block to specify a rotational, translational, or mixed rigid transform between frames. The transforms are directional. They set the rotation and translation of a frame known as follower relative to a frame known as base.

The frame port labels on the Rigid Transform block identify the base and follower frames. The frame connected to port B serves as base. The frame connected to port F serves as follower. Reversing the port connections reverses the direction in which the frame transform is applied.

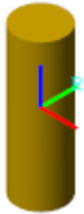


You can specify a transform using different methods. For rotational transforms, these include axis-angle pairs, rotation matrices, and rotation sequences. For translational transforms, they include translational offset vectors defined in Cartesian or cylindrical coordinate systems.

If the rotational and translational transforms are both zero, the connected frames are coincident in space. This relationship is known as *identity* and it is equivalent to a direct frame connection line between frame ports—i.e., one without a Rigid Transform block.

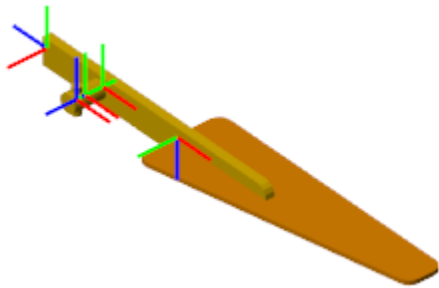
Visualizing Frame Transforms

You can visualize frames and examine the transforms between frames using the Solid block visualization pane or Mechanics Explorer. Use the Solid block visualization pane to examine the frames of a single solid element. Click the **Toggle visibility of frames** button in the visualization toolstrip to show all the solid frames.



A Frame on a Solid

Use Mechanics Explorer to visualize the frames of more than a single solid element—e.g., in compound bodies, multibody subsystems, or complete multibody models. Select **View > Show Frames** in the Mechanics Explorer menu to show all frames. Select a node from the tree view pane to show only those frames belonging to the selected component.



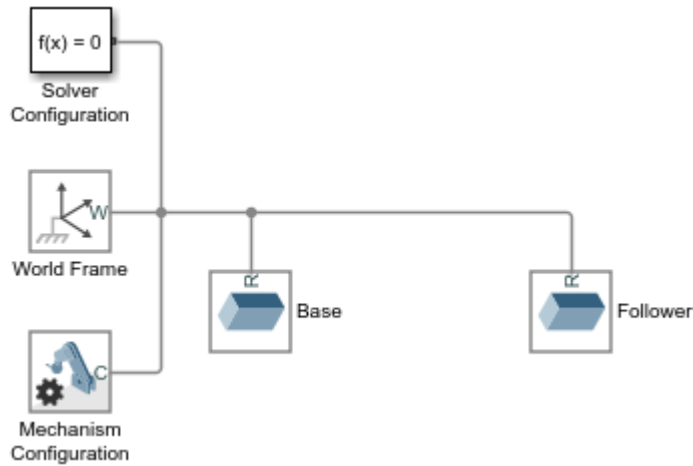
Frames on a Body

Try It: Specify a Frame Transform

This example shows how to offset two solids relative to each other by specifying a frame transform between the solid reference frames. It shows three types of transformations: translation, rotation, and combined transformation.

Create a Model with Two Solids

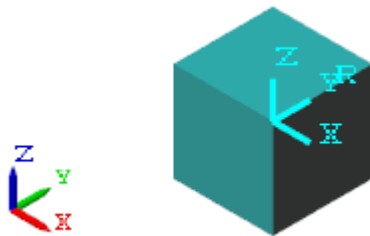
- 1 At the MATLAB command line, enter `smnew`. A Simscape Multibody model template with commonly used blocks opens.
- 2 Delete the Simulink-PS Converter, PS-Simulink Converter, and Scope blocks because they are not used in this example.
- 3 Show the names of the Rigid Transform and Brick Solid blocks. Right-click the blocks and select **Format > Show Block Name > On**.
- 4 Make a copy of the Brick Solid block and rename the Brick Solid blocks to Base and Follower.
- 5 Connect the remaining blocks like the following figure.



- 6 Run the model. **Mechanics Explorer** opens with a model visualization. In the **Mechanics Explorer**, click the **Isometric view** button.



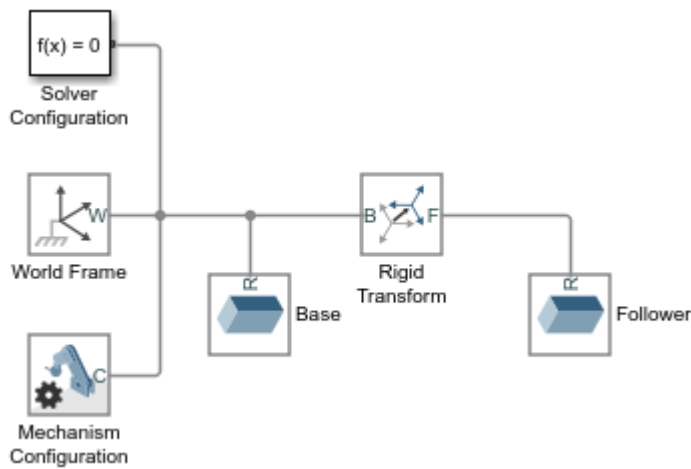
- 7 In the tree view pane, click the **Base** and **Follower** nodes. The visualization pane shows the reference frames of the two Brick Solid blocks, which shows they are coincident in space.



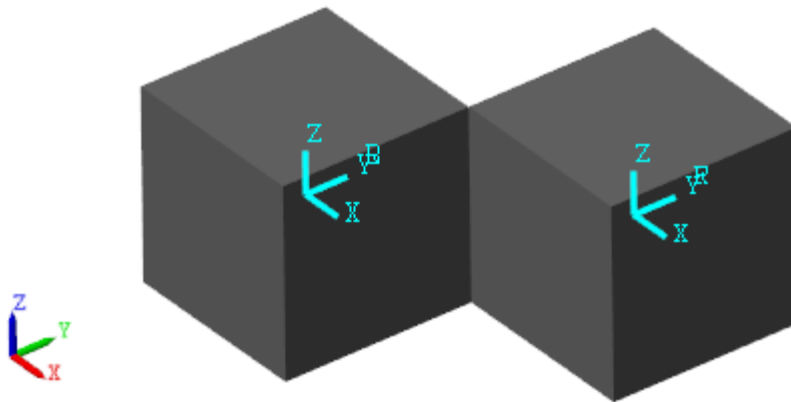
Apply a Translation

This section shows how to apply a translation to a follower frame by using the Rigid Transform block.

- 1 Connect the Rigid Transform block between the two Brick Solid blocks.



- 2 Double-click the Rigid Transform block. In its dialog box, set:
 - **Translation > Method** to Cartesian.
 - **Translation > Offset** to $[1 \ 1 \ 0]$ m. The array elements are the translation offsets along the base frame x , y , and z -axes.
- 3 Click **OK** to save the settings and close the Rigid Transform dialog box.
- 4 Click **Update diagram** button in the Mechanics Explorer to update the model.
- 5 In the tree view pane, click the Rigid Transform node to show the Base and Follower frames.

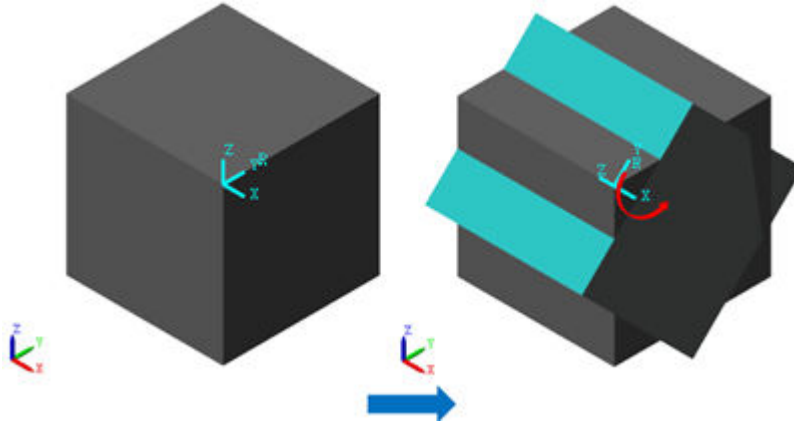


Apply a Rotation

This section shows how to apply a rotation about the x -axis to the follower frame by using the Rigid Transform block.

- 1 In the Rigid Transform block dialog box, set:
 - **Translation > Method** to None. No translation is specified here.
 - **Rotation > Method** to Standard Axis.
 - **Rotation > Axis** to $+X$. x -axis is the axis of rotation.
 - **Rotation > Angle** to 45 deg.

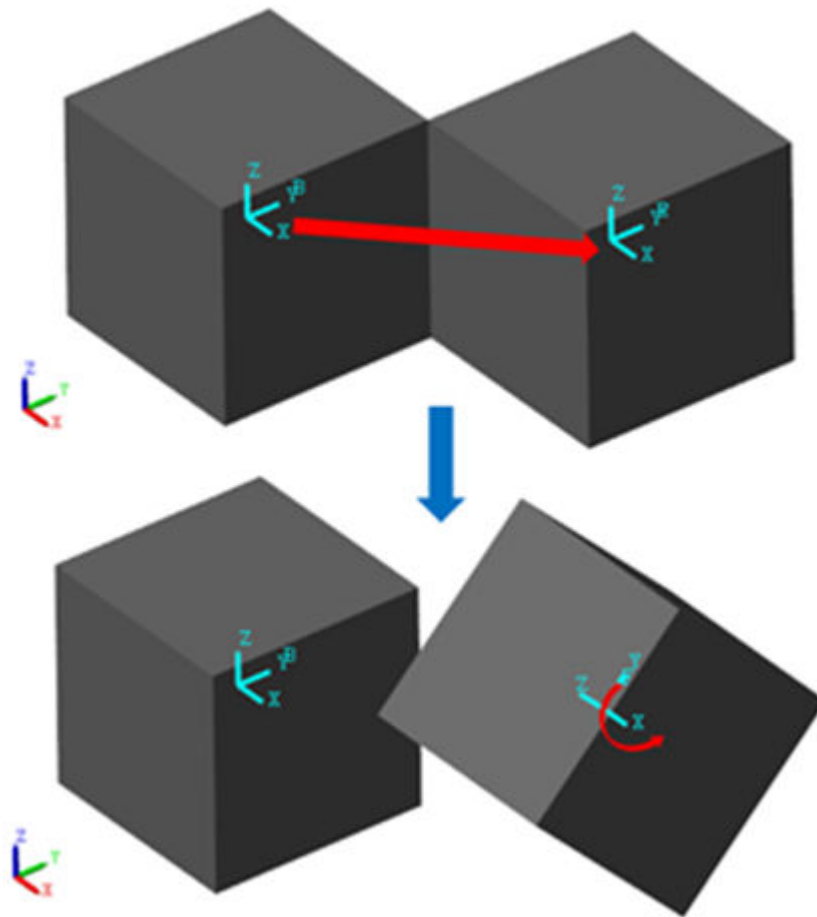
- 2 Click **OK** and update the block diagram.
- 3 In the tree view pane, click the **Rigid Transform** node. The following figure shows the process and the final result of the rotation. In the final result, the Follower brick solid is highlighted.



Apply a Combined Transformation

This section shows how to apply a combined transformation, which includes a translation and a rotation to a follower frame by using the Rigid Transform block.

- 1 In the Rigid Transform block dialog box, set:
 - **Translation > Method** to Cartesian.
 - **Translation > Offset** to [1 1 0].
 - **Rotation > Method** to Standard Axis.
 - **Rotation > Axis** to +X. x-axis is the axis of rotation.
 - **Rotation > Angle** to 45 deg.
- 2 Click **OK** and update the block diagram.
- 3 In the tree view pane, click the **Rigid Transform** node. The following figure shows the process and the final result of the combined transformation.



When both rotational and translational transformations are specified in a Rigid Transform block, the block always applies the translation to the follower frame first. The translations describe how the follower frame is moved relative to its base frame. Any rotations about the base frame axes are always in respect to the axes of the translated base frame.

See Also

More About

- “Modeling Bodies” on page 1-4
- “Creating Connection Frames” on page 1-33
- “Visualize Simscape Multibody Frames” on page 5-24

Creating Connection Frames

Frames as a Connection Points

The frames of bodies provide the connection points for the joints and constraints in your model. They determine also the relative orientations of those joints and constraints, and therefore the directions of motion that are allowed during simulation. To successfully connect bodies through joints and constraints, you must create suitable connection frames, and this is a task that is best done when modeling the bodies themselves.

Creating and Transforming Frames

You can add frames directly to solids using the frame creation interface of the solid blocks. This interface enables you to define the position and orientation of a frame interactively, in terms of key geometry features, such as vertices, edges, and faces, or in terms of key inertia features, such as the center of mass and the principal axes of inertia. Frames that you create are known as custom and appear as frame ports on the solid block to which they belong. The figure shows a Brick Solid block with two custom frame ports labeled **F1** and **F2**.



You can also create frames using the Rigid Transform block. This block enables you to define the position and orientation of a frame numerically, in terms of rotation and translation transforms. You can use a variety of transform parameterizations, including rotation matrices and rotation sequences in the case of rotation transforms, and Cartesian and cylindrical offset coordinates in the case of translation transforms. The figure shows a new frame (**F**) created using a Rigid Transform block from an existing solid frame.



Frames that you create using Rigid Transform blocks are independent of any specific solid features. You can place them anywhere relative to another frame as long as you can determine the transform required to obtain that placement. When placing a Rigid Transform block between two frame ports, you can more aptly think of the block as a means to offset frames that already exist. The figure shows an offset specified through a Rigid Transform block placed between two existing frames.

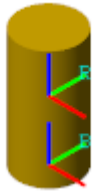


For more information about frames and transforms, see “Working with Frames” on page 1-24.

See It: Frames in a Typical Body

At the MATLAB command prompt, enter `sm_cam_flapping_wing`. A model of a flapping wing mechanism opens up. Look inside the mask of the body subsystem named `RightWing`. Note that two of the solid blocks, `Peg` and `WingLink`, each have two frame ports. One of the ports identifies a custom frame created using the Cylindrical Solid block. Explore one of the custom frames:

- 1 Open the dialog box of the Cylindrical Solid block named `Peg`.
- 2 In the visualization toolstrip, click the frame button. The visualization pane updates to show the reference and custom frames of the solid.



- 3 In the **Properties** section of the dialog box, expand the **Frames** node and click the **Edit** button. The frame creation interface opens with the current frame definition:
 - The frame origin has been placed at the center of the bottom surface of the cylinder.
 - The frame axes have been left in alignment with those of the local reference frame.
 - The name of the frame matches the label of the corresponding frame port (**B**).

Note also that the frame connection lines between some of the solid blocks contain Rigid Transform blocks, named `CPXform` and `WingXForm`. These blocks specify the rotational and translational offsets between the solid frames that they connect. Explore the transforms specified in one of the blocks:

- 1 Open the property inspector of the Rigid Transform block named `WingXForm`. Note that the rotation transform is set by aligning two axes of the follower frame relative to two axes of the base frame.

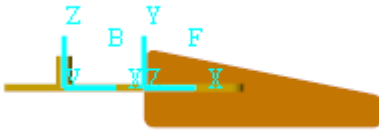
▼ Rotation

Method	Aligned Axes ▼
▼ Pair 1	
Follower	+X ▼
Base	+Z ▼
▼ Pair 2	
Follower	+X ▼
Base	+X ▼

- 2 Expand the **Translation** parameters. Note that the translational offset is specified along the axis (of the base frame). The translational offset is parameterized in terms of a MATLAB variable, `0w`, whose value you specify in the `WingLink` subsystem block.

Translation			
Method	Standard Axis		
Axis	+X		
Offset	Ow	cm	Compile-time

- Update the block diagram. Mechanics Explorer opens with a static visualization of the flapping wing model. In the tree view pane, expand the `RightWing` node and click the `WingXForm` node to highlight the frames belonging to the Rigid Transform block.

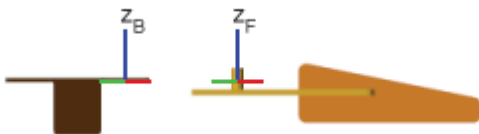


Planning Connection Frames

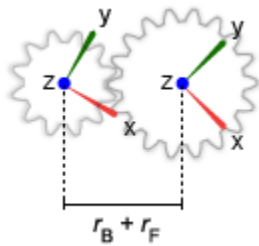
You must consider the target of a connection frame—the specific joint or constraint—when defining its placement on a body. Joints and constraints often impose special assembly requirements on the frames that they connect. These requirements impact the proper placement of a connection frame. You can find them in the reference page of the joint or constraint block.

If the target is a joint, take note of its degrees of freedom—the types of motion allowed between the joint connection frames—and of the frame axes to which they correspond. For example, the Revolute Joint block provides one rotational degree of freedom about the common z -axis of the connection frames on the base and follower bodies.

To connect two bodies through a Revolute Joint block, you must then place their joint connection frames so that their z -axes each align with the desired rotation axis on the respective body. The figure shows an example: the `Housing` and `RightWing` bodies of the `sm_cam_flapping_wing` model with connection frames properly positioned for a Revolute Joint block.



If the target of a connection frame is a constraint, such as that characteristic of gears in mesh, take note of the connection frame placements required for assembly. For example, the Common Gear Constraint block requires that the connection frames on the base and follower bodies be apart by a distance equal to the sum of their pitch radii when the meshing type is set to `External`. It requires also that the z -axes be parallel, and that the x - and y -axes of one be coplanar with those of the other.



The assembly requirements of constraint blocks specify how the remainder of the model must hold the connection frames, partly through body definitions and partly through other joints and constraints, for the constraint to apply without error. To see how to place connection frames on gear bodies for assembly via gear constraint blocks, see “Assemble a Gear Model” on page 2-32.

Addressing Assembly Errors

Closed-loop models, such as those of four-bar and crank-slider mechanisms, are limited in the positions and orientations that their constituent bodies can take. If the placement of a connection frame renders a joint or constraint incompatible with any other in the loop—that is, if successfully connecting one requires that another break—then assembly fails.

This is the case in a planar four-bar mechanism, for example, when the distance between the connection frames on any one link exceeds the sum of the equivalent distances in the remaining links—or when the orientations of the connection frames force the rotation axes of the joints into anything but a parallel alignment.

You can prevent many assembly errors by carefully defining the connection frames on a body with its future connections in mind. Consider both the assembly requirements specific to a joint or constraint block and the kinematic constraints imposed by the remainder of the model. In general, if you encounter an assembly failure, you must:

- 1 Identify the unassembled joint or constraint. Use the Simscape Multibody Model Report (**Tools > Model Report** in the Mechanics Explorer menu bar).

Name	Status
CPJ	✘
Pz	✘
Rz	✘
CamJoint	●
Rz	●
LFJ	●
Pz	●

- 2 Examine the corresponding connection frames. Use Mechanics Explorer to visualize these frames. Click the name of the unassembled joint or constraint in the tree view pane to highlight its frames in the visualization pane. Compare the placement of those frames to the assembly requirements of the joint or constraint.



- 3 Transform the connection frames to satisfy the joint or constraint assembly requirements. You can use the Rigid Transform block to apply the required rotation and translation transforms. If a connection frame is a custom frame belonging to a solid block, you can use that block instead to edit the frame definition.

For an example showing how to resolve an assembly failure caused by an improperly placed connection frame, see “Troubleshoot an Assembly Error” on page 2-21.

See Also

More About

- “Modeling Bodies” on page 1-4
- “Working with Frames” on page 1-24
- “Creating Custom Solid Frames” on page 1-82
- “Visualize Simscape Multibody Frames” on page 5-24

Representing Solid Geometry

In this section...

“Geometry in a Model” on page 1-38

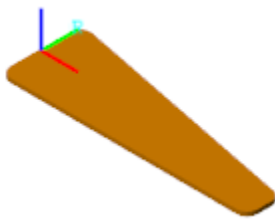
“Preset Solid Shapes” on page 1-40

“Imported Solid Shapes” on page 1-41

“Compound Solid Shapes” on page 1-42

Geometry in a Model

Geometry is a key attribute of solids and of the bodies they comprise. It features in the solid visualizations provided by Solid blocks as visual aides during modeling. It features also in the multibody visualizations displayed in Mechanics Explorer following model assembly and during simulation. This is one purpose of solid geometry: to enable visualization for an entire modeling workflow, from the conception of a single solid to the simulation of a complete multibody model.



Geometry of a Body Element

Solid geometry serves a second, less visible, purpose: to simplify the specification of inertia in the solid blocks. The bulk of solid inertia parameters are readily computed if both geometry and mass, or, alternatively, mass density, are known. The solid blocks provide an inertia parameterization, **Calculate from Geometry**, that performs these calculations for you. You specify the solid geometry and a measure of its mass; the block carries out the required numerical integrations to obtain the remaining inertia parameters—the moments of inertia, products of inertia, and center of mass.

Inertia	
Type	Custom
Mass	Calculate from Geometry
Center of Mass	Point Mass
Moments of Iner...	Custom
Products of Inertia	[1 1 1] kg*m^2

Geometry in Body Elements Blocks

Solid geometry differs in a practical way from frames and inertia. The latter are attributes that you can model in isolation using blocks such as Rigid Transform and Inertia. There is no equivalent, dedicated block for solid geometry. The Graphic and Spline blocks represent geometries—and provide a visualization means for those geometries—but neither is an adequate replacement for an actual solid geometry.

The Graphic block merely adds a marker to a frame, typically as a means of highlighting that frame. The Spline block adds a plane or space curve largely intended for use with the Point on Curve Constraint block. If you want to visualize solids and bodies, or benefit from the automatic inertia calculations that solid geometry enables, you must use a solid block.

Try It: Specify a Simple Cylindrical Shape

Use the Cylindrical Solid block to model a body with a simple preset shape—a cylinder with a radius of 5 cm and a length of 20 cm. Visualize the solid in the visualization pane of the Cylindrical Solid block. Ignore the relative placement of the solid in the (incomplete) model.

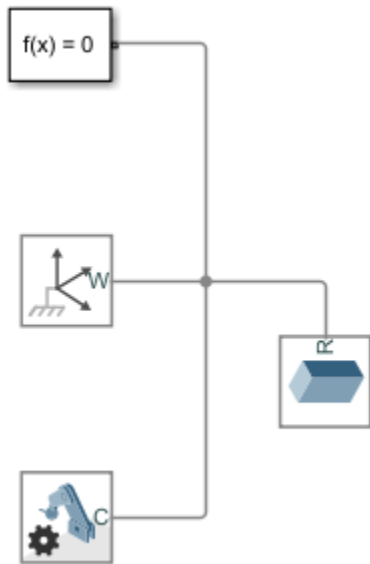
- 1 Add a Cylindrical Solid block to a new Simulink model and open the block dialog box. Note the **Geometry** parameters section, which by default specifies a cylinder shape 1 m in side.
- 2 In the **Radius** parameter line, enter a value of 5 and select units of cm. You can select your units from the dropdown list or enter them manually.
- 3 In the **Length** parameter line, enter a value of 20 and again select units of cm. Note the warning in the visualization pane urging you to update the solid visualization.
- 4 In the visualization toolstrip, click the **Update Visualization** button. The visualization pane refreshes with the new solid geometry but, due to its small dimensions, it is barely visible. Click the **Fit to View** button to optimize the zoom level. Ensure that the solid geometry is as expected.



- 5 Expand the **Inertia** parameters section and take note of the **Type** parameter setting. The automatic calculation of inertia properties from geometry is by default enabled. To complete the model of your solid, you need only ensure that its mass or mass density is set to the correct value. Click **OK** to accept the current solid settings.

Positioning and Orienting a Solid in a Model

If a solid block is unconnected, the relative placement of that solid is undefined. To resolve the solid pose—its position and orientation—in a model, you must connect the reference frame port (**B**) or, if you prefer, a custom frame port, belonging to the solid block. For example, connecting the **R** port to the **W** port of a World Frame block would align the solid so that its reference frame is coincident with the world frame. The figure shows such a connection



Specifying spatial relationships such as this is key to modeling in the Simscape Multibody environment. You can rotate and translate two frames with respect to one another by applying operations called *rigid transforms* between those frames. To learn more about frames and transforms, see “Working with Frames” on page 1-24.

For ease of modeling, the solid blocks provide a frame creation interface. You can use this interface to append and align new frames to select geometry features, such as vertices, edges, faces, and volumes. To learn how to create frames using this interface, see “Creating Custom Solid Frames” on page 1-82.

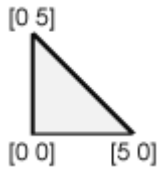
Preset Solid Shapes

Solid blocks provides a sizeable array of preset shapes—those with simple parameterizations featuring readily accessible parameters, such as **Radius** and **Length**, as inputs. Preset shapes make it possible to quickly model spherical, cylindrical, and prismatic solids, among others. For greater versatility, the preset shapes include the Extruded Solid block and Revolved Solid block—shapes whose cross-sections, be they along or about an axis, you can modify. To learn more about these shapes, see “Modeling Extrusions and Revolutions” on page 1-44.

Try It: Specify a Simple Revolved Shape

Use the Revolved Solid block to model a solid of revolution—a cone with a height of 5 ft and a base radius also of 5 ft. Visualize the solid in the visualization pane of the Revolved Solid block. Ignore the relative placement of the solid in the (still incomplete) model.

- 1 Add a Revolution Solid block to a Simulink model.
- 2 In the **Cross-Section** parameter line, enter the coordinate matrix $\begin{bmatrix} 0 & 0; & 5 & 0; & 0 & 5 \end{bmatrix}$ and select units of ft. Each matrix row provides an $[x \ z]$ coordinate pair, specified in that order, for a cross-section point.



- 3 Click the **Update Visualization** button and the **Fit to View** button. Ensure that the solid geometry is as expected. Click **OK** to accept the new solid geometry and close the block dialog box.



Specifying the Solid Cross-Sections

The Revolved Solid block generates the revolved shape by sweeping the specified xz cross-section about the z -axis. To consistently generate a valid shape without errors, the Revolved Solid block enforces a few rules. Foremost among these is the requirement that, as you proceed from one point in the coordinate matrix to the next, the solid region lie to your left and the empty (or hollow) region to your right. The same rule applies to extruded shapes, with one distinction: the cross-section coordinates are (x, y) pairs and the cross-section lies in the xy plane.

Imported Solid Shapes

Alternatively, you can use a File Solid block to import a solid from an external geometry file. File Solid blocks enable you to create solids with complex geometries. Currently, the File Solid block supports STEP (also referred to as STP) and STL formats.

The STEP format is recommended as it leads to what are generally smaller files than equivalent STL geometries. STEP is also the only of the two formats that allows for automatic inertia calculation from geometry. You must explicitly specify the moments of inertia, products of inertia, and center of mass of the solid when importing an STL geometry.

Note that very large files may load slowly and delay the usually fast model update step (in the **Modeling** tab, click **Update Model**). The size of a STEP or STL file depends to an extent on the application used to generate the file. You can, in some cases, reduce size by using a different application to export your solid geometry.

Try It: Import a STEP Geometry File

Use the File Solid block to import a detailed bevel gear geometry. The gear geometry was created in CAD software and subsequently exported in STEP format. Visualize the solid in the visualization pane of the Solid block and ignore the relative placement of the solid in the model.

- 1 Add a File Solid block to a Simulink model.

- 2 The **Geometry** parameters section updates to show the required file import properties—**File Type**, **File Name**, and, for STL files only, **Units**.
- 3 From the **File Type** dropdown list, select STEP. This is the recommended geometry file type. STEP files are generally smaller than their STL counterparts and enable the automatic calculation from geometry.
- 4 In the **File Name** parameter field, enter `bevel_c.step`. This file name corresponds to an example STEP geometry that is by default on your MATLAB path. If you experience any issues, you can enter the file path:

```
matlabroot/toolbox/phymod\sm\smdemos\doc\bevel_gear\bevel_c.step
```

where `matlabroot` is the root folder of your MATLAB installation. If you are unsure of the location of your root folder, at the MATLAB command prompt, enter `matlabroot`.

- 5 Click the **Update Visualization** button and then the **Fit to View** button. Ensure that the solid geometry is as expected. Click **OK** to accept the new solid geometry and close the block dialog box.



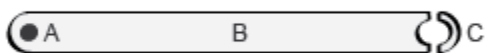
Obtaining the Solid Geometry Files

You can obtain a STEP or STL geometry file from a CAD model. Most CAD applications enable you to export your part geometries in these (among other) formats. If you are adept at using a CAD application, or have the support of someone who is, you can create a detailed solid geometry in CAD, export it in a STEP or STL file, and import the final geometry file into a File Solid block.

If you lack a license to a professional CAD application, open-source software such as FreeCAD may provide a suitable alternative. Onshape, a professional, full-cloud CAD application, provides free subscription plans. This tool has the advantage of allowing you to import complete multibody assemblies into the Simscape Multibody environment using the `smexporttonshape` function. For more information, see “Onshape Import” on page 6-17.

Compound Solid Shapes

If you cannot obtain a STEP or STL file with the desired solid geometry, you can still approximate that geometry—by combining simpler preset shapes into a larger, compound, shape. You must use multiple Solid blocks—one for each preset solid shape. Often, you must also use Rigid Transform blocks, to specify the spatial relationships that exist between the solid reference frames. The figure shows a solid geometry that you can model as a compound shape—a binary link with a hole section (labeled **A**), a main section (**B**), and a peg section (**C**).



For an example showing how to model this compound body, see “Try It: Create a Compound Geometry” on page 1-15.

See Also

More About

- “Modeling Extrusions and Revolutions” on page 1-44
- “Representing Solid Inertia” on page 1-62
- “Manipulate the Color of a Solid” on page 1-90
- “Visualize a Model and Its Components” on page 1-58

Modeling Extrusions and Revolutions

In this section...

“Extrusions and Revolutions” on page 1-44

“The Cross-Section Profiles” on page 1-45

“Cross-Sections with Holes” on page 1-47

“From Cross-Sections to Solids” on page 1-49

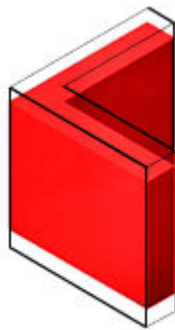
Extrusions and Revolutions

For solids with custom cross-sections, the Extruded Solid and Revolved Solid blocks enable you to create truer solid representations than simpler shapes such as Brick Solid, Cylindrical Solid, and Spherical Solid often allow. Use them when modeling solids that have arbitrary yet constant cross-sections along or about an axis. The Extruded Solid block can either be a **General** extrusion or a **Regular** extrusion.

What is the Extruded Solid Block?

A general extrusion is a linear sweep of a custom cross-section along an axis that is normal to the cross-section plane. The sweep spans the length specified in the Extruded Solid block dialog box. The cross-section can have an arbitrary outline and one or more hollow sections—though the rules for specifying cross-sections differ slightly when holes are present.

General extrusion examples include straight beams, plates, spars, struts, and rods. The figure shows an angle beam, a general extrusion whose cross-section consists of two thin rectangles arranged in an L shape. The Extruded Solid block sweeps the L shape linearly out of the cross-section plane to obtain the final beam geometry.



General Extruded Solid Block as a Linear Sweep

A Note on Extruded Solid Shapes

The Extruded Solid block provides a second extrusion shape, named **Regular** extrusion. This shape is a simpler version of **General** extrusion and is suited only for solids whose cross-sections are regular polygons—those with sides of the same length. The cross-section outline is fixed by the number of sides of the polygon and it cannot contain holes.

What is the Revolution Solid block?

Use the Revolved Solid block to make an angular sweep of a cross-section about an axis that lies on the same plane as the cross-section. The sweep can span a full revolution or a lesser angle between 0 and 360 degrees. As with extruded shapes, the cross-section can have an arbitrary outline with or without holes.

Revolved examples include cones, domes, pistons, gear shafts, and pipe bends. The figure shows a cylindrical peg, a Revolved Solid whose cross-section, like the angle beam, consists of two thin rectangles arranged in an L shape. The Revolved Solid block sweeps the L shape about an axis lying on the cross-section plane to obtain the final peg geometry.



Revolved Solid as an Angular Sweep

The Cross-Section Profiles

You specify the cross-sections numerically, as MATLAB matrices populated with the coordinate pairs of select cross-section points. Each matrix row provides a coordinate pair for one point. There is no upper bound on the number of rows of a coordinate matrix, but a minimum of three is required to completely define a closed shape.

```
% Coordinate Matrix Example:
% Square Cross-Section with Center at [0, 0] and Side Length 2
[
-1 -1; % Lower left corner
1 -1; % Lower right corner
1 1; % Upper right corner
-1 1; % Upper left corner
]
```

The coordinate pairs are treated as (x, y) values in the case of extruded shapes and as (x, z) values—specified in that order—in the case of revolved shapes. The coordinates are resolved in the reference frame of the Extruded Solid or Revolved Solid block, with the $(0, 0)$ pair coinciding with the origin of that frame. It is common practice to parameterized the coordinates in terms of MATLAB variables associated with key solid dimensions—for example, radius or length.

From Coordinates to Cross-Sections

The coordinate pairs connect sequentially in the order implicit in the coordinate matrix. The connections are by means of straight line segments. The result is a closed polyline that separates the region to be filled with material (the solid part) from the region to be left hollow (any holes that might be present and the empty surroundings).

The boundary between the two regions is such that, as you proceed along the polyline from one point to the next, the solid region lies to your left and the hollow region to your right. The first and last coordinate pairs are often the same, but if they are not, a connection line is inserted between them to ensure that the cross-section is in fact closed. The animated figure shows the drawing of a binary link cross-section without holes.



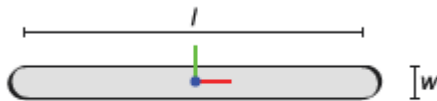
Note that the cross-section is invalid if at any point the polyline crosses itself. However, it is okay for two line segments to be arbitrarily close or even coincident with each other. In fact, you can exploit this property to specify a cross-section that has one or more holes.

A Special Constraint for Revolutions

Revolved Solid coordinate matrices are subject to a special constraint: x -coordinates cannot be negative. This rule follows partly from the revolution axis (z) used by the Revolved Solid block. When sweeping the cross-section, any areas to the left of this axis (those with negative x -coordinates) become overlapped with those to the right (positive x), resulting in an unexpected solid geometry. To prevent this issue, an error is issued if a Revolved Solid cross-section is found with negative x -coordinates.

Try It: Define a Simple Cross-Section

Consider the cross-section shown in the figure. This cross-section belongs to a binary link with round ends and no holes. Parameterize the cross-section in terms of the dimensions shown and specify it in the form of a coordinate matrix.



Binary Link Cross-Section (No Holes)

Start by opening a new MATLAB script and save it in a convenient location under the name `modelParams`. Add two variables for the dimensions shown in the figure, length (l) and width (w). Set the length to 20 and the width to 2 (in what will later be units of cm).

```
l = 20;
w = 2;
```

Define the round ends as semicircles. First, generate two arrays with the angular spans of the left and right ends. These arrays enable you to parameterize the (x , y) coordinates using simple trigonometric expressions. Each array has five points, but for smoother shapes you can specify more. The transpose symbol (`'`) ensures that A and B are column arrays.

```
A = linspace(-pi/2, pi/2, 5)';
B = linspace(pi/2, 3*pi/2, 5)';
```


Define the coordinate matrices of the right end (`csRight`) and left end (`csLeft`). The first column of each matrix corresponds to the x -coordinate. The second column corresponds to the y -coordinate. The x -coordinates of the two ends are offset in opposite directions by $l/2$.

```
csRight = [l/2 + w/2*cos(A) w/2*sin(A)];
csLeft = [-l/2 + w/2*cos(B) w/2*sin(B)];
```

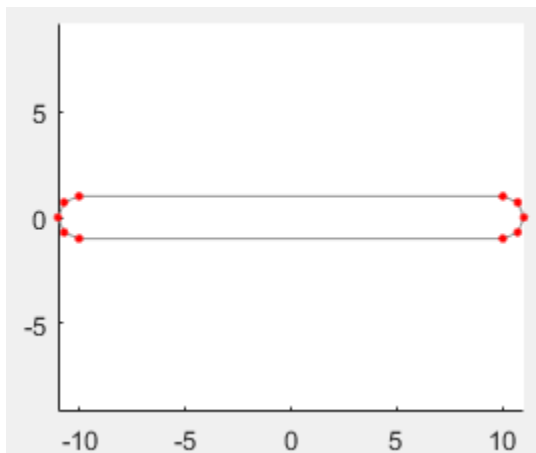
Combine the coordinate matrices into a single matrix named `cs`. This is the matrix that you must specify in the **Cross-Section** parameter of the Revolved Solid block. Note that the straight segments of the cross-section are automatically generated when the end points of the semicircles are connected.

```
cs = [csRight; csLeft];
```

You can visualize the cross-section outline using the MATLAB `plot` command. Enter the code shown below at the MATLAB command prompt.

```
figure; hold on; axis equal;
plot(cs(:,1), cs(:,2), 'Color', [0.6 0.6 0.6], 'Marker', '.', ...
      'MarkerSize', 9, 'MarkerEdgeColor', [1 0 0]);
```

The plot shows the cross-section of the binary link. The points in the coordinate matrix are shown as red dots. The resulting cross-section outline is shown as a light gray line. Notice that the end sections each comprise five points—the number specified in the angular span arrays.



Cross-Sections with Holes

The coordinate matrix should always represent a single continuous path. This rule works well when specifying a cross-section without holes but it demands extra care when holes exist. Because the outline of a hole is not contiguous with the outline of the cross-section, you must now add a thin cut between the two. The cut enables you to traverse the cross-section and hole outlines in a single loop.

Consider a binary link with a hole at one end. The cross-section of this body comprises two closed paths—one for the cross-section outline, the other for the hole. The paths are physically separated. However, you can connect them by cutting each path at a vertex and joining the cut vertices with additional line segments. The animated figure shows the drawing of a binary link cross-section with one hole.



You can extend this approach to cross-sections with multiple holes. Note that each hole must have a cut. There is no single best way to approach the cuts. The key is to plan them so that you can traverse the cross-section—and all of its holes—in a single continuous path while keeping the polyline from intersecting itself. The animated figure shows the drawing of a binary link cross-section with two holes.



Try It: Define a Cross-Section with Two Holes

Modify the coordinate matrix in your `modelParams` script to include two identical holes as shown in the figure. Save the script often as you go.



Binary Link Cross-Section (Two Holes)

Start by adding a new variable for the hole diameter (d). Set the diameter to 1.2 (in what will later be units of cm).

```
d = 1.2;
```

Generate a new angular span array for the left and right holes. The holes are drawn in the same order and a single array suffices. The array elements are ordered in a clockwise direction, ensuring that when generating the cross-section the solid region stays to the left. The number of array elements has doubled to reflect the wider angular span of the holes (360° vs 180°).

```
C = linspace(3*pi/2, -pi/2, 10)';
```

Define the outlines of the left hole (`csLeftHole`), the right hole (`csRightHole`), and the connection line between the two (`csConnLine`). The x-coordinates are shifted left by half the length ($l/2$) for the left hole and right by the same distance for the right hole.

```
csLeftHole = [-l/2 + d/2*cos(C) d/2*sin(C)];
csRightHole = [+l/2 + d/2*cos(C) d/2*sin(C)];
csConnLine = [-l/2 -w/2; +l/2 -w/2];
```

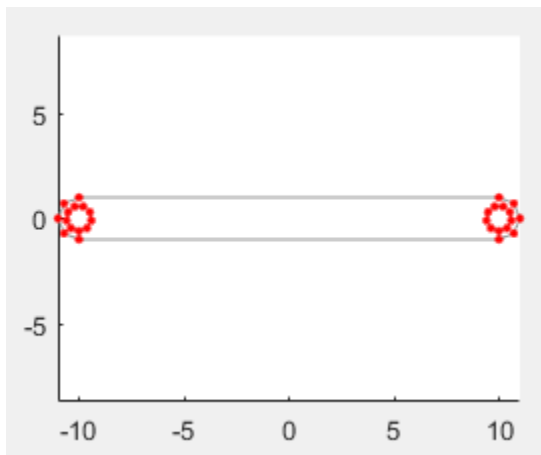
Add the new coordinate matrices to the existing `cs` matrix. The order of the matrices determines the order in which the complete cross-section is drawn. The result is a variable that you can specify in the **Cross-section** parameter of the Revolved Solid block.

```
cs = [csRight; csLeft; csLeftHole; ...
      csConnLine; csRightHole];
```

As before, you can visualize the cross-section outline using the MATLAB `plot` command. Ensure that the plotting code shown below is included in your script. Then, run the script to generate the plot.

```
figure; hold on; axis equal;
plot(cs(:,1), cs(:,2), 'Color', [0.6 0.6 0.6], 'Marker', '.', ...
      'MarkerSize', 9, 'MarkerEdgeColor', [1 0 0]);
```

The plot shows the cross-section of the binary link. The points in the coordinate matrix are shown as red dots. The resulting cross-section outline is shown as a light gray line. Note that the hole sections each comprise ten points—the number specified in the angular span arrays.




From Cross-Sections to Solids

The z -axis of the reference frame serves as the sweep axis in both the Extruded Solid and Revolved Solid blocks. The specified cross-section is swept along this axis in the case of Extruded Solid block and about this axis in the case of the Revolved Solid block. The sweep is symmetrical with respect to the cross-section plane: it runs half of the sweep length or angle in each direction of the sweep axis. This symmetry leaves the cross-section plane—and therefore the origin of the reference frame—halfway between the ends of the sweep.

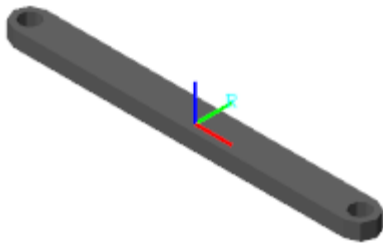
Try It: Use Your Cross-Section to Model a Solid

- 1 Open a new Simulink® model and, from the Bodies library, add a Extruded Solid block. You can click the model canvas, type the block name, and make a selection from the options shown. Save the model in a convenient location as `binaryLinkSolid`.
- 2 In the Extruded Solid block dialog box, set the **Geometry** parameters as shown in the table. Set the parameter units to cm. The **Cross-Section** parameter is defined in terms of the `cs` variable in your `modelParams` script.

Parameter	Value
Cross-Section	CS
Length	1

- 3 Load your `modelParams` script to your model workspace:
 - a In the **Modeling** tab, click **Model Explorer**. You use this tool to load the `modelParams` script that you previously created onto your model workspace.
 - b In the **Model Hierarchy** pane, expand the node corresponding to your model (**binaryLinkSolid**) and select **Model Workspace**.
 - c In the **Model Workspace** pane, set the **Data source** parameter to **MATLAB File** and browse for your `modelParams` script. Click the **Reinitialize from Source** button to load variables defined in the script.
- 4 In the Extruded Solid block dialog box, click the **Update Visualization** button, . The visualization pane refreshes with the final solid geometry.

Click the **Fit to View** button to scale the binary link to the size of the visualization pane. Click the **Toggle visibility of frames** button to show the solid reference frame. The reference frame origin coincides with the [0, 0] cross-section coordinate and lies halfway between the extrusion ends.



Note the jagged appearance of the round ends and holes. This effect results from the small number of points used in the round portions of the coordinate matrix—`csLeftEnd`, `csRightEnd`, `csLeftHole`, and `csRightHole`. Increase the number of elements in the angular span arrays to obtain a smoother geometry.

See Also

More About

- “Representing Solid Geometry” on page 1-38
- “Representing Solid Inertia” on page 1-62
- “Manipulate the Color of a Solid” on page 1-90
- “Visualize a Model and Its Components” on page 1-58

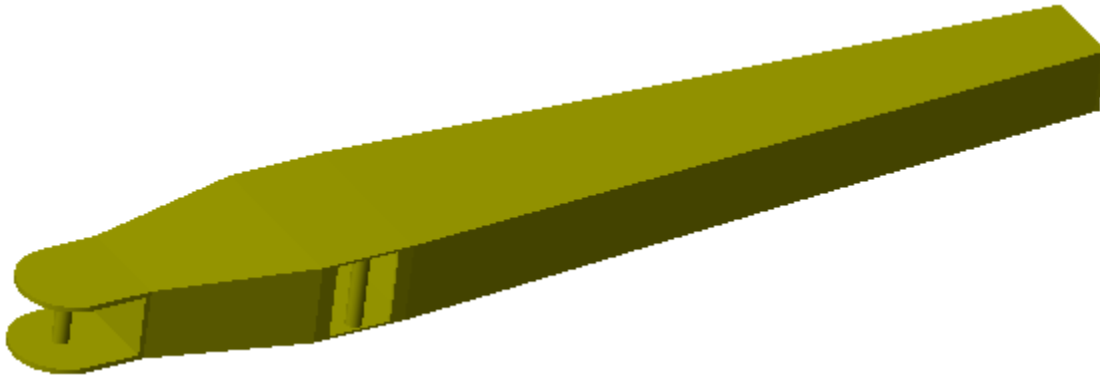
Model an Excavator Dipper Arm as a Flexible Body

The Reduced Order Flexible Solid block models a deformable body based on a reduced-order model that characterizes the geometric and mechanical properties of the body. The basic data imported from the reduced-order model includes:

- A list of coordinate triples that specify the position of all interface frame origins relative to a common reference frame.
- A symmetric stiffness matrix that describes the elastic properties of the flexible body.
- A symmetric mass matrix that describes the inertial properties of the flexible body.

There are several ways to generate the reduced-order data required by this block. Typically, you generate a substructure (or superelement) by using finite-element analysis (FEA) tools.

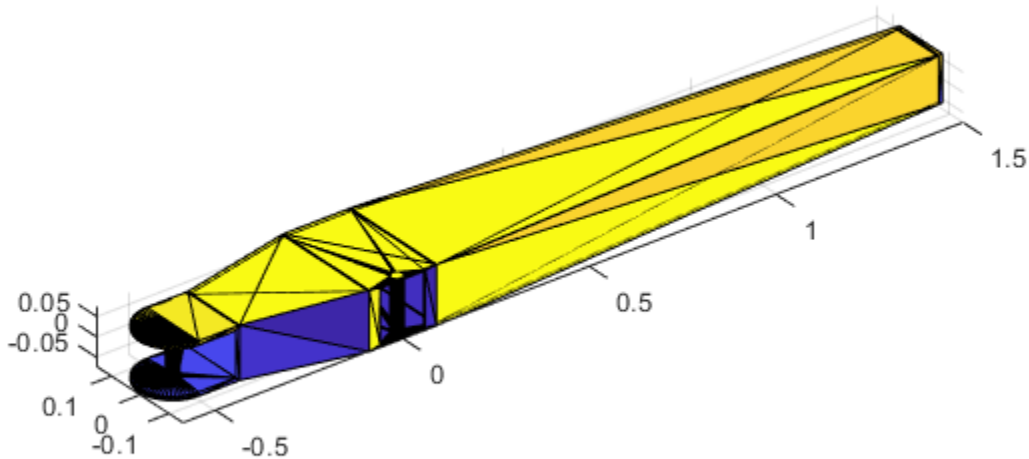
This example uses the Partial Differential Equation Toolbox™ to create a reduced-order model for a flexible dipper arm, such as the arm for an excavator or a backhoe. You start with the CAD geometry of the dipper arm, generate a finite-element mesh, apply the Craig-Bampton FEA substructuring method, and generate a reduced-order model. The model `sm_flexible_dipper_arm` uses the reduced-order data from this example. In the model, the dipper arm is mounted on top of a rotating tower as part of a test rig. For more information, see “Flexible Dipper Arm” on page 8-113.



Step 1: Define the Geometry and Material Properties of the Dipper Arm

The file `sm_flexible_dipper_arm.STL` contains a triangulation that defines the CAD geometry of the dipper arm. To view the geometry stored in this file, use the MATLAB® functions `stlread` and `trisurf`:

```
stlFile = 'sm_flexible_dipper_arm.STL';  
figure  
trisurf(stlread(stlFile))  
axis equal
```



The dipper arm is constructed from steel. To represent its material properties, set these values for Young's modulus, Poisson's ratio, and mass density:

```
E = 200e9;      % Young's modulus in Pa
nu = 0.26;     % Poisson's ratio (nondimensional)
rho = 7800;    % Mass density in kg/m^3
```

Step 2: Specify the Locations of Interface Frames

The dipper arm has three interface frames where you can connect other Simscape™ Multibody™ elements, such as joints, constraints, forces, and sensors:

- The cylinder connection point, where the arm connects to a hydraulic cylinder that actuates the arm vertically.
- The bucket connection point, where the arm connects to the excavator bucket.
- The fulcrum point, where the arm connects to the excavator boom.

The positions of all interface frame origins are specified in meters relative to same common reference frame used by the CAD geometry.

```
origins = [-0.500  0      0      % Frame 1: Cylinder connection point
           1.500  0      0      % Frame 2: Bucket connection point
           0      -0.130  0];   % Frame 3: Fulcrum point
numFrames = size(origins,1);
```

Step 3: Create the Finite-Element Mesh

To generate the mesh for the dipper arm, first call the `createpde` (Partial Differential Equation Toolbox) function, which creates a structural model for modal analysis of a solid (3-D) problem. After importing the geometry and material properties of the arm, the `generateMesh` (Partial Differential Equation Toolbox) function creates the mesh.

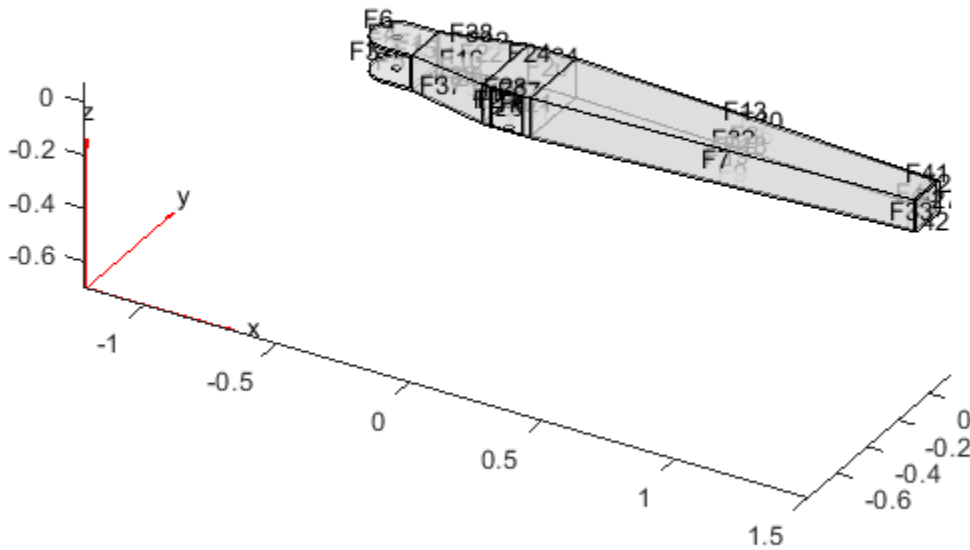
```
feModel = createpde('structural','modal-solid');
importGeometry(feModel, stlFile);
structuralProperties(feModel, ...
    'YoungsModulus', E, ...
    'PoissonsRatio', nu, ...
    'MassDensity', rho);
generateMesh(feModel, ...
    'GeometricOrder', 'quadratic', ...
    'Hmax', 0.2, ...
    'Hmin', 0.02);
```

Step 4: Set up the Multipoint Constraints for the Interface Frames

Each interface frame on the block corresponds to a boundary node that contributes six degrees of freedom to the reduced-order model. There are several ways to ensure that the FEA substructuring method preserves the required degrees of freedom. For example, you can create a rigid constraint to connect the boundary node to a subset of finite-element nodes on the body. You can also use structural elements, such as beam or shell elements, to introduce nodes with six degrees of freedom.

This example uses a multipoint constraint (MPC) to preserve the six degrees of freedom at each boundary node. To identify the geometric regions (such as faces, edges, or vertices) to associate with each MPC, first plot the arm geometry by using the function `pdegplot` (Partial Differential Equation Toolbox):

```
figure
pdegplot(feModel, 'FaceLabels', 'on', 'FaceAlpha', 0.5)
```



You can zoom, rotate, and pan this image to determine the labels for the faces corresponding to the boundary nodes. These faces define the MPCs associated with the boundary nodes in the dipper arm:

- Cylinder connection point: face 1
- Bucket connection point: face 27
- Fulcrum point: face 23

```
faceIDs = [1,27,23]; % List in the same order as the interface frame origins
```

To verify these values, plot the mesh and highlight the selected faces:

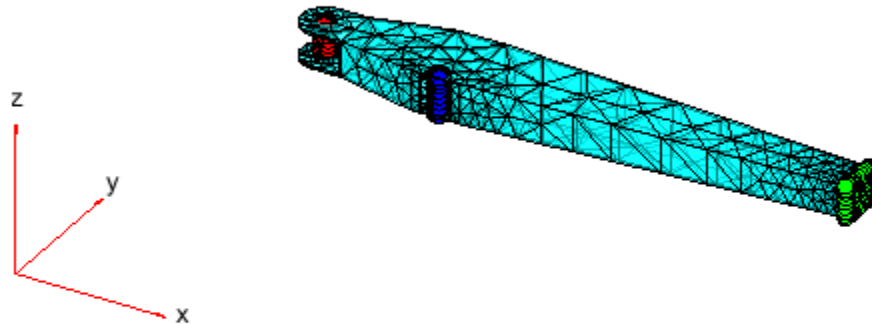
```
figure
pdemesh(feModel, 'FaceAlpha', 0.5)
hold on
colors = ['rgb' repmat('k', 1, numFrames-3)];
assert(numel(faceIDs) == numFrames);
for k = 1:numFrames
    nodeIdxs = findNodes(feModel.Mesh, 'region', 'Face', faceIDs(k));
    scatter3( ...
        feModel.Mesh.Nodes(1, nodeIdxs), ...
        feModel.Mesh.Nodes(2, nodeIdxs), ...
        feModel.Mesh.Nodes(3, nodeIdxs), ...
        'ok', 'MarkerFaceColor', colors(k))
    scatter3( ...
        origins(k,1), ...
        origins(k,2), ...
        origins(k,3), ...
```



```

    80,colors(k),'filled','s')
end
hold off

```



Call the function `structuralBC` (Partial Differential Equation Toolbox) to define the MPCs for the boundary nodes in these faces:

```

for k = 1:numFrames
    structuralBC(feModel, ...
        'Face',faceIDs(k), ...
        'Constraint','multipoint', ...
        'Reference',origins(k,:));
end

```

Step 5: Generate the Reduced-Order Model

The function `reduce` (Partial Differential Equation Toolbox) applies the Craig-Bampton order reduction method and retains all fixed-interface modes up to a frequency of 10^4 radians per second.

```
rom = reduce(feModel,'FrequencyRange',[0 1e4]);
```

Store the results of the reduction in a data structure `arm`. Transpose the `ReferenceLocations` matrix to account for the different layout conventions used by Partial Differential Equation Toolbox and Simscape Multibody.

```

arm.P = rom.ReferenceLocations'; % Interface frame locations (n x 3 matrix)
arm.K = rom.K;                  % Reduced stiffness matrix
arm.M = rom.M;                  % Reduced mass matrix

```

The function `computeModalDampingMatrix`, which is defined at the bottom of this page on page 1-56, computes a reduced modal damping matrix with a damping ratio of 0.05:

```
dampingRatio = 0.05;
arm.C = computeModalDampingMatrix(dampingRatio, rom.K, rom.M);
```

The boundary nodes in the reduced-order model must be specified in the same order as the corresponding interface frames on the block. This order is given by the rows of the array `origins`. If the order of the MPCs is different than the order specified by `origins`, permute the rows and columns of the various matrices so that they match the original order.

```
frmPerm = zeros(numFrames,1);    % Frame permutation vector
dofPerm = 1:size(arm.K,1);      % DOF permutation vector

assert(size(arm.P,1) == numFrames);
for i = 1:numFrames
    for j = 1:numFrames
        if isequal(arm.P(j,:),origins(i,:))
            frmPerm(i) = j;
            dofPerm(6*(i-1)+(1:6)) = 6*(j-1)+(1:6);
            continue;
        end
    end
end
assert(numel(frmPerm) == numFrames);
assert(numel(dofPerm) == size(arm.K,1));

arm.P = arm.P(frmPerm,:);
arm.K = arm.K(dofPerm,:);
arm.K = arm.K(:,dofPerm);
arm.M = arm.M(dofPerm,:);
arm.M = arm.M(:,dofPerm);
arm.C = arm.C(dofPerm,:);
arm.C = arm.C(:,dofPerm);
```

Step 6: Import Reduced-Order Data

The model `sm_flexible_dipper_arm` uses the data structure `arm` to set up the parameters of the Reduced Order Flexible Solid block. In the block, these parameters import the reduced-order data:

- **Origins:** `arm.P`
- **Stiffness Matrix:** `arm.K(1:24,1:24)`
- **Mass Matrix:** `arm.M(1:24,1:24)`
- **Damping Matrix:** `arm.C(1:24,1:24)`

For more information, see “Flexible Dipper Arm” on page 8-113.

Compute the Modal Damping Matrix

This function computes a modal damping matrix associated with the stiffness matrix `K` and mass matrix `M`. This function applies a single scalar damping ratio to all of the flexible (non-rigid-body) normal modes associated with `K` and `M`.

```
function C = computeModalDampingMatrix(dampingRatio,K,M)

% To avoid numerical issues (such as complex eigenvalues with very small
```

```

% imaginary parts), make the matrices exactly symmetric.

    K = (K+K')/2;    % Stiffness matrix
    M = (M+M')/2;    % Mass matrix

% Compute the eigen-decomposition associated with the mass and stiffness
% matrices, sorting the eigenvalues in ascending order and permuting
% the corresponding eigenvectors.

    [V,D] = eig(K,M);
    [d,sortIdxs] = sort(diag(D));
    V = V(:,sortIdxs);

% Due to small numerical errors, the six eigenvalues associated with the
% rigid-body modes may not be exactly zero. To avoid numerical issues,
% check that the first six eigenvalues are close enough to zero. Then
% replace them with exact 0 values.

    assert(all(abs(d(1:6))/abs(d(7)) < 1e-9),'Error due to "zero" eigenvalues.');
```

```

    d(1:6) = 0;

% Vectors of generalized masses and natural frequencies

    MV = M*V;
    generalizedMasses = diag(V'*MV);
    naturalFrequencies = sqrt(d);

% Compute the modal damping matrix associated with K and M

    C = MV * diag(2*dampingRatio*naturalFrequencies./generalizedMasses) * MV';

end

```

See Also

Reduced Order Flexible Solid | stlread | trisurf | createpde | importGeometry | structuralProperties | generateMesh | pdegplot | structuralBC | reduce

More About

- “Flexible Dipper Arm” on page 8-113

Visualize a Model and Its Components

In this section...

“Visualize a Complete Multibody Model” on page 1-58

“Visualize an Individual Solid Geometry” on page 1-59

“A Note on Imported Geometries” on page 1-60

Visualize a Complete Multibody Model

Model visualizations open in **Mechanics Explorer**—the Simscape Multibody visualization utility. By default, Mechanics Explorer starts automatically when you first update a model (in the **Modeling** tab, click **Update Model**) or simulate a model. To change this setting, see “Enable Mechanics Explorer” on page 5-2.

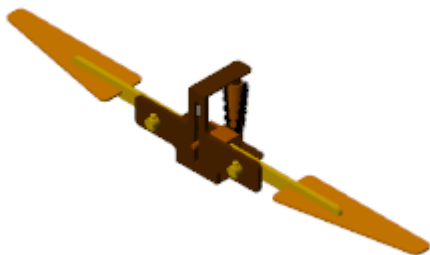
The visualizations consist of the multibody subassemblies, compound bodies, and simple bodies present in your models. Mechanics Explorer displays model entities such as solid geometries, spline curves, inertia icons, and frames. You can selectively show and hide individual entities using the context-sensitive menu of the Mechanics Explorer tree view pane. See “Selective Model Visualization” on page 5-15.

A visualization is static when you update a model and dynamic when you simulate a model. The static visualization in this case shows the model in its initial configuration, with the joints in their initial states. The dynamic visualization shows a 3-D animation that you can record—using either **Video Creator** or the `smwritevideo` function—see “Create a Model Animation Video” on page 5-28.

You can manipulate the viewpoint using the tool strip located above the visualization pane. The tool strip enables you to rotate, roll, pan, and zoom the model view. A **Camera Manager** enables you to create dynamic cameras that move with the model to keep it in view during simulation—see “Visualization Cameras” on page 5-8.

Try It: Visualize a Multibody Model

- 1 At the MATLAB command prompt, enter `sm_cam_flapping_wing`. The model shown in the flapping-wing featured example opens.
- 2 In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a view of the model in its initial state.



- 3 Click **Run**. Mechanics Explorer plays an interactive 3-D animation based on the simulation results.



A Note About Inertia Ellipsoids

Equivalent inertia ellipsoids provide an intuitive means to visualize variable inertias (modeled using the General Variable Mass block). These ellipsoids are rendered dynamically, with their dimensions and poses obtained at each time step from the specified instantaneous inertial properties. Ellipsoid visualization is not available on model update. Variable inertial properties are specified through physical signals whose values are inaccessible to the blocks until the simulation begins.



Inertia Ellipsoid in a Tank Truck Model

Visualize an Individual Solid Geometry

Blocks such as Brick Solid and Spline provide visualization panes that show the geometries specified in the block **Parameters** sections. Use the block visualizations to catch geometry errors as they occur—for example, incorrect dimensions, colors, and, in solid blocks, frame placements.

The block visualizations are similar to those provided in Mechanics Explorer. You can rotate, roll, pan, and zoom the view using a tool strip located above the visualization pane. The tool strip includes buttons for standard views such as **Front**, **Top**, and **Isometric**. You can show or hide the frames of the block.

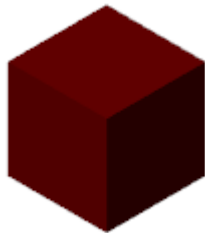
You can refresh block visualizations without updating or simulating a model. The visualizations refresh the moment you click the **Update Visualization** button in the visualization tool strip. Using MATLAB variables in block parameters does not affect your ability to refresh a visualization.

Try It: Visualize a Solid

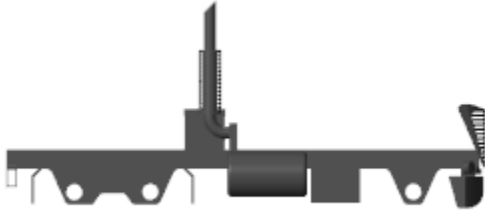
- 1 At the MATLAB command prompt, enter `smnew` to open the Simscape Multibody model template. The template contains commonly used blocks, including the Brick Solid block.
- 2 Open the Brick Solid block dialog box. The visualization pane is by default expanded to show an isometric view of the default solid—a gray brick.



- 3 Set the **Graphic > Color** parameter to `[0.5 0 0]`—an RGB vector corresponding to a dark red color.
- 4 Click the **Update Visualization** button, located above the solid visualization pane. The visualization updates to show what is now a red brick.

**A Note on Imported Geometries**

You can import solid geometries from STEP or STL files. Corrupt or invalid geometry files cause visualization and simulation issues. Geometries associated with such files are not shown during model visualization. The corresponding solid inertias cannot be automatically computed from the solid geometries and the simulation fails if the **Inertia > Type** parameter is set to `Calculate from Geometry`.



An Imported Solid Geometry

See Also

Related Examples

- “Manipulate the Visualization Viewpoint” on page 5-4
- “Create a Model Animation Video” on page 5-28
- “Go to a Block from Mechanics Explorer” on page 5-27

Representing Solid Inertia

In this section...

“Representing Inertias” on page 1-62

“Compounding Solids and Inertias” on page 1-66

Representing Inertias

Inertia is a basic attribute of anything you might construe as a body. It is a resistance to a change in one’s state of motion, and, equivalently, a measure of the force or torque needed to induce a certain acceleration. Unlike other solid attributes, such as geometry or color, it is strictly required for the simulation of a model. In particular, the ends of a joint—its *frames*—must each connect to an inertia, which is to say that where motion is allowed, there must exist an inertia for an applied force or torque to act upon.

You can model an inertia element in isolation, without the intent to represent a body. Such inertias are useful, for example, when simulating the vibrations induced by a clump of mud on a rotating automobile wheel. The clump is separate from the wheel body and you can model it as such. In addition, its geometry and color are in this case trivial details and you can disregard them for modeling purposes. In so doing, you treat the clump as a plain inertia—one lacking any attributes other than inertia.

Isolated plain inertias are uncommon in a model. Generally, you account for inertia in the course of modeling a complete body—something with geometry and color, like a wing in the flapping wing mechanism discussed in “Modeling Bodies” on page 1-4. You start with a concept of the body, model that body as a collection of solids, and specify the attributes of those solids to obtain a complete representation of the body. Solids are the things that you model and inertia merely one of their attributes.



A Body (1) and a Plain Inertia (2)

Relevant Blocks

You add inertia to a model using blocks from the Body Elements library. Relevant blocks include the solid blocks, Inertia, and those in the Variable Mass sublibrary. You can model a complete solid or a plain inertia. Either can have fixed or variable inertia parameters, though the exact parameterization, and therefore the type of solid or inertia, depends on the block. The term “solid” is used here to denote an element whose attributes extend beyond merely inertia and the term “inertia,” when used to refer to an element, one whose attributes encompass only inertia.

Fixed Solids

Use the solid blocks when modeling solids and the bodies they comprise. These blocks enable you to specify geometry and color, key attributes if solid visualization is important to you. They also enable you to have the less accessible parameters of rotational inertia automatically computed from the solid geometry and either mass or mass density. Even in cases where geometry and color are superfluous

details, the solid blocks are often the most convenient means of specifying inertia. Note that the geometry and inertia parameters of the solid blocks are strictly constant. To model solids with either as a variable attribute, you must use blocks from the Variable Mass sublibrary.

Variable Solids

Use the solid blocks in the Body Elements > Variable Mass library to model complete solids with variable inertia parameters, such as mass, and inertia-dependent dimensions, such as length and radius, that can vary dynamically with the inertia inputs. Blocks that represent solids are identified as such by having the word *Solid* in their names—for example, Variable Cylindrical Solid and Variable Brick Solid. These blocks differ from the solid blocks in the parent library in that one or more inertia parameters can change, and from the General Variable Mass block in that they possess geometry and color.

Fixed Inertias

Use the Inertia block as a means of adjusting the inertia of a solid or body. Geometry and color are considered irrelevant for modeling purposes. You can subtract a mass to account for the existence of a hollow region, such as an empty compartment in a vessel originally modeled without one. You can also add a mass to account for the presence of small disturbances, such as the clumps of mud that sometimes linger on an automobile wheel. Note that you can make the same adjustments, sometimes more intuitively, using the more sophisticated solid blocks.

Variable Inertias

Reserve the General Variable Mass block for the special cases in which mass, center of mass, or the inertia tensor must change in response to some input—often just time itself—without making assumptions about solid geometry. You can model events such as the scooping of a load by a backhoe (an example of a variable mass), the movement of an occupant on a manlift (an example of a variable center of mass), and the sloshing of a fluid load contained in a tank truck (an example of a variable inertia tensor).

Inertia Parameters

Solid blocks have access to geometry data and can therefore calculate inertia parameters given a shape and a mass. This feature greatly reduces the number of parameters that you must specify in a model. Automatic inertia calculation is always enabled in solid blocks, such as Brick Solid and Cylindrical Solid. It is enabled by default in the solid blocks, meaning that you can change this setting.

You can also specify the inertia parameters explicitly, for example, to precisely capture the inertia of a body for which you have only a rough geometry. An example is an odd-shaped link, for example one typical of a backhoe excavator arm, that you have approximated using a simple Brick Solid block shape. The solid geometry is in this case not very accurate and you may prefer to specify the inertia parameters using CAD (or other) data.

If you choose to specify inertia explicitly, there are two parameterizations that you can use. One enables you to treat the solid or inertia as a point mass: the **Point Mass** parameterization. The other enables you to treat the solid or inertia as a distributed mass: the **Custom** parameterization. You can select the option best suited for your application using the **Inertia > Type** dropdown list.

Inertia		
Type	Custom	
Mass	Calculate from Geometry	
Center of Mass	Point Mass	
Moments of Iner...	Custom	
Products of Inertia	[1 1 1]	kg*m^2

Note that the Point Mass and Custom parameterizations are available only in those blocks that support the explicit specification of inertia. The variable solid blocks in the Body Elements > Variable Mass library do not provide either. In those blocks, the center of mass and inertia tensor are strictly constrained to the solid geometry and density and are, for this reason, always automatically calculated during simulation.

The Point Mass Approximation

A point mass is an approximation that has as its only inertial parameters the center of mass and the total mass—a measure of translational inertia and therefore of the resistance to a sudden change in translational velocity. Rotational inertia is assumed negligible and is ignored. The location of the center of mass can vary with respect to the origin of the local reference frame.

Inertia		
Type	Point Mass	
Mass	1	kg

Custom Mass Distributions

A distributed mass is a more general representation of inertia. It has among its inertial parameters not only the total mass and center of mass, but also the moments of inertia, and products of inertia. The moments and products of inertia comprise what is known as the inertia tensor or matrix. Together, these parameters suffice to completely describe, from a multibody modeling perspective, the distribution of a mass in space.

Inertia		
Type	Custom	
Mass	1	kg
Center of Mass	[0 0 0]	m
Moments of Iner...	[1 1 1]	kg*m^2
Products of Inertia	[1 1 1]	kg*m^2

A Note on Joint Connections

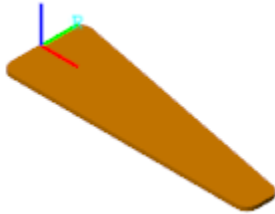
Use caution when connecting inertias with zero moments of inertia, such as point masses, to joints with rotational degrees of freedom—those composed at least in part of revolute or spherical joint primitives. The combined moment of inertia about the rotational axes of the joint must be nonzero on each side. The reason for this is simple: the angular acceleration about an axis becomes infinite regardless of the torque applied if the moment of inertia about that axis is zero. This behavior is not physical and is disallowed in a model.

Reference Frames

Blocks in the Body Elements library have reference frame ports that you connect to resolve the placement of the respective elements—solids, inertias—in the context of a model. The reference

frames are a rigid part of those elements and naturally move with them as a unit. They are used, directly or indirectly, to define the inertias and, in solids, the geometries of the elements.

If the concept of a frame is foreign to you, see “Working with Frames” on page 1-24. Succinctly, a frame is an axis triad much like a Cartesian coordinate system. It has a position and orientation that you can define using the frame creation interface of the Solid block or the parameters of the Rigid Transform block. All positions and orientations in a model—of solids, inertias, joints and constraints, forces and torques, sensors—are defined through frames.



Reference Frame of a Solid

Visualization Options

You can visualize solids and inertias in a model. The type of visualization that you get depends on the block that you use. Solid blocks, including those from the Body Elements > Variable Mass library, enable you to visualize their respective elements using the geometries that you specify. You can also visualize the solid using a simple graphic marker such as a sphere—for example, to highlight its position in cases where geometry is known to be inaccurate.

Inertias lack geometry and color and naturally do not support geometry-based visualization. You must visualize such elements using alternative means. If the element is associated with a General Variable Mass block, you can use the same graphic markers provided in the solid blocks or an equivalent inertia ellipsoid—a shape whose dimensions depend directly on the inertia parameters that you specify. If the element is associated with an Inertia block, you can use the markers of the solid blocks or an inertia icon.

For more information on visualization, see “Visualize a Model and Its Components” on page 1-58.

Try It: Add an Inertia to a Model

Add a plain fixed inertia to a double-pendulum model, position it at the free end of the outer link, and set its mass to 25 g using the `Point Mass` parameterization:

- 1 At the MATLAB command prompt, enter `smdoc_double_pendulum`. A model of a double-pendulum opens up. In it are three Simulink Subsystem blocks, each representing a body. Save the model under a different name in a convenient folder.
- 2 From the Body Elements library, add an Inertia block and connect its reference frame port (labeled **R**) to the rightmost frame port of the `Binary Link A1` block. The frame associated with this port is located at the free end of the double pendulum.
- 3 In the Inertia block dialog box, set the Mass parameter to 25 g—a value roughly equivalent to a quarter of the mass of a binary link (130 g). The `Point Mass` parameterization in this block enables you to ignore the rotational inertia parameters.
- 4 Simulate the model. Mechanics Explorer opens with a dynamic visualization of your updated double-pendulum. Notice the inertia icon used to denote the location of your inertia element.



See “Try It: Specify a Custom Inertia” for an example showing how to specify the parameters of a Custom inertia.

Compounding Solids and Inertias

As solid shapes grow in complexity, inertia parameters become increasingly cumbersome to specify and a different approach may suit you better: compounding. You can conceive of a complex solid or inertia as a collection of simpler elements and specify their inertia parameters explicitly, if using the Inertia or General Variable Mass block, or configure them for automatic calculation, if using a solid block.

When you rigidly connect the simpler elements—via frame connection lines and, if needed, Rigid Transform blocks—you obtain an aggregate whose inertia properties mirror those of the complex solid or inertia you intended to represent. The binary link shown in the figure serves as an example. You can divide the link into three sections, represent each section using a separate block, and connect the respective reference frames using appropriate rigid transforms.



For an example showing how to specify the geometry of a binary link through compounding, see “Try It: Create a Compound Geometry” on page 1-15.

Negative Inertia as Subtraction

There is no requirement in the Simscape Multibody environment that the inertia parameters be positive. This includes the mass and moments of inertia, both parameters that in the physical world are strictly positive. Negative inertias enable you to model compound inertias with hollow sections by subtraction and are therefore useful in certain models.

The binary link again serves as an example. You can represent the link as a single piece without holes using one block, and subtract from its ends the inertias of the holes using additional blocks. As before, you must use rigid transforms to properly position the inertia reference frames relative to each other.



For an example, showing how to specify an inertia by compounding, see “Try It: Create a Compound Inertia” on page 1-18.

See Also

More About

- “Modeling Bodies” on page 1-4
- “Compounding Body Elements” on page 1-15
- “Specifying Custom Inertias” on page 1-68
- “Specifying Variable Inertias” on page 1-76

Specifying Custom Inertias

In this section...

“Key Inertia Conventions” on page 1-68
 “Inertia Matrix Definitions” on page 1-68
 “CAD as an Inertia Data Source” on page 1-71
 “Automatic Inertia Calculation” on page 1-74

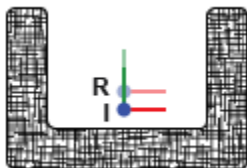
Key Inertia Conventions

Simscape Multibody software adopts a number of conventions in its inertia definitions. Take note of these as they may impact your inertia calculations if you perform them manually. The conventions may also bear on what additional processing, if any, your inertia data needs—for example, if it was sourced from a CAD application or other third-party software. In particular:

- The products of inertia are defined using a negated convention, one with a minus sign inserted, explicitly, in the mass integrals. An alternate convention exists in which a minus sign does not accompany the mass integrals. Recall that the products of inertia are the off-diagonal elements of the inertia matrix.
- The center of mass is defined with respect to the local reference frame of the block. In solids with imported CAD shapes, this frame is generally that assumed by your CAD application in its inertia calculations. It is possible, however, to modify a solid geometry file so that the two frames no longer match.

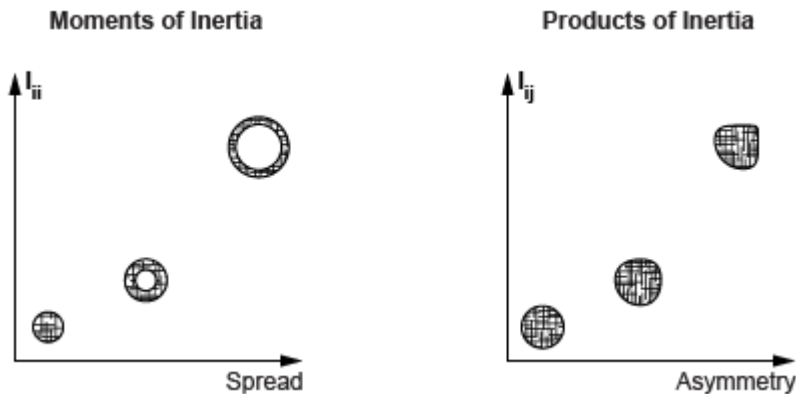
Inertia Matrix Definitions

The inertia matrix captures the spatial distribution of matter about a local frame referred to here as the inertia frame of resolution. This frame is labeled **I** in the figure. Its axes are parallel to those of the local reference frame, associated with frame port **R** and correspondingly labeled **R**. However, its origin coincides instead with the local center of mass.



The inertia matrix comprises the moments and products of inertia. The moments of inertia occupy the diagonal matrix positions and measure the dispersion, or spread, of the mass distribution about the axes of the inertia frame of resolution. The greater the spread about an axis, the larger the moment of inertia corresponding to that axis.

The products of inertia occupy the off-diagonal positions and measure the asymmetry of the mass distribution with respect to the planes of the inertia frame of resolution. The greater the asymmetry about a plane, the larger the products of inertia associated with any axis in that plane. The figure illustrates these relationships.



The Inertia Equations

The matrix is symmetric with respect to the main diagonal line: off-diagonal elements whose indices are reciprocals of each other share the same value. This constraint reduces the number of unique products of inertia from the original six (all those in off-diagonal positions) to the three that you must specify in a block (those with a unique combination of indices):

- $I_{yz} = I_{zy}$
- $I_{zx} = I_{xz}$
- $I_{xy} = I_{yx}$

The products of inertia, I_{ij} , are defined using the prevalent, negated, convention adopted by a number of CAD applications:

- $I_{yz} = - \int_V (yz)\rho dv$
- $I_{zx} = - \int_V (zx)\rho dv$
- $I_{xy} = - \int_V (xy)\rho dv$

where ρ is mass density, v is volume, and V is the total volume of integration. The x , y , and z coordinates are the Cartesian components of the distance vector spanning from the center of mass to an infinitesimal element of mass ρdv . The moments of inertia, I_{ii} , are similarly defined:

- $I_{xx} = \int_V (y^2 + z^2)\rho dv$
- $I_{yy} = \int_V (z^2 + x^2)\rho dv$
- $I_{zz} = \int_V (x^2 + y^2)\rho dv$

When applied to simple shapes such as cylindrical shells and rectangular beams, these definitions give rise to well-known algebraic equations that are often published in standard engineering tables.

You can reference such tables when specifying the inertia parameters explicitly. The complete inertia matrix, according to the Simscape Multibody convention, is:

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

Principal Axes of Inertia

The moments of inertia are by definition positive numbers. However, the products of inertia can be either positive, negative, or zero. They are zero if the axes of the inertia frame of resolution happen to coincide with the principal axes of inertia. The moments of inertia are then called the principal moments of inertia and the inertia matrix is said to be *diagonal*:

$$I = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}$$

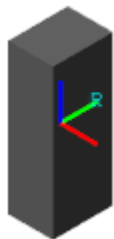
The number of nontrivial inertia matrix elements that you must specify is in this case reduced to three—the principal moments of inertia. For this reason, the principal axes of inertia can be a convenient frame in which to specify the inertia matrix elements. This is the inertia frame of resolution assumed in the highly symmetrical preset shapes of the solid blocks.

The same, however, is not generally true of Extruded Solid or Revolved Solid solid shapes, nor is it of those imported via STEP or STL files. In Extruded Solid and Revolved Solid shapes, the frame placement depends closely on how you define the geometrical cross-sections. In imported shapes, it depends on how, relative to the local zero coordinate, the part geometries were modeled.

As a best practice, always consider the placement of the inertia frame of resolution when specifying the elements of the inertia matrix explicitly, particularly when using a solid block. The frame position is always that of the center of mass, but its orientation relative to a solid geometry, when using a solid block, may not always coincide with the principal axes of inertia.

Try It: Specify a Custom Inertia

Consider the rectangular beam shown in the figure. Determine its mass, center of mass, moments of inertia, and products of inertia. Specify the calculated parameters explicitly in a Brick Solid block using a Custom inertia parameterization.



Material and Dimensions

Assume a construction of aluminum and a corresponding mass density of 0.09754 lbm/in³. Use the beam dimensions:

- Width $x = 3$ in
- Height $y = 4$ in
- Length $z = 10$ in

Prepare the Beam Model

Add a Brick Solid block to a Simscape Multibody model. In the Brick Solid block dialog box, specify the beam geometry: set the **Geometry > Dimensions** parameter to $[3\ 4\ 10]$ in. This array corresponds to the beam dimensions $[x\ y\ z]$.

The geometry type affects the placement of the local reference frame (**R**) and therefore the inertia calculations themselves. In the visualization toolstrip, click the **Toggle visibility of frames** button. Frame **R** is located at the center of mass and its axes are parallel to the beam dimensions (x , y , and z).

Specify the Inertia Parameters

Calculate the inertia parameters from the density and dimensions of the beam. Then, specify the calculated values in the **Inertia** section of the Brick Solid block parameters:

- **Mass** — Product of mass density (ρ) and volume ($x \cdot y \cdot z$):

$$m = \rho(x \cdot y \cdot z) = 11.7 \text{ lb}$$

- **Center of Mass** — Centroid coordinates with respect to the local reference frame (**R**):

$$[\bar{x}, \bar{y}, \bar{z}] = [0, 0, 0]$$

- **Moments of Inertia** — From standard expressions with respect to the inertia frame of resolution (**I**):

$$[I_{xx}, I_{yy}, I_{zz}] = \frac{m}{12} [(y^2 + z^2), (z^2 + x^2), (x^2 + y^2)] = [113.1, 106.3, 24.4] \text{ lbm} \cdot \text{in}^2$$

- **Products of Inertia** — From symmetry with respect to the inertia frame of resolution (**I**):

$$[I_{yz}, I_{zx}, I_{xy}] = [0, 0, 0]$$

CAD as an Inertia Data Source

CAD applications often provide the inertia data for your part models. Examples include SolidWorks software, through its Mass Properties tool, and Onshape software, through its version of the same tool. You can reference this data and specify it manually in the Simscape Multibody environment.

Alternate Inertia Conventions

Some CAD applications, SolidWorks among them, use an alternate inertia convention to define the elements of the inertia matrix. This convention removes the minus sign from the product-of-inertia definitions. The I_{yz} product of inertia, for example, becomes:

$$I_{yz} = \int_V (yz) \rho dv$$

If your source of inertia data adopts this convention, you must explicitly negate the products of inertia before specifying their values in the Simscape Multibody environment. As an example, consider a SolidWorks inertia matrix given as:

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

To correctly specify the matrix elements in the Simscape Multibody environment, you must treat them as follows:

$$I = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{pmatrix}$$

CAD Import as an Alternative

Rather than reference the inertia data in a CAD assembly model, you can import that model into the Simscape Multibody environment. CAD import is based on the `smimport` function, which parses a multibody description file in XML format and generates an equivalent block diagram with all block parameters prespecified—inertia parameters included.

You must export your CAD model in a valid XML format, meaning one that conforms to the Simscape Multibody XML schema, before you can import it. This option may suit you only if you have a complete CAD assembly model. For individual CAD parts, use the STEP file import feature of the solid block and set the **Inertia > Type** parameter to `Calculate from Geometry`.

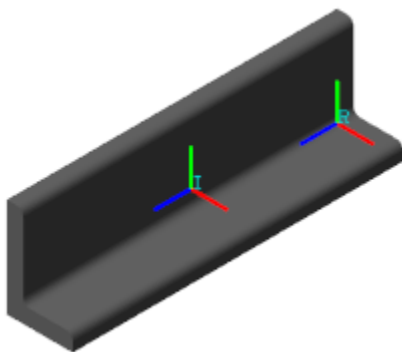
For more information, see “CAD Translation” on page 6-2.

Try It: Reference a SolidWorks Model

Determine the inertia parameters for the L-beam shape shown in the figure. Then, specify them explicitly in a solid block by setting the inertia parameterization to `Custom`. Use the mass properties data provided in this example for a SolidWorks model of the beam.

Open the Solid Model

At the MATLAB command prompt, enter `smdoc_lbeam_inertia`. A simple model opens with a File Solid block representing the L-beam solid. Open the File Solid block and explore its **Geometry** parameters. The beam geometry is imported from a STEP file previously exported from a SolidWorks model. This geometry is shown in the figure.



In the visualization pane, click the **Toggle visibility of frames** button. The visualization pane shows two frames, one labeled **R** and the other **I**.

Frame **R** is the local reference frame of the solid. It coincides with what SolidWorks users refer to as the *output coordinate system* of the part model. This frame is located at the lower corner of the L-shape on one of the two longitudinal ends of the beam. You must specify the center of mass relative to this frame.

Frame **I** is a custom solid frame included for your convenience. This frame coincides with the inertia frame of resolution. Its origin is at the center of mass and its axes are parallel to those of the local reference frame. You must specify the moments and products of inertia relative to this frame.

Review the SolidWorks Data

The SolidWorks model provides for the L-beam part the following mass properties data:

```
Mass properties of l_beam_solid
Configuration: Default
Coordinate system: -- default --
```

```
Density = 0.10 pounds per cubic inch
Mass = 2.19 pounds
Volume = 22.41 cubic inches
Surface area = 101.91 square inches
```

```
Center of mass: ( inches )
X = 0.58
Y = 1.08
Z = 5.00
```

```
Principal axes of inertia and principal moments of inertia:
( pounds * square inches )
Taken at the center of mass.
Ix = ( 0.00, 0.00, 1.00)      Px = 2.49
Iy = ( 0.38, -0.92, 0.00)   Py = 18.65
Iz = ( 0.92, 0.38, 0.00)   Pz = 20.35
```

```
Moments of inertia: ( pounds * square inches )
Taken at the center of mass and aligned with
the output coordinate system.
Lxx = 20.10   Lxy = -0.60   Lxz = 0.00
Lyx = -0.60   Lyy = 18.89   Lyz = 0.00
Lzx = 0.00   Lzy = 0.00   Lzz = 2.49
```

```
Moments of inertia: ( pounds * square inches )
Taken at the output coordinate system.
Ixx = 77.40   Ixy = 0.76   Ixz = 6.33
Iyx = 0.76   Iyy = 74.39   Iyz = 11.79
Izx = 6.33   Izy = 11.79   Izz = 5.76
```

The data includes the coordinates of the center of mass with respect to the “output coordinate system.” This coordinate system coincides with the local reference frame (**R**) of the corresponding Simscape Multibody solid.

The data includes also a matrix with the moments and products of inertia “taken at the center of mass and aligned with the output coordinate system.” This coordinate system coincides with the inertia frame of resolution (**I**) of the Simscape Multibody solid.

Specify the Inertia Parameters

Expand the **Inertia** parameters section of the File Solid block dialog box. Then, change the inertia parameterization by setting the **Inertia > Type** parameter to **Custom**. The complete set of inertia parameters appears for you to specify.

- 1 Set the **Mass** parameter to 2.19 lb. This is the mass corresponding to the density of aluminum.
- 2 Set the **Center of Mass** parameter to [0.58 1.08 5.00] in. These are the [x y z] coordinates of the center of mass shown in the SolidWorks report.
- 3 Set the **Moments of Inertia** parameter to [20.10 18.89 2.49] lbm*in². These are the [Lxx Lyy Lzz] moments of inertia shown in the SolidWorks report.
- 4 Set the **Products of Inertia** parameter to [0 0 0.6] lbm*in². These are the negated [Lyz Lzx Lxy] products of inertia shown in the SolidWorks report.

Automatic Inertia Calculation

The solid blocks provides an option to automatically compute the majority of inertia parameters given a solid geometry. This option, available from the **Inertia > Type** drop-down list and by default on, requires you to specify only the geometry parameters and either the mass or mass density.

The block uses the geometry and mass parameters to compute the remaining inertia parameters—the center of mass, moments of inertia, and products of inertia—relative to the appropriate frame of reference. The calculations are based on the assumption of a mass density that is constant and uniform.

You can view the calculation results inside the solid block, in an expandable section named **Display Inertia**. The center of mass is given relative to the local reference frame (**R**), and the moments and products of inertia relative to the inertia frame of resolution (**I**). These are the same frames relative to which you might specify these parameters.

Try It: Display the Calculated Inertia Results

Configure the File Solid block of the `smdoc_lbeam_inertia` model to calculate the inertia parameters from the solid geometry and its mass density. Then, view the calculated parameters.

- 1 In the dialog box of the File Solid block, switch the **Inertia > Type** parameter to **Calculate** from **Geometry**. A **Display Inertia** node appears below the **Density** parameter.
- 2 Set the **Density** parameter to 0.09754 lbm/in². This value corresponds to a solid of aluminum construction and it is the same assumed in the SolidWorks data provided in “Review the SolidWorks Data” on page 1-73.
- 3 Expand the **Display Inertia** node and click the **Update** button. The inertia parameters under **Display Inertia** are populated with the calculated values. Compare them to the values provided in the SolidWorks mass properties data.

Derived Values	Update
Mass	+9.916786e-01 kg
Center of Mass	[+1.469190e-02, +2.734326e-02, +1.270000e-...
Moments of Ine...	[+5.871301e-03, +5.518708e-03, +7.268190e-...
Products of Iner...	[+8.673617e-19, -2.168404e-19, +1.756705e-04]

See Also

More About

- “Modeling Bodies” on page 1-4
- “Compounding Body Elements” on page 1-15
- “Specifying Variable Inertias” on page 1-76

Specifying Variable Inertias

In this section...

“Modeling Variable Inertias” on page 1-76

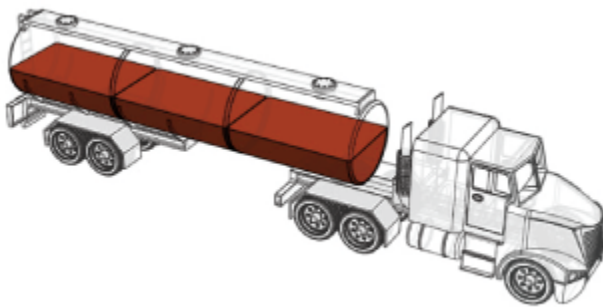
“Visualizing Variable Inertias” on page 1-76

“Modeling Body Interactions” on page 1-77

“Model a Variable-Mass Oscillator” on page 1-77

Modeling Variable Inertias

A variable inertia is a mass element whose mass, center of mass, or inertia tensor can vary through time. Variable inertias include the scooped contents of a backhoe bucket, the moving occupants of a boom manlift, and the sloshing fluid load of a decelerating tank truck. You model a variable inertia using the General Variable Mass block from the **Body Elements > Variable Mass** library. This block accepts the various inertial properties as constants or variables. Physical signal ports provide the means to specify the variable properties.



A Fluid Load as a Variable Inertia

Specifying the Variable Inputs

You can specify your variable inputs using Simscape or Simulink blocks. You must convert any Simulink signals into physical signals using the Simulink-PS Converter block. Avoid sudden changes as these can increase model stiffness and slow down simulation. Ensure that the signal dimensions agree with the ports:

- Scalar for mass (port **m**)
- Three-element vector for center of mass (port **com**)
- Nine-element matrix for inertia tensor (port **I**)

Visualizing Variable Inertias

Variable inertias associated with General Variable Mass blocks have no geometry. You must visualize these inertias as graphical markers or as equivalent inertia ellipsoids. The ellipsoid dimensions and geometry center vary with mass, center of mass, and inertia tensor, lending the ellipsoids to more informative model visualizations. Inertia markers are shown on model update and during simulation.

Variable inertia ellipsoids are shown during simulation only. The figure shows an inertia visualization representing a fluid load carried by a tank truck.



Equivalent Inertia Ellipsoid Visualization

Modeling Body Interactions

The General Variable Mass block captures inertial effects only. Any interactions between variable inertias and other model components must be modeled explicitly. Examples of interactions include contact forces between the fluid load of a tank truck and the surrounding enclosure. They include also the change in momentum obtained by expelling the combustion products of a propulsion system—for example, in a marine or space vehicle. Use other Simscape Multibody, Simscape, and Simulink blocks to capture interactions like these.

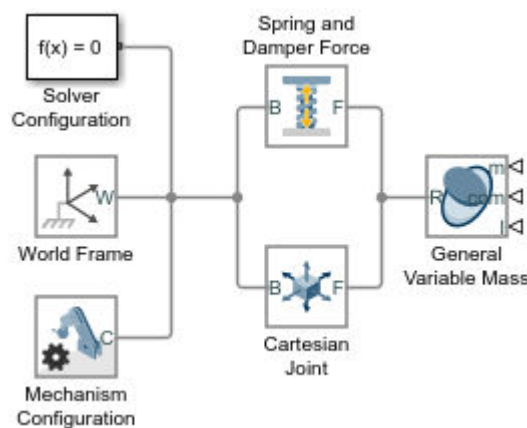
Model a Variable-Mass Oscillator

Create a simple model of a mass-spring system to simulate under constant-mass and variable-mass conditions. The model uses a General Variable Mass block to represent a container onto which a load of sand is progressively dropped. A Cartesian Joint block provides the variable-mass body with three translational degrees of freedom, although only one—along the vertical z -axis—is relevant during simulation. A Spring and Damper Force block represents the spring element, which connects the variable-mass body to the World frame.



Create Block Diagram

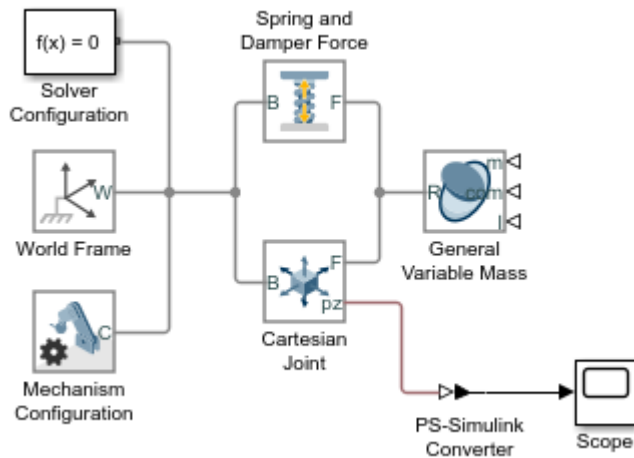
- 1 At the MATLAB command prompt, enter `smnew`. The command opens a model template with commonly used Simscape Multibody blocks.
- 2 Add the following blocks to the model canvas:
 - General Variable Mass (**Body Elements > Variable Mass**)
 - Cartesian Joint (**Joints**)
 - Spring and Damper Force (**Forces and Torques**)
- 3 Connect the blocks as shown in the figure and delete the remaining blocks. Ensure that the joint block orientation is as shown, with the base frame port facing the World Frame block.



- 4 In the Spring and Damper Force block dialog box, set the **Natural Length** parameter to 0.2 m and the **Spring Stiffness** parameter to 10 N/m.
- 5 In the Cartesian Joint block dialog box, expand the **Z Prismatic Primitive (Pz)** area, select the **State Targets > Specify Position Target** check box, and set the **Value** parameter to 0.1 m.

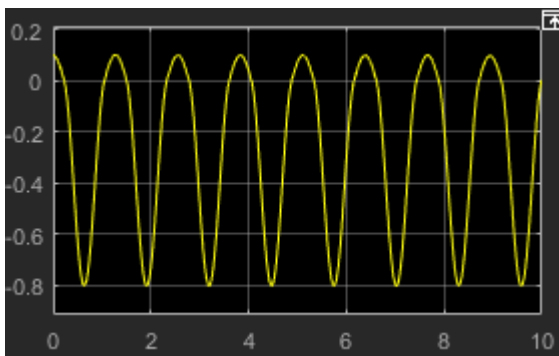
Add Position Sensing

- 1 In the Cartesian Joint block dialog box, expand the **Z Prismatic Primitive (Pz)** area and select the **Sensing > Position** check box. The block exposes a physical signal output port with the oscillator frame position.
- 2 Add the following blocks to the model canvas:
 - PS-Simulink Converter (**Simscape > Foundation Library > Utilities**)
 - Scope (**Simulink > Sinks**)
- 3 Connect the blocks as shown in the figure.



Simulate with Constant Mass

- 1 In the General Variable Mass block dialog box, set the **Type** parameter to Custom. This option enables you to model a variable mass distribution with rotational inertia.
- 2 Set the **Mass**, **Center of Mass**, and **Inertia Matrix** parameters to Constant and the **Mass > Value** parameter to 0.2 kg.
- 3 Run the simulation and open the Scope block. The plot shows the position of the reference frame of the variable mass. Note that the oscillation frequency and amplitude stay constant throughout simulation.

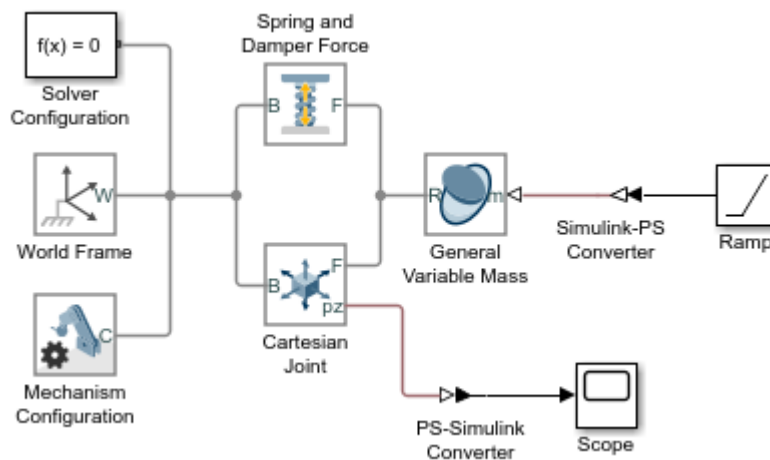


Mechanics Explorer opens with a 3-D animation of the model. The visualization comprises only an inertia ellipsoid—here a sphere due to the symmetry of the default inertia tensor used in the model. In the menu bar, select **View > Show Frames** to show all the frames in the model. Note that the ellipsoid dimensions stay constant during simulation, reflecting the constant inertial properties specified in the model.

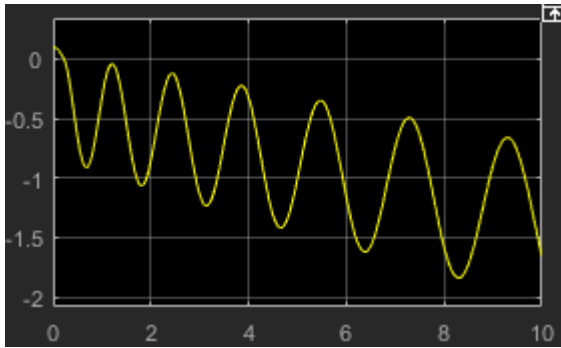


Simulate with Variable Mass

- 1 In the General Variable Mass block dialog box, set the **Center of Mass** and **Inertia Matrix** parameters each to **Constant**. The physical signal ports used to vary their values during simulation become hidden, leaving only the solid mass as a variable.
- 2 Add these blocks to the model canvas:
 - Simulink-PS Converter (**Simscape** > **Foundation Library** > **Utilities**)
 - Ramp (**Simulink** > **Sources**)
- 3 Connect the blocks as shown in the figure.



- 4 In the Ramp block dialog box, set the **Slope** parameter to 0.1 and the **Initial output** parameter to 0.2. The Ramp signal is passed to the General Variable Mass block in the default Simscape units of mass, kg. The signal corresponds to a steadily increasing mass that starts at 0.2 kg and ends at 1.2 kg following a 10-second simulation.
- 5 Run the simulation and open the Scope block. The position plot shows a variable oscillation frequency and amplitude. The increasing mass causes the oscillation frequency to increase and the amplitude to decrease.



Mechanics Explorer updates the visualization results. Note that the ellipsoid dimensions decrease as the simulation progresses, reflecting their inverse proportionality to the variable mass.



See Also

More About

- “Modeling Bodies” on page 1-4
- “Compounding Body Elements” on page 1-15
- “Specifying Custom Inertias” on page 1-68
- “Specifying Variable Inertias” on page 1-76

Creating Custom Solid Frames

In this section...

“Solid Frames” on page 1-82

“Opening the Frame-Creation Interface” on page 1-83

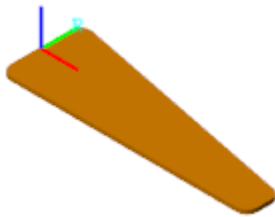
“Geometry-Based Frame Placement” on page 1-83

“Primary and Secondary Axes” on page 1-83

“Try It: Create a Custom Solid Frame” on page 1-84

Solid Frames

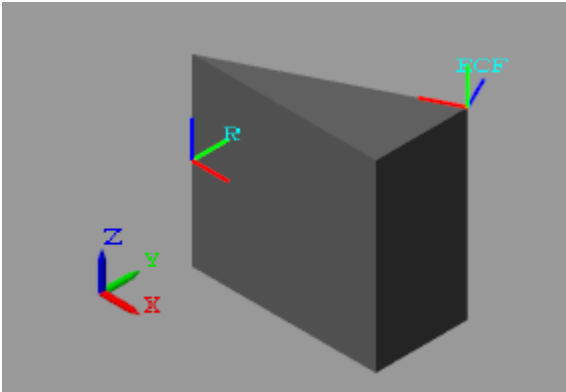
By default, the Solid block provides only a reference frame port, labeled **R**. In simple shapes, such as Brick Solid, Spherical Solid, and Cylindrical Solid, the reference frame origin coincides with the solid center of mass. The same is generally not true of the more sophisticated Extruded Solid and Revolved Solid shapes, nor is it of imported solid shapes.



Reference Frame of a Solid



In many applications, the reference frame of a solid is inadequate for connecting joints and constraints or for applying forces and torques. In such cases, you can create new frames external to the Solid block using the Rigid Transform block. This block enables you to define the new frame by specifying translation and rotation transforms numerically.

An alternative approach, and one that is often more intuitive, is to create new frames directly in the Solid block dialog box using the frame-creation interface. This interface enables you to define new frames interactively by aligning the frame origin and axes with geometric features such as planes, lines, and points.



Reference and Custom Frames of a Solid

Opening the Frame-Creation Interface

The frame-creation interface is accessible through the Solid block dialog box. To open the interface, in the **Frames** expandable area, select the **Create** button . If you change any of the block parameters, you must first update the solid visualization by selecting the **Update Visualization** button .

Geometry-Based Frame Placement

You can define frames based on geometric features of the solid or a choice of two frames—reference and principal inertia frames. The reference frame is the default frame of the solid. The principal inertia frame is one whose origin coincides with the center of mass and whose axes coincide with the principal axes of the solid.

Frames that you define by geometric features are specific to the shapes the features belong to. If you make the frame origin coincident with the vertex of a brick, the new frame is valid only for that particular brick shape. If you change shapes, you must edit or delete the new frame, as the geometric features it depends on no longer exist.

The frame-creation interface has three sections for specifying the following:

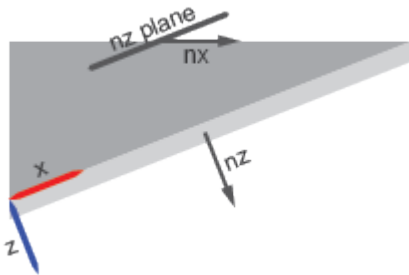
- Frame origin
- Primary axis
- Secondary axis

Primary and Secondary Axes

The primary axis constrains the possible directions of the remaining two axes. These axes must lie in the normal plane of the primary axis. If the axis or geometric feature used to define the secondary axis does not lie on this plane, the secondary axis is the projection of that axis or feature onto the normal plane.

The figure shows a top view of the three-sided extrusion you model in this tutorial. You align the primary axis (z) with the surface normal vector n_z and the secondary axis (x) with the line vector n_x .

Because n_x is not normal to the primary axis, the secondary axis is the projection of n_x onto the normal plane of the primary axis.




Try It: Create a Custom Solid Frame

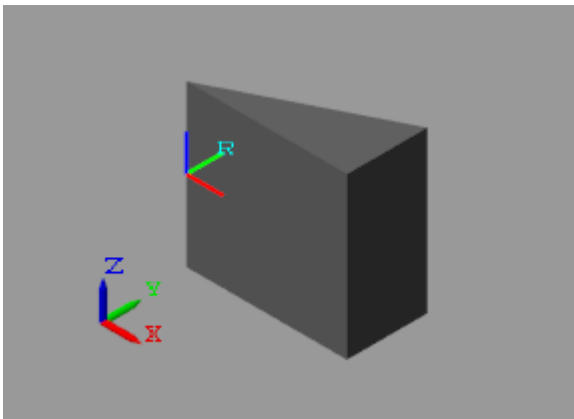
Create a frame on a solid using the frame-creation interface. The solid shape is a general extrusion with three unequal sides. This shape helps to demonstrate the difference between the primary and secondary frame axes that you specify in the frame creation interface.

Specify the Solid Shape


- 1 From the Body Elements library, add one Extruded Solid block to a new model. The Extruded Solid block provides its own visualization utility. You do not need to update the block diagram to visualize the solid shape or its frames.
- 2 In the Extruded Solid block dialog box, specify these parameters.

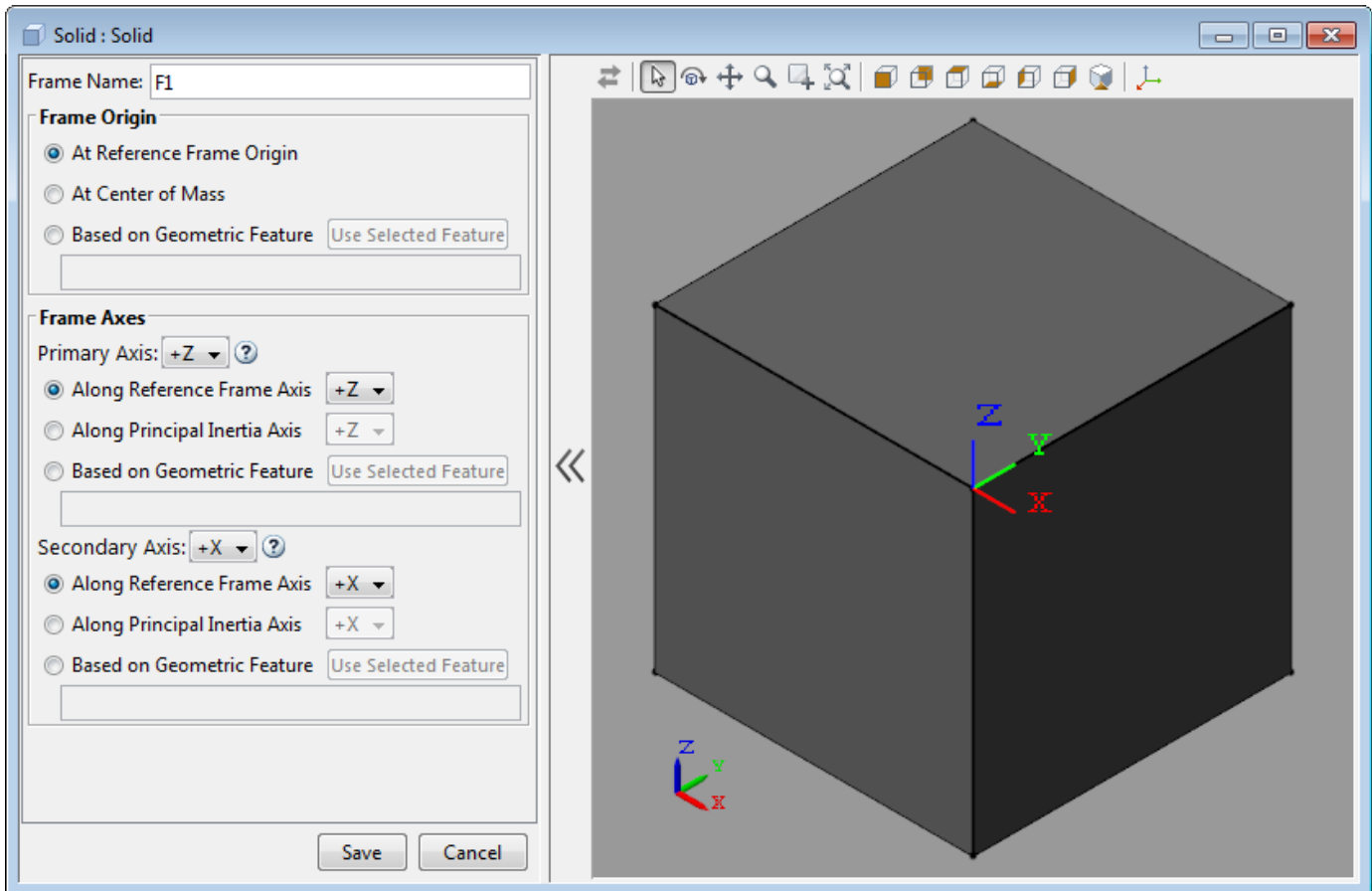
Parameter	Value
Geometry > Cross-section	[0, 0; 1, 0; 1, 0.5]

- 3 In the visualization toolbar, select the **Update Visualization** button . The visualization pane updates with the three-sided extrusion that you specified.
- 4 Select the **Toggle visibility of frames** button. The visualization pane shows all the frames in the solid. At this point, the solid has a single frame—its reference frame. The reference frame origin coincides with the [0,0] cross-section coordinate in the midplane of the extrusion.



Create the Frame

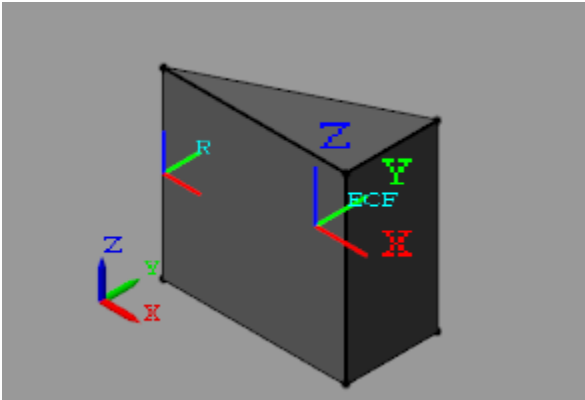
In the **Frames** expandable area of the Extruded Solid block dialog box, select the **Create** button . The Solid block opens the frame-creation interface.



In the **Frame Name** parameter, enter ECF (short for Extrusion Corner Frame). The frame name identifies the new frame in the Solid block visualization pane. It also appears as the frame port label on the Solid block.

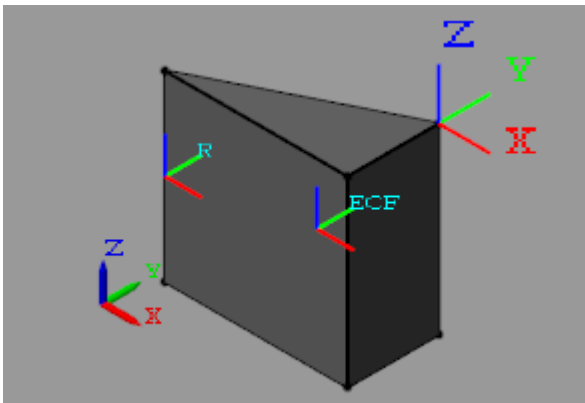
Specify the Frame Origin

Under **Frame Origin**, select **At Center of Mass**. The visualization pane updates with the new frame at the center of mass of the solid. This frame has the default frame orientation, that of the reference frame. The label ECF identifies the new frame.



Experiment with other frame origin locations. Define the origin location using one of the extrusion vertices.

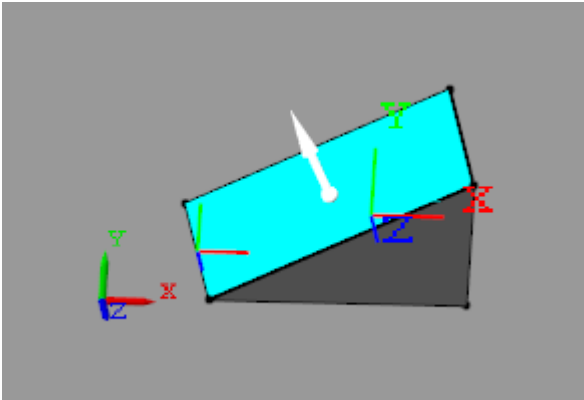
- 1 Under **Frame Origin**, select **Based on Geometric Feature**. This option enables you to select a point or the center of a plane or line as the frame origin.
- 2 In the visualization pane, select the vertex shown in the figure. The vertex is in the top plane of the extrusion. Ensure the view point is set to **Isometric**. In the **Frame Origin** area, ensure the vertex is named Location of top point 3.
- 3 Under **Frame Origin**, select the **Use Selected Feature** button. The visualization pane updates with the frame origin at the selected corner.



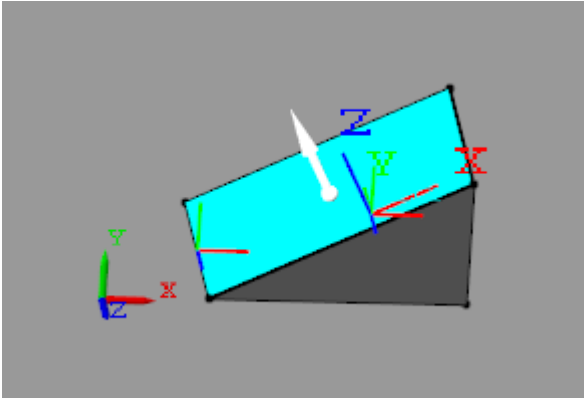
Specify the Primary Axis

The primary axis constrains the remaining two axes to lie on its normal plane. In this sense, the primary axis plays the dominant role in setting the orientation of the frame. Make the primary axis normal to the surface that contains the cross-section hypotenuse:

- 1 In the **Frame Axes** area under **Primary Axis**, select **Based on Geometric Feature**. The direction you specify in the next steps is that of the default primary axis, +Z.
- 2 In the visualization pane, rotate the solid and select the surface shown. The visualization pane highlights the surface and shows its normal vector. In the **Frame Axes** area under **Primary Axis**, ensure the surface is named Surface normal of side surface 3.



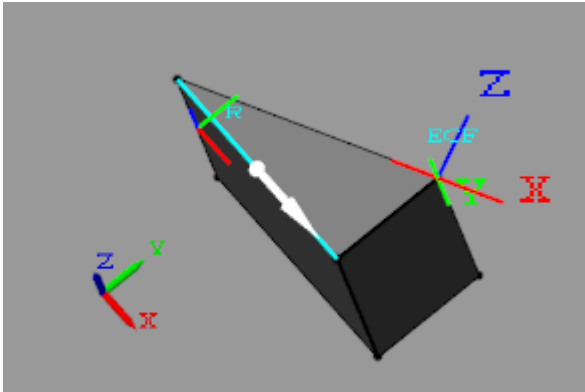
- 3 In the **Frame Axes** area under **Primary Axis**, select the **Use Selected Feature** button. The visualization pane updates with the z axis of the ECF frame, shown in dark blue, parallel to the normal vector of the selected surface.



Specify the Secondary Axis

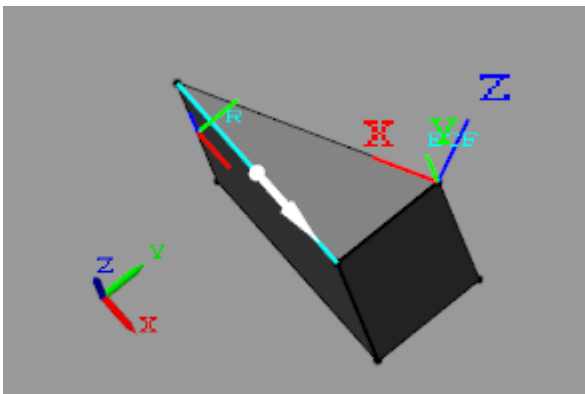
The secondary axis completes the definition of the new frame. In conjunction with the primary axis, the secondary axis fully constrains the direction of the third axis. The secondary axis is itself constrained to lie on the normal plane of the primary axis. To see the effects of this constraint, define the secondary axis based on a line not normal to the primary axis:

- 1 In the **Frame Axes** area, set the **Secondary Axis** parameter to **-X**. The direction you specify in the following steps is that of the **-X** axis.
- 2 In the **Frame Axes** area, under **Secondary Axis**, select **Based on Geometric Feature**.
- 3 In the visualization pane, rotate the solid and select the line shown. In the **Frame Axes** area, under **Secondary Axis**, ensure this line is named **Curve direction of top curve 1**.



- 4 Select the **Use Selected Feature** button. The visualization pane updates with the x axis of the frame, shown in red, partially aligned with the selected line.

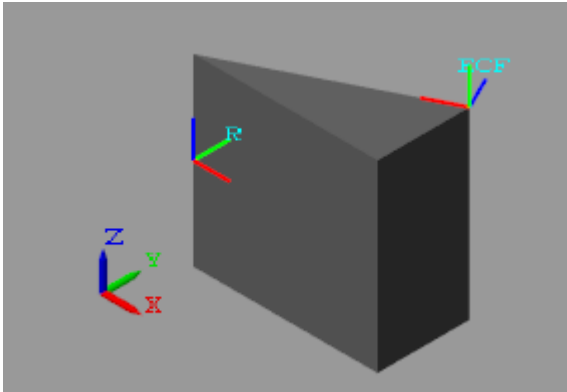
The two are not completely aligned as the selected line does not lie on the normal plane of the primary axis. The secondary axis is therefore the projection of the selected line onto the normal plane of the primary axis.



Save the New Frame

To save the frame you defined and commit it to the model:

- 1 Select the **Save** button. The visualization pane shows the solid with the final version of the frame you defined.
- 2 In the main interface of the Solid block dialog box, select **OK** or **Apply**. The Solid block commits the new frame to the model and exposes a new frame port labeled with the frame name you specified.



See Also

More About

- “Modeling Bodies” on page 1-4
- “Working with Frames” on page 1-24
- “Creating Connection Frames” on page 1-33
- “Visualize Simscape Multibody Frames” on page 5-24

Manipulate the Color of a Solid

In this section...

“Visual Property Parameterizations” on page 1-90

“RGB and RGBA Vectors” on page 1-90

“Simple Visual Properties” on page 1-91

“Advanced Visual Properties” on page 1-92

“Adjust Solid Opacity” on page 1-92

“Adjust Highlight Color” on page 1-93

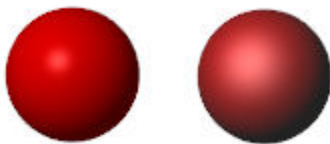
“Adjust Shadow Color” on page 1-93

“Adjust Self-Illumination Color” on page 1-94

Visual Property Parameterizations

The solid blocks provides two parameterizations, **Simple** and **Advanced**, that you can use to specify solid visual properties. The **Simple** parameterization provides control over the solid color and opacity. The **Advanced** parameterization adds control over highlight, shadow, and self-illumination colors. You can use the **Advanced** visual properties to model emissive solids such as the sun and glossy solids such as polished metal parts.

You select a visual-property parameterization through the **Graphic > Visual Properties** block parameter. The figure contrasts the two parameterizations. On the left is a solid with **Simple** visual properties. On the right is the same solid with **Advanced** visual properties—including **Specular Color** and **Shininess** parameters, which impart to the solid a slight metallic sheen.



Try It

Set the visual-property parameterization of a solid to **Advanced**:

- 1 Add a Spherical Solid block to a new model canvas. The block provides its own visualization pane. You can use this pane to visualize the solid even if the model is not topologically valid. In later examples, the curved spherical surface makes the specular highlights and ambient shadows easier to see.
- 2 In the Spherical Solid block dialog box, set the **Graphic > Visual Properties** parameter to **Advanced**. You set the visual-property parameterization individually for each solid.

RGB and RGBA Vectors

You can specify colors directly as [R,G,B] and [R,G,B,A] vectors on a normalized scale of 0-1. The R, G, and B elements provide the red, green, and blue contents of the specified color. The A element

provides the color opacity—the degree to which the solid obstructs other components located behind it. Omitting the A element is equivalent to setting its value to 1.

Try It

Identify the color and opacity given by the [R,G,B,A] vectors below:

- $[0, 0, 1, 1]$ — Denotes a solid color with no red or blue contents, maximum green content, and maximum opacity. The solid is bright green and fully opaque.



- $[1, 0, 0, 0.5]$ — Denotes a solid color with maximum red content, no green or blue contents, and partial opacity. The solid is bright red and transparent.



Simple Visual Properties

The **Simple** parameterization enables you to set the solid color and opacity. You can select a color using an interactive color picker or specify a color as an [R,G,B] vector. The **Color** parameter in the **Simple** parameterization is the same as the **Diffuse Color** parameter in the **Advanced** parameterization.

See It

See the parameters that comprise the **Simple** parameterization:

- 1 In the Spherical Solid block dialog box, set the **Visual Properties** parameter to **Simple**. This setting is the parameter default.
- 2 Expand the **Visual Properties** node. **Color** and **Opacity** appear as the active visual property parameters.

Graphic	
Type	From Geometry
Visual Properties	Simple
Color	[0.5 0.5 0.5]
Opacity	1.0

Advanced Visual Properties

The Advanced parameterization adds control over the highlight, shadow, and self-illumination colors as well as the size of the highlight areas. You must specify the colors directly as [R,G,B,A] vectors. The optional A element serves the same purpose as the **Opacity** parameter in the Simple parameterization.

See It

See the parameters that comprise the Advanced parameterization:

- 1 In the Spherical Solid block dialog box, set the **Visual Properties** parameter to Advanced.
- 2 Expand the **Visual Properties** node. The solid colors and shininess appear as the active visual property parameters.


Graphic	
Type	From Geometry
Visual Properties	Advanced
Diffuse Color	[0.5 0.5 0.5]
Specular Color	[0.5 0.5 0.5 1.0]
Ambient Color	[0.15 0.15 0.15 1.0]
Emissive Color	[0.0 0.0 0.0 1.0]
Shininess	75

Adjust Solid Opacity

You can make a solid transparent using either parameterization. If using the Simple parameterization, set the **Opacity** parameter to a value less than one. If using the Advanced parameterization, set the optional fourth element of the **Diffuse Color** [R,G,B,A] vector to a value less than one.

Try It

Model a transparent red solid using the Advanced visual-property parameterization:

- 1 Under the **Graphic > Visual Properties** node, change the **Diffuse Color** parameter to [1, 0, 0, 0.5].
- 2 In the visualization pane, click the  button to refresh the solid visualization.



Delete the fourth vector element in the **Diffuse Color** parameter or set its value to 1 in order to make the solid opaque again.




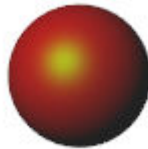
Adjust Highlight Color

You can control the size and color of specular highlights by adjusting the **Shininess** and **Specular Color** parameters in the Advanced visual-property parameterization. Lower the shininess value for large but soft highlights. Increase its value for small but sharp highlights.

Try It

Give the specular highlights a bright green hue. Set the **Diffuse Color** vector to $[1, 0, 0, 1]$ in order to make the solid opaque. Then:

- 1 In the **Graphic > Visual Properties** node, lower the **Shininess** parameter to 10. This value increases the highlight size, making the specular color easier to see.
- 2 Change the **Specular Color** parameter to $[0, 1, 0, 1]$. This vector sets the highlight color to bright green.
- 3 In the visualization pane, click the  button to refresh the solid visualization. The specular color combines with the diffuse color to give highlight areas a green hue.




Adjust Shadow Color

You can control the color of shadow areas by adjusting the **Ambient Color** parameter in the Advanced visual-property parameterization.

Try It

Give the shadow areas a slight blue hue:

- 1 In the **Graphic > Visual Properties** node, set the **Ambient Color** parameter to $[0.15, 0.15, 0.3]$. This vector sets the shadow color to dark blue.
- 2 In the visualization pane, click the  button to refresh the solid visualization. The ambient color combines with the diffuse color to give shadow areas a blue hue.




Adjust Self-Illumination Color

You can model self-illuminating solids such as the sun by adjusting the **Emissive Color** parameter in the Advanced visual-property parameterization.

Try It

Give the solid surface a red emissive color:

- 1 Under the **Graphic > Visual Properties** node, change the **Emissive Color** parameter to $[1, 0, 0, 1]$.
- 2 In the visualization pane, click the  button to refresh the solid visualization.



See Also

More About

- “Representing Solid Geometry” on page 1-38
- “Representing Solid Inertia” on page 1-62
- “Modeling Bodies” on page 1-4

Multibody Systems

- “Multibody Assembly Workflow” on page 2-2
- “Modeling Joint Connections” on page 2-4
- “How Multibody Assembly Works” on page 2-8
- “Counting Degrees of Freedom” on page 2-12
- “Model an Open-Loop Kinematic Chain” on page 2-13
- “Model a Closed-Loop Kinematic Chain” on page 2-16
- “Troubleshoot an Assembly Error” on page 2-21
- “Modeling Gear Constraints” on page 2-27
- “Assemble a Gear Model” on page 2-32
- “Model a Compound Gear Train” on page 2-47
- “Constrain a Point to a Curve” on page 2-58

Multibody Assembly Workflow

In this section...

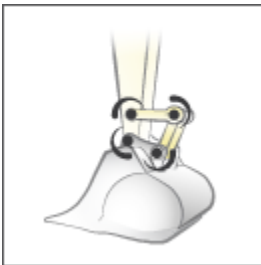
“Study the Joints and Constraints to Model” on page 2-2

“Assemble Bodies Using Joints and Constraints” on page 2-2

“Guide Model Assembly” on page 2-2

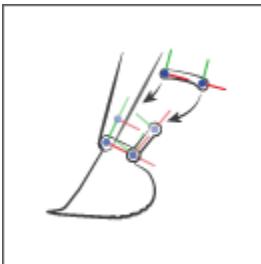
“Verify Model Assembly” on page 2-3

Study the Joints and Constraints to Model



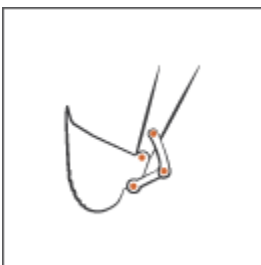
Identify the joints and constraints between the various bodies. Joints can be real, such as that between a piston and its case, or virtual, such as that between two planets.

Assemble Bodies Using Joints and Constraints



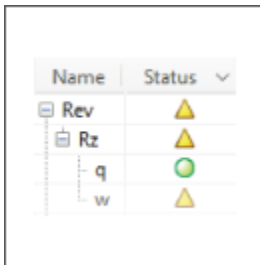
Model the degrees of freedom between bodies by connecting their frames through joints. You can further constrain these degrees of freedom through specialized constraints, such as those between gears. See “Assemble a Gear Model” on page 2-32 for an example.

Guide Model Assembly



Specify the state targets of the various joints. You can specify the desired position and velocity of a joint at time zero. If the state targets are valid and compatible, the joints assemble in the states specified. See the “Guide Assembly and Visualize Model” on page 2-19 section of “Model a Closed-Loop Kinematic Chain” on page 2-16 for an example.

Verify Model Assembly



Name	Status
Rev	⚠
Rz	⚠
q	●
w	⚠

Update the block diagram. Examine the model visualization for assembly issues. Open the Simscape Variable Viewer or the Simscape Multibody Model Report to see if all state targets have been satisfied. See the “Verify Model Assembly” on page 2-19 section of “Model a Closed-Loop Kinematic Chain” on page 2-16 for an example.

Modeling Joint Connections

In this section...

“Joint Degrees of Freedom” on page 2-4

“Joint Primitives” on page 2-5

“Joint Inertia” on page 2-6

Joints impose between bodies the primary kinematic constraints that determine how they can move relative to each other. A joint can be a physical connection, such as that between the case and shaft of a linear hydraulic actuator, or a virtual connection, such as that between the Earth and the moon. In Simscape Multibody, you model both connection types using Joint blocks.



Examples of physical and virtual connections between bodies

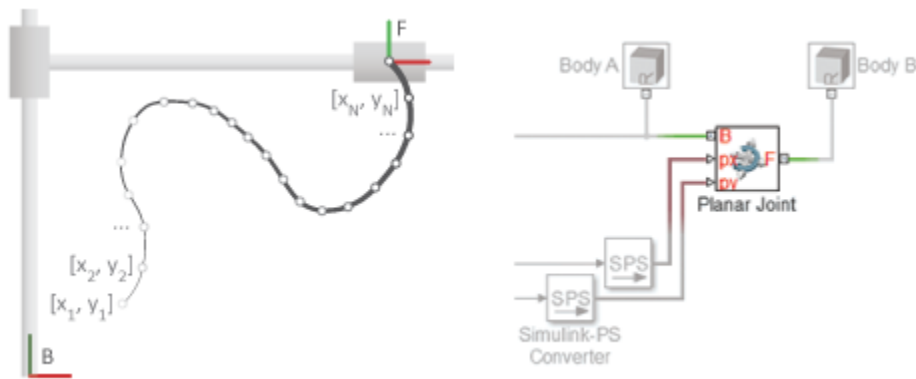
Gear and Constraint blocks too impose kinematic constraints between bodies. How are joint blocks different? While Gear and Constraint blocks are parameterized in terms of the DoFs they remove between bodies, Joint blocks are parameterized in terms of the DoFs they provide, through modules called joint primitives.

Joint Degrees of Freedom

Each Joint block connects exactly two bodies. Such a connection determines the maximum degrees of freedom, or DoFs, that the adjoining bodies can share. These DoFs range from zero in the Weld Joint block to six—three translational and three rotational—in 6-DOF Joint and Bushing Joint blocks. Translation refers to a change in position and rotation to a change in orientation.

Joint DoFs are a measure of joint mobility. Precluding other constraints in a model, a joint with more DoFs allows greater freedom of motion between the adjoining bodies. Joint DoFs also have a mathematical interpretation. They are the minimum number of state variables needed to fully determine the configuration of a joint at each time step during simulation.

Consider a rectangular joint. This joint allows translation in a plane and it therefore has two translational DoFs—one for each spatial dimension. At each time step, the joint configuration is fully determined by two state variables, the position coordinates in the plane of motion $[x(t), y(t)]$. This means, for example, that you can fully prescribe motion at this joint using two position input signals.



The table summarizes the DoFs that the various Joint blocks provide.

Joint Block	Translational DoFs	Rotational DoFs	Total DoFs
6-DOF Joint	3	3	6
Bearing Joint	1	3	4
Bushing Joint	3	3	6
Cartesian Joint	3	0	3
Constant Velocity Joint	0	2	2
Cylindrical Joint	1	1	2
Gimbal Joint	0	3	3
Leadscrew Joint	1 (coupled translational-rotational)		1
Pin Slot Joint	1	1	2
Planar Joint	2	1	3
Prismatic Joint	1	0	1
Rectangular Joint	2	0	2
Revolute Joint	0	1	1
Spherical Joint	0	3	3
Telescoping Joint	1	3	4
Universal Joint	0	2	2
Weld Joint	0	0	0

The actual DoFs at a joint are often fewer in number than the joint alone would allow. This happens when kinematic constraints elsewhere in the model limit the relative motion of the adjoining bodies. Such constraints can arise from gears in mesh, forbidden DoFs due to other joints in closed kinematic loops, and fixed distances and angles between bodies, among other factors.

Joint Primitives

Joint blocks are assortments of joint primitives, basic yet complete joints of various kinds you cannot decompose any further—at least without losing behavior such as the rotational-translational coupling of the lead screw joint. Joint primitives range in number from zero in the Weld Joint block to six in the Bushing Joint block. There are five joint primitives:

- Prismatic — Allows translation along a single standard axis (x, y, or z). Joint blocks can contain up to three prismatic joint primitives, one for each translational DoF. Prismatic primitives are labeled P*, where the asterisk denotes the axis of motion, e.g., Px, Py, or Pz.
- Revolute — Allows rotation about a single standard axis (x, y, or z). Joint blocks can contain up to three revolute joint primitives, one for each rotational DoF. Revolute primitives are labeled R*, where the asterisk denotes the axis of motion, e.g., Rx, Ry, or Rz.
- Spherical — Allows rotation about any 3-D axis, [x, y, z]. Joint blocks contain no more than one spherical primitive, and never in combination with revolute primitives. Spherical primitives are labeled S.
- Lead Screw Primitive — Allows coupled rotation and translation on a standard axis (e.g., z). This primitive converts between rotation at one end and translation at the other. Joint blocks contain no more than one lead screw primitive. Lead screw primitives are labeled LS*, where the asterisk denotes the axis of motion.
- Constant Velocity Joint — Allows rotation at constant velocity between intersecting though arbitrarily aligned shafts. Joint blocks contain no more than one constant velocity primitive. Constant velocity primitives are labeled CV.

The table summarizes the joint primitives and DoFs that the various Joint blocks provide.

Joint Block	Joint Primitives								
6-DOF Joint	Px	Py	Pz	.	.	.	S	.	.
Bearing Joint	.	.	Pz	Rx	Ry	Rz	.	.	.
Bushing Joint	Px	Py	Pz	Rx	Ry	Rz	.	.	.
Cartesian Joint	Px	Py	Pz
Constant Velocity Joint	CV	.
Cylindrical Joint	.	.	Pz	.	.	Rz	.	.	.
Gimbal Joint	.	.	.	Rx	Ry	Rz	.	.	.
Leadscrew Joint	LSz
Pin Slot Joint	Px	Rz	.	.	.
Planar Joint	Px	Py	.	.	.	Rz	.	.	.
Prismatic Joint	.	.	Pz
Rectangular Joint	Px	Py
Revolute Joint	Rz	.	.	.
Spherical Joint	S	.	.
Telescoping Joint	.	.	Pz	.	.	.	S	.	.
Universal Joint	.	.	.	Rx	Ry
Weld Joint

Why use Joint blocks with spherical primitives? Those with three revolute primitives are susceptible to gimbal lock—the natural but often undesired loss of one rotational DoF when any two rotation axes become aligned. Gimbal lock leads to simulation errors due to numerical singularities. Spherical primitives eliminate the risk of gimbal-lock errors by representing 3-D rotations using 4-D quantities known as quaternions.

Joint Inertia

Simscape Multibody joints are idealized. They differ from real joints in that they have no inertia—a suitable approximation in most models, where the impact of joint inertia on system dynamics is often

negligible. This is the case, for example, in the constant-velocity joints of automobile driveline systems, where shaft inertia can dwarf joint inertia.

If joint inertia is important in your model, you can account for it using Solid or Inertia blocks. Connect the block reference frame ports to the appropriate joint frames and specify the joint inertial properties in the block dialog boxes. You can specify joint mass or density, products of inertia, moments of inertia, and center of mass. For more information on how to specify inertia, see “Representing Solid Inertia” on page 1-62.

See Also

More About

- “Representing Solid Inertia” on page 1-62

How Multibody Assembly Works

In this section...

“Model Assembly” on page 2-8

“Connecting Joints” on page 2-8

“Orienting Joints” on page 2-9

“Guiding Assembly” on page 2-9

“Verifying Model Assembly” on page 2-10

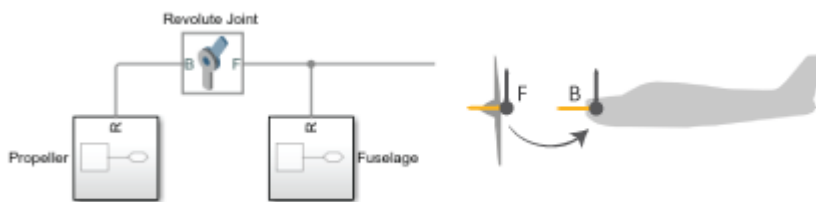
Model Assembly

You model an articulated system by interconnecting bodies through joints and occasionally gears and other constraints. Bodies contribute their inertias to the model, while joints, gears, and constraints determine the relative degrees of freedom that exist between the bodies. You interconnect the two component types by linking frame ports on Joint, Gear, and Constraint blocks to frame ports on body subsystems.

Simscape Multibody automatically assembles your model when you update the block diagram. During model update, Simscape Multibody determines the initial states of joints—their positions and velocities—so that the resulting assembly satisfies all kinematic constraints in the model. This process occurs in two phases, with the assembly algorithm first computing the joint positions and then the joint velocities. The complete process is called model assembly.

Connecting Joints

Joints connect to bodies through frames. Each Joint block contains two frame ports, base (B) and follower (F), identifying the connection points in the adjoining bodies and the relative directions they can move in. When you connect these ports to frames in the body subsystems, you determine how the bodies themselves connect upon model assembly.



Joint Frames Identifying Connection Points and Rotation Axis of Aircraft Propeller

If a joint has no actuation and no sensing outputs, its frame ports are fully interchangeable. In this case, you can switch the bodies that the ports connect to without affecting model dynamics or joint sensing outputs. If the joint does have actuation inputs or sensing outputs, you may need to reverse the actuation or sensing signals to obtain the same dynamic behavior and simulation results.

To change the connection points of a joint, you must modify the connection frames in the adjoining body subsystems. You do this by specifying a translation transform using a Rigid Transform block. You can add new Rigid Transform blocks to the body subsystems or, if appropriate, change the translation transforms in existing Rigid Transform subsystems.

For more information on how Simscape Multibody software interprets frame ports, nodes, and lines, see “Working with Frames” on page 1-24.

Orienting Joints

To obtain the motion expected in a model, you must align its various joint motion axes properly. This means aligning the joints themselves as observed or anticipated in the real system. Misaligning the joint axes may lead to unexpected motion but it often leads to something more serious, such as a failure to assemble and simulate.

You can specify and change joint alignment by rotating the connection frames local to the adjoining body subsystems. For this purpose, you specify rotation transforms using Rigid Transform blocks, either by adding new blocks to the body subsystems or, if appropriate, by changing the rotation transforms in existing blocks within the subsystems.

Why change the orientation of joints through body subsystem frames? The primitives in a Joint block each have a predetermined motion axis, such as x or z . The axis definition is fixed and cannot be changed. Realigning the connection frames local to the adjoining body subsystems provides a natural way to reorient joints while avoiding confusion over which axis a particular joint uses.

Guiding Assembly

Joints can start simulation from different states. For example, the crank joint of a crank-rocker linkage can start at any angle from 0° to 360° . As a result, during model assembly, Simscape Multibody must choose from many equally valid states. You can guide the states chosen by specifying state targets in the Joint block dialog boxes.



Crank-Slider Mechanism in Fully Extended and Fully Retracted Initial Configurations

State targets need not be exact values. If Simscape Multibody cannot achieve a state target exactly, it searches for the joint state nearest to the state target. For example, if you specify a position state target of 60° but the joint can only reach angles of 0° to 45° , Simscape Multibody attempts to assemble the joint at 45° .

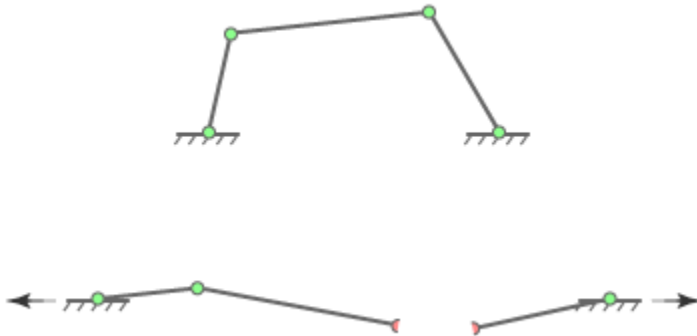
How close the actual joint state is to the state target depends on the kinematic constraints in your model, any conflicts with other state targets, and the state target priority level—a ranking that determines which of two state targets to satisfy if they prove to be mutually incompatible. You can set the priority level to Low or High.

Simscape Multibody first attempts to satisfy all state targets exactly. If a state target conflict arises, Simscape Multibody ignores the low-priority state targets and attempts to satisfy only the high-priority state targets. If a state target conflict still exists, Simscape Multibody ignores also the high-priority state targets and attempts to assemble the model in the nearest valid configuration.

You can specify state targets for all joints in an open kinematic chain. However, to avoid simulation errors, every closed chain must contain at least one joint without state targets.

Verifying Model Assembly

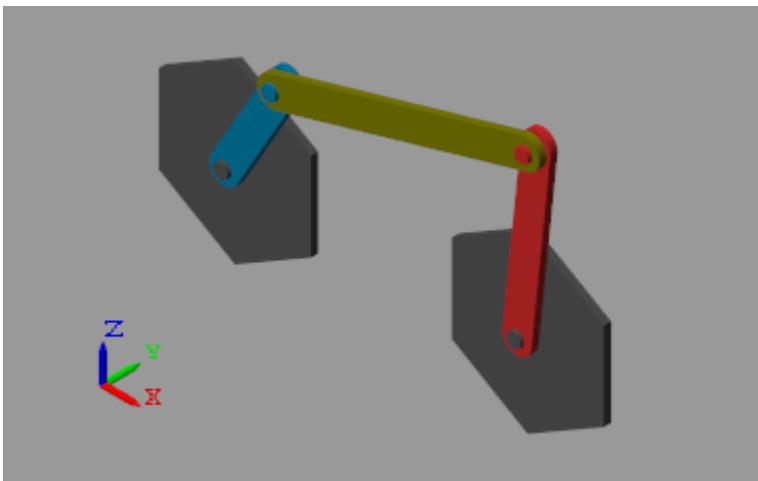
A model assembles successfully only if the connections between its bodies are congruous with each other. If in satisfying one kinematic constraint, Simscape Multibody must violate another kinematic constraint, the model is kinematically invalid and assembly fails. This happens, for example, when the ground link of a four-bar assembly exceeds the combined length of the remaining three links, preventing at least one joint from assembling.



Joint Assembly Failure in Four-Bar Linkage with Exceedingly Long Ground Link

To ensure that your model has assembled correctly, use these Simscape Multibody and Simscape utilities:

- **Mechanics Explorer** — Simscape Multibody visualization utility. Visually examine your model from different points of view to ensure that its bodies connect at the expected locations and with the proper orientations.



- **Variable Viewer** — Simscape state-reporting utility. Check the assembly status of individual joints and constraints and compare your state targets to the actual joint states achieved during assembly.

Name	Status	Priority	Target	Start	Unit
Base_Crank_Revolute	●				
Rz	●				
q	●	High	150.0	150.0	deg
w	●	High	-360.0	-360.0	deg/s
Base_Rocker_Revolute	●				
Rz	●				
q	●			3.0338	rad
w	●			-3.13757	rad/s
Connector_Rocker_Revolute	●				
Rz	●				
q	●			1.1814	rad
w	●			-4.35682	rad/s
Crank_Connector_Revolute	▲				
Rz	▲				
q	▲	Low	-45.0	-43.86525496465046	deg
w	●			7.50244	rad/s

▲ All high priority targets satisfied but some low priority targets not satisfied Variables at start

- Statistics Viewer — Simscape metrics-reporting utility. Check, among other metrics, the degrees of freedom, number of joints, and number of constraints in your model.

Name	Value
3-D Multibody System	
Number of rigidly connected components (excluding ...	3
Number of joints (total)	4
Number of explicit tree joints	3
Number of implicit 6-DOF tree joints	0
Number of cut joints	1
Number of constraints	0
Number of tree degrees of freedom	3
Number of position constraint equations (total)	5
Number of position constraint equations (non-redund...	2
Number of mechanism degrees of freedom (minimum)	1
State vector size	8
Average kinematic loop length	4

Counting Degrees of Freedom

The number and types of joints, gears, and constraints in a mechanism partially determine its mobility—the total number of degrees of freedom, or DoFs, that the mechanism provides and therefore the minimum number of input variables needed to fully constrain its configuration. The mobility F of a mechanism with N bodies and j joints, each with f DoFs follows from expressions such as the Kutzbach criterion, which for a planar mechanism states:

$$F = 3(N - 1) - \sum_{i=1}^j (3 - f_i)$$

Applying this criterion to a four-bar linkage, an assembly of four bodies ($n = 4$) and four joints ($j = 4$) with one rotational DoF each ($f_1 = 1$), yields a mobility of one DoF—indicating that a single input variable suffices to fully control the linkage configuration. As mechanisms grow in complexity, manually calculating total DoFs becomes more time-consuming, so Simscape Multibody automatically computes them for you.

You can view the mechanism DoFs through the Simscape Statistics Viewer, shown below for the four-bar featured example. You open the Statistics Viewer. In the **Apps** gallery, click **Simscape Variable Viewer**. Enter `sm_four_bar` at the MATLAB command prompt to open the four-bar model and view its DoFs through the Statistics Viewer.

Name	Value
Number of implicit 6-DOF tree joints	0
Number of cut joints	1
Number of constraints	0
Number of tree degrees of freedom	3
Number of position constraint equations (total)	5
Number of position constraint equations (non-redundant)	2
Number of mechanism degrees of freedom (minimum)	1
State vector size	8
Average kinematic loop length	4

Model an Open-Loop Kinematic Chain

In this section...

“Model Overview” on page 2-13

“Build Model” on page 2-13

“Guide Model Assembly” on page 2-14

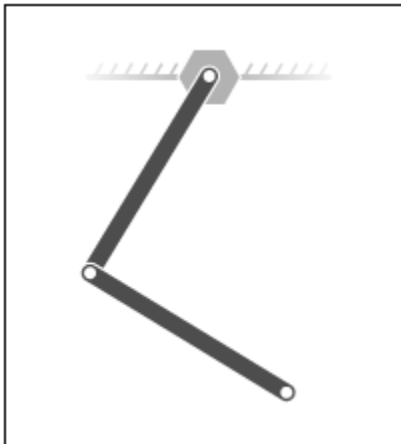
“Visualize Model and Check Assembly Status” on page 2-14

“Simulate Model” on page 2-15

“Open Reference Model” on page 2-15

Model Overview

This example shows how to model a double pendulum—a simple kinematic chain comprising two moving bodies connected in series via two revolute joints. A third body represents a mechanical ground and is rigidly connected to the inertial World frame. The custom `smdoc_compound_rigid_bodies` library provides the body subsystem blocks used in the example.



Revolute Joint blocks enable you to model the joints connecting adjacent bodies and help set their initial states. Simscape Multibody software satisfies a joint state target precisely if it is kinematically valid and not in conflict with other state targets. A **Priority** parameter lets you specify which targets to attempt to satisfy first.

Build Model

- 1 Start a new model.
- 2 Drag these blocks into the model. The two Revolute Joint blocks provide the double pendulum two rotational degrees of freedom.

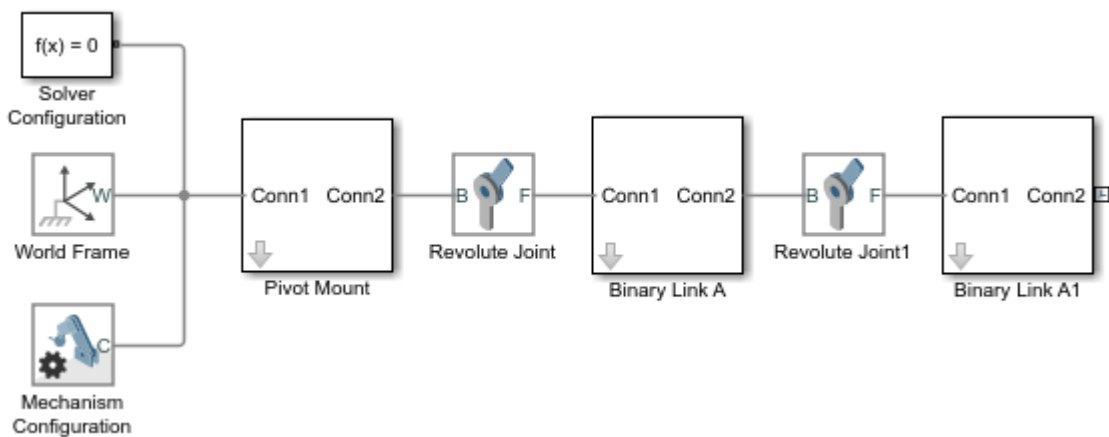
Library	Block	Quantity
Simscape > Utilities	Solver Configuration	1

Library	Block	Quantity
Simscape > Multibody > Utilities	Mechanism Configuration	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Multibody > Joints	Revolute Joint	2

- At the MATLAB command prompt, enter `smdoc_compound_rigid_bodies`. A custom block library with the same name opens up.
- Drag these custom blocks into the model. Each block represents a body in the double pendulum.

Block	Quantity
Pivot Mount	1
Binary Link A	2

- Connect the blocks as shown in the figure.



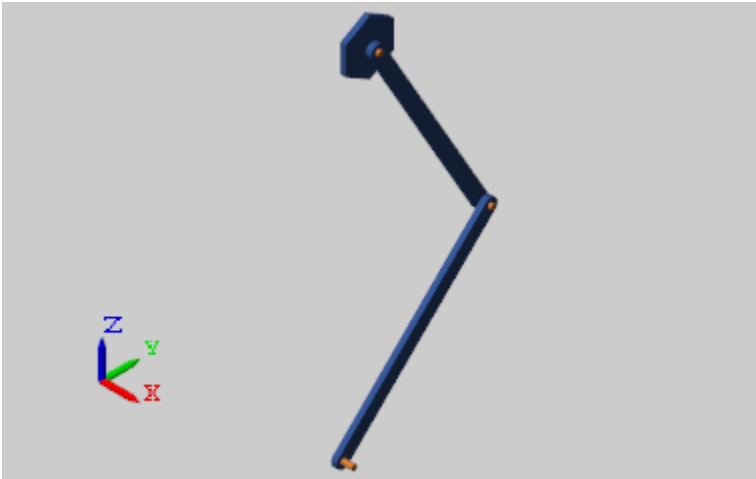
Guide Model Assembly

- In the Revolute Joint block dialog boxes, select **State Targets > Specify Position Target**. You can now specify the desired starting positions of the two joints.
- In **Value**, enter these joint angles.

Block Name	Value (degrees)
Revolute Joint	30
Revolute Joint1	-75

Visualize Model and Check Assembly Status

To visualize the model, update the block diagram. In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a 3-D view of the double pendulum assembly. Click the isometric view button to obtain the perspective in the figure.



To check the assembly status of the revolute joints, use the Model Report utility. You can open this utility from the Mechanics Explorer menu bar by selecting **Tools > Model Report**. The figure shows the assembly information for the double pendulum.

Joint	Asse...	Primit...	Position					Velocity				
			Actual	Specif...	Unit	Priority	Status	Actual	Specif...	Units	Priority	Status
Revolute_Joint	●	Rz	+30	+30	deg	High	●	+0		deg/s		
Revolute_Joint1	●	Rz	-75	-75	deg	High	●	+0		deg/s		

Simulate Model

Run the simulation. Mechanics Explorer shows a 3-D animation of the double pendulum assembly. The assembly moves due to gravity, specified in the Mechanism Configuration block.

Open Reference Model

To see a complete model of the double pendulum assembly, at the MATLAB command prompt enter:

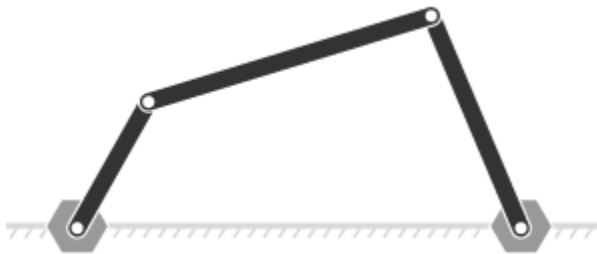
- `smdoc_double_pendulum`

Model a Closed-Loop Kinematic Chain

In this section...

“Build Model” on page 2-16
 “Specify Block Parameters” on page 2-18
 “Guide Assembly and Visualize Model” on page 2-19
 “Verify Model Assembly” on page 2-19
 “Simulate Model” on page 2-20

This example shows how to model a four bar system, which is a closed kinematic chain that comprise four links through revolute joints. One of the links connects to the World Frame block and acts as a ground. Here, use two rigidly connected pivots to replace the link. The custom `smdoc_compound_rigid_bodies` library provides the body subsystem blocks used in the example.



Use Revolute Joint blocks to model the joints that connect adjacent links and set the initial states of the joints. You can satisfy a joint state target precisely if the target is kinematically valid and not in conflict with other state targets. Note that in the **Z Revolute Primitive (Rz) > State Targets** section, you can use the **Priority** parameter to specify the priority level of a target.

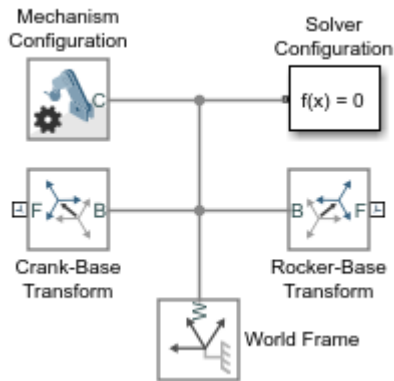
Build Model

To model the four-bar linkage:

- 1 Start a new model.
- 2 Drag these blocks to the model. The Rigid Transform blocks specify the distance between the two pivot mounts. This distance is the length of the implicit ground link.

Library	Block	Quantity
Simscape > Utilities	Solver Configuration	1
Simscape > Multibody > Utilities	Mechanism Configuration	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Multibody > Frames and Transforms	Rigid Transform	2

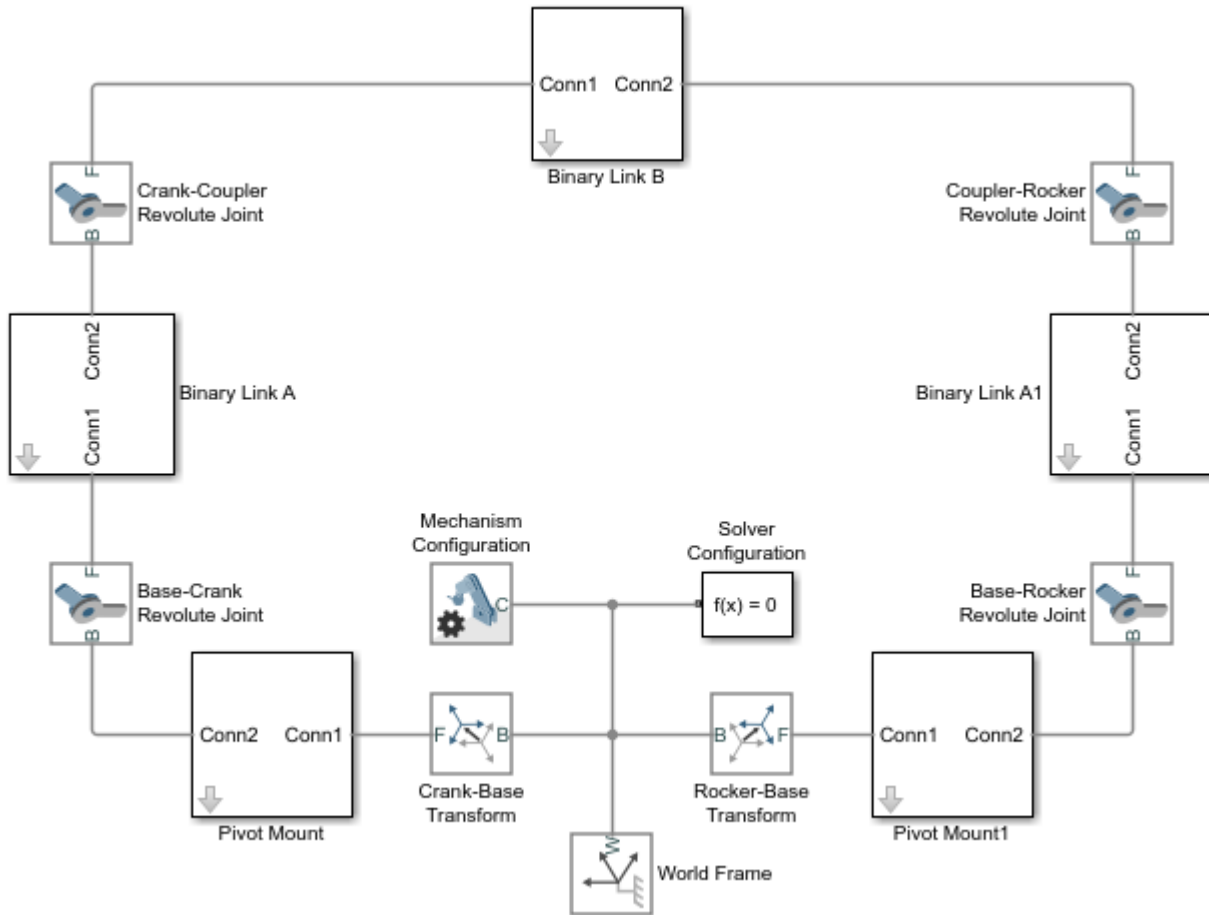
- 3 Connect and name the blocks as shown in the figure. Make sure that the base frame ports of the Rigid Transform blocks connect to the World Frame block.



- 4 From the **Simscape > Multibody > Joints** library, drag four Revolute Joint blocks into the model.
- 5 Enter `smdoc_compound_rigid_bodies` at the MATLAB command prompt. A custom library with compound body subsystem blocks opens up.
- 6 From the `smdoc_compound_rigid_bodies` library, drag the following blocks.

Block	Quantity
Pivot Mount	2
Binary Link A	2
Binary Link B	1

- 7 Connect and name the blocks as shown in the figure. Make sure that you position the frame ports of the custom body subsystem blocks exactly as shown.



Specify Block Parameters

- 1 In the Rigid Transform block dialog boxes, specify the offset between the pivot mounts and the world frame. The pivot mounts are assumed to be symmetrically positioned about this frame.

Parameter	Crank-Base Transform	Rocker-Base Transform
Translation > Method	Standard Axis	Standard Axis
Translation > Axis	-Y	+Y
Translation > Offset	15 in units of cm	15 in units of cm

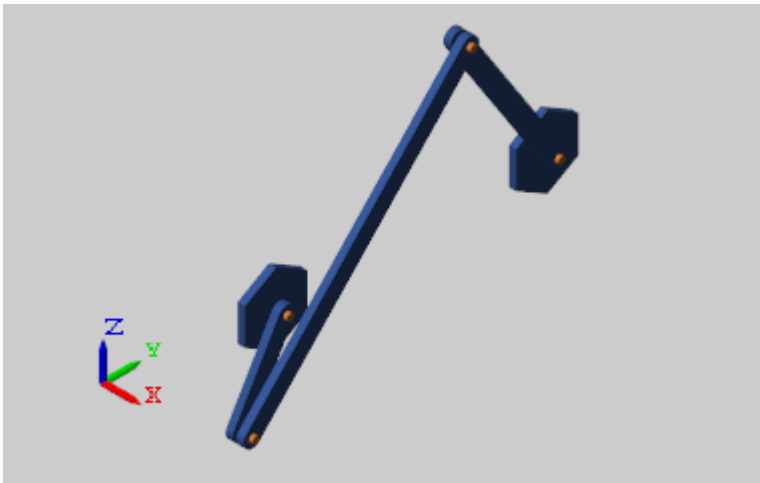
- 2 In each binary link block dialog box, specify the length parameter.

Block	Length (cm)
Binary Link A	10
Binary Link B	35
Binary Link A1	20

Guide Assembly and Visualize Model

Guide model assembly by specifying the desired initial angle for one or more joints in the model. To specify an initial angle of 30° for the Base-Crank joint:

- 1 In the Base-Crank Revolute Joint block dialog box, expand **State Targets** and select **Specify Position Target**.
- 2 Specify the **Value** parameter as -30 deg.
- 3 For the Coupler-Rocker Revolute Joint block, in the **Z Revolute Primitive (Rz) > State Targets** section, select the **Specify Position Target** parameter, and specify the **Value** parameter as 0 deg and **Priority** parameter as Low (approximate).
- 4 On the **Modeling** tab of the Simulink Toolstrip, click **Update Model**. Mechanics Explorer opens with a static display of the four-bar linkage in its initial configuration. The image shows the isometric view of the four-bar linkage.



Verify Model Assembly

To check whether and how precisely the state targets were met, you can use the Simscape Variable Viewer or the Simscape Multibody Model Report.

To open the Simscape Variable Viewer, on the **Debug** tab of the Simulink Toolstrip, in the **Diagnostics** section, select **Simscape > Variable Viewer**. To open the Simscape Multibody Model Report, on the Mechanics Explorer menu bar, select **Tools > Model Report**.

As shown in the image, the yellow marker indicates that the Base-Rocker Revolute Joint state target was satisfied approximately only. The remaining green marker indicates that the Base-Crank Revolute Joint state target was satisfied precisely.

The screenshot shows a 'Model Report - untitled' window with a summary of assembly status and a detailed table of joints. The summary indicates that all joints, constraints, and the overall assembly status are green. The table below provides specific data for four joints: Base_Cran..., Base_Rock..., Coupler_R..., and Crank_Co....

Joint	Assembled	Primitive	Position					Velocity				
			Actual	Specified	Unit	Priority	Status	Actual	Specified	Units	Priority	Status
Base_Cran...	●	Rz	-30	-30	deg	High	●	+0		deg/s		
Base_Rock...	●	Rz	-5.33164	+0	deg	Low	▲	+0		deg/s		
Coupler_R...	●	Rz	+76.5767		deg			+0		deg/s		
Crank_Co...	●	Rz	+101.245		deg			+0		deg/s		

Simulate Model

Run the simulation. Mechanics Explorer shows a 3-D animation of the four-bar assembly. The assembly moves due to gravity.

To see a complete model of the four-bar assembly, enter `smdoc_four_bar` at the MATLAB command prompt.

See Also

PS-Simulink Converter | Revolute Joint

More About

- “Model an Open-Loop Kinematic Chain” on page 2-13

Troubleshoot an Assembly Error

In this section...

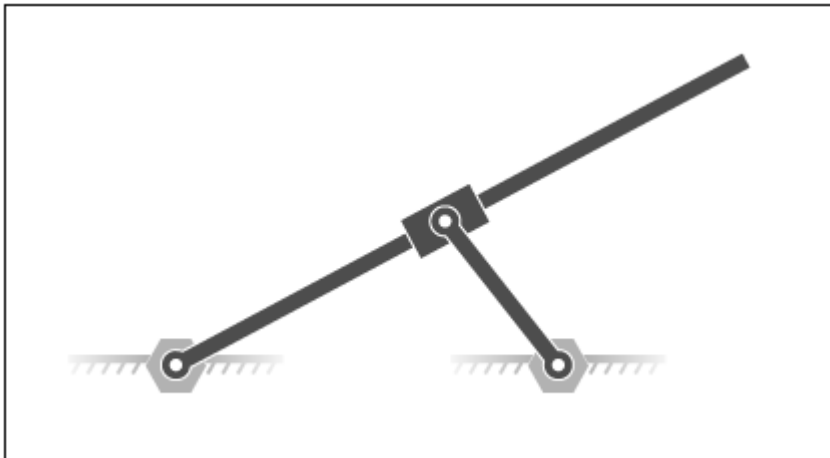
“Model Overview” on page 2-21
 “Explore Model” on page 2-21
 “Update Model” on page 2-23
 “Troubleshoot Assembly Error” on page 2-23
 “Correct Assembly Error” on page 2-25
 “Simulate Model” on page 2-25

Model Overview

In closed-loop systems, joints and constraints must be mutually compatible. For example, in a four-bar linkage, all revolute joints must spin about parallel axes. If one of the joints spins about a different axis, assembly fails and the model does not simulate.

To simplify the troubleshooting process, Simscape Multibody provides Model Report. This tool helps you pinpoint the joints and constraints that caused assembly to fail. Once you identify these joints and constraints, you can then determine which of their frames to correct—and how to correct them.

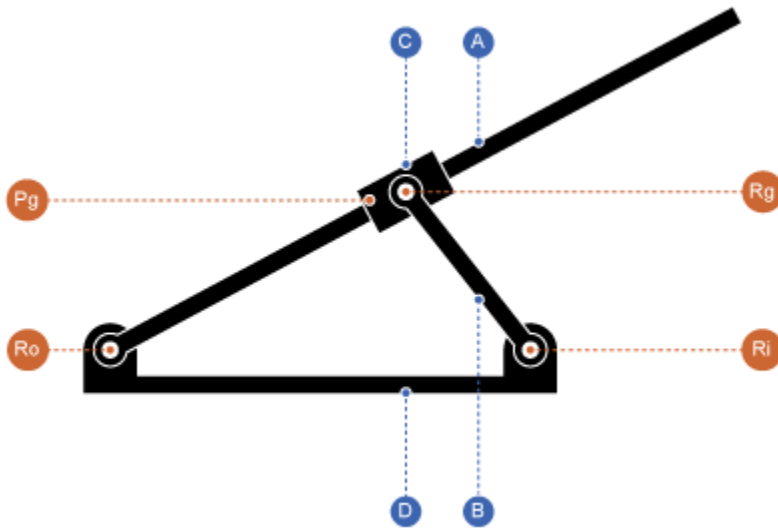
In this example, you identify the assembly error source in an aiming mechanism model using Model Report. Then, using Mechanics Explorer, you determine how to correct that error source. The `sm_dcrankaim_assembly_with_error` featured example provides the basis for this example.



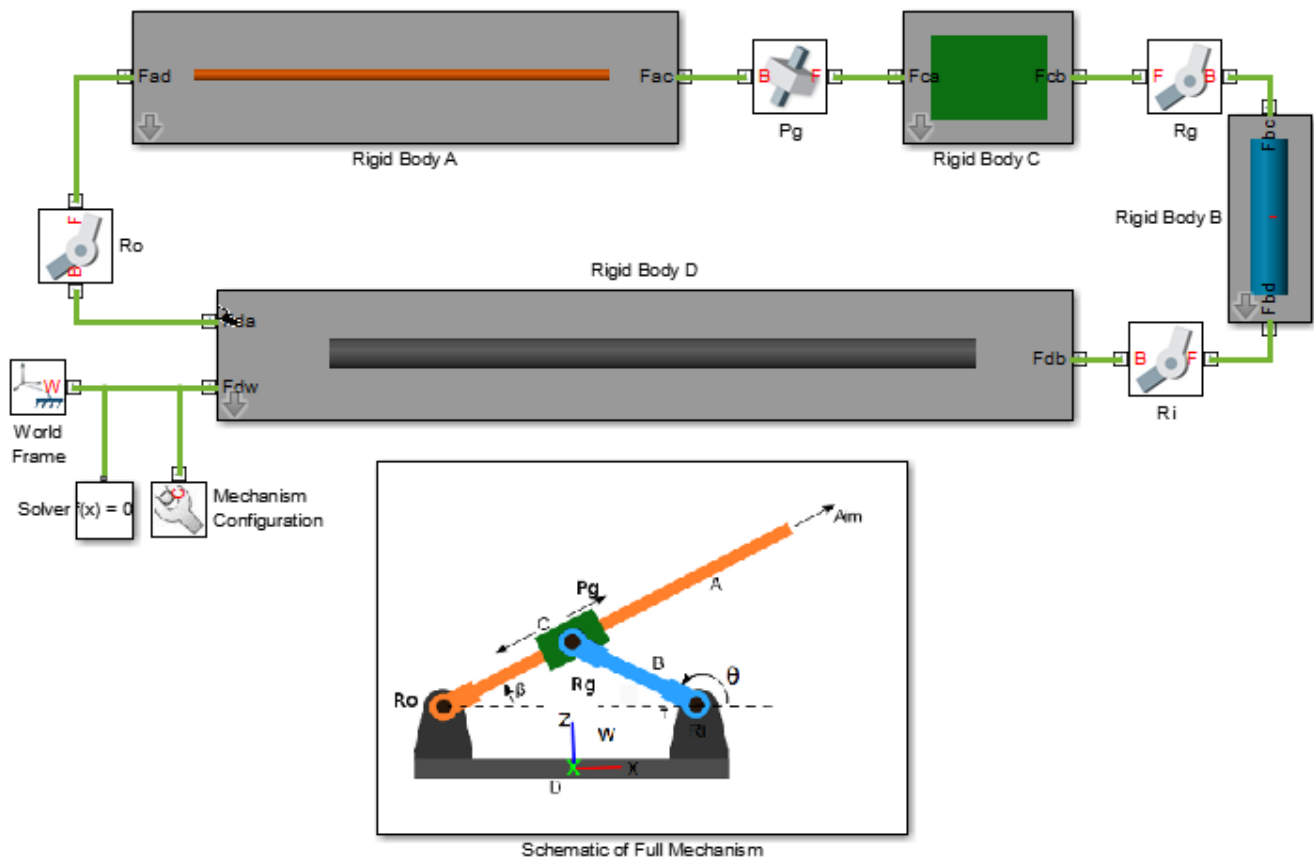
Explore Model

To open the model, at the MATLAB command line, enter `sm_dcrankaim_assembly_with_error`. The model opens in a new window.

The figure shows a schematic of the system that the model represents. This system contains four bodies, labeled A-D. These bodies connect in a closed loop via four joints, labeled Ri, Ro, Rg, and Pg. When connected to each other, these components form a system with one degree of freedom.

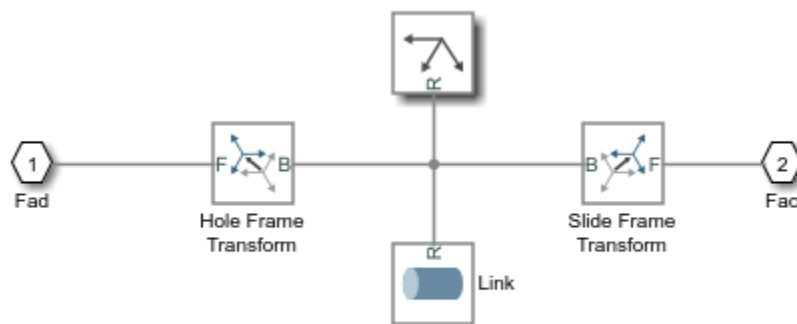


The model represents the components of this system using blocks. Each block represents a physical component. A World Frame block provides the ultimate reference frame in the model. The figure shows the block diagram that the model uses to represent the double-crank aiming mechanism.



To represent the bodies, the model contains four subsystem blocks, labeled Rigid Body A-D. Each subsystem contains one Cylindrical Solid block and multiple Rigid Transform blocks. The Cylindrical Solid block provides geometry, inertia, and color to the body subsystem. The Rigid Transform blocks provide the frames that you connect the joints to. A Reference Frame block identifies the ultimate reference frame in the subsystem block.

The model labels the body subsystem blocks Rigid Body A-D. To examine the block diagram for a body subsystem, right-click the subsystem block and select **Mask > Look Under Mask**. The figure shows the block diagram for Rigid Body A.



Components that make up rigid body A

To represent the joints, the model contains four joint blocks. Three joints provide one rotational degree of freedom between a pair of bodies. You represent each of these joints with a Revolute Joint block. A fourth joint provides one translational degree of freedom between a pair of bodies. You represent this joint with a Prismatic Joint block. The model labels the revolute joint blocks Ro, Rg, and Ri, and the prismatic joint block Pg.

Update Model

As the model name suggests, this model contains an error. The error prevents the model from assembling successfully, which causes simulation to fail. To update the model and investigate the assembly error:

- In the **Modeling** tab, click **Update Model**.

Mechanics Explorer opens with a static display of your model in its initial state. Because the model contains an assembly error, Simscape Multibody issues an error message. Ignore that message for now.

Troubleshoot Assembly Error

Mechanics Explorer provides access to Model Report, a Simscape Multibody utility that summarizes the assembly status of each joint and constraint in a model. Open this utility to determine which joint has failed to assemble. To do this, in the Mechanics Explorer menu bar, select **Tools > Model Report**.

Model Report opens in a new window. A red square indicates that the model, as expected, has failed to assemble. A second red square indicates that an unassembled joint, Pg, is the only contributing

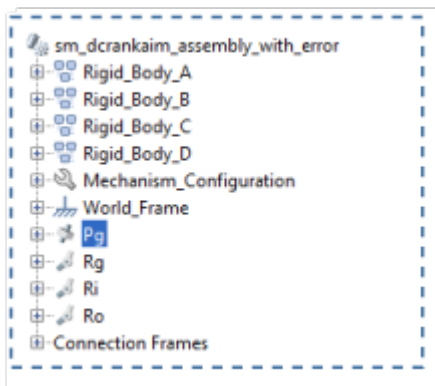
factor in the model assembly error. This information enables you to concentrate your troubleshooting efforts on a small block diagram region—that surrounding the Pg joint block.

Joint	Assembled	Primitive	Position				Velocity						
			Actual	Specified	Unit	Priority	Status	Actual	Specified	Units	Priority	Status	
Pg	■	Pz	N/A		m				N/A		m/s		
Rg	●	Rz	-0.00442103		deg				+0		deg/s		
Ri	●	Rz	+0.00773987		deg				+0		deg/s		
Ro	●	Rz	+0.00331709		deg				+0		deg/s		

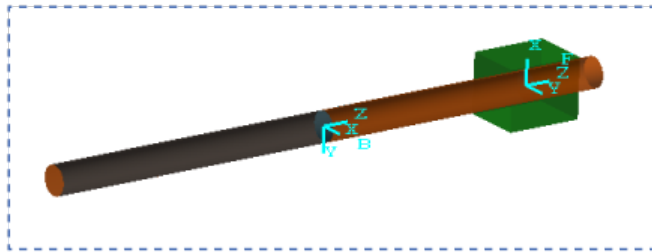
Identifying Error Root Cause

The error message that Simscape Multibody issued during model update identifies position violation as the root cause of assembly failure. This suggests that the frames connected by joint Pg are improperly aligned. To confirm this hypothesis, check the orientation of these frames in Mechanics Explorer.

- 1 In the Mechanics Explorer tree pane, select **Pg**.



- 2 In the Mechanics Explorer visualization pane, examine the position and orientation of the highlighted frames. These are the frames that appear in a light turquoise blue color.



The two frames are offset along the Z axis. This offset is valid, since joint Pg contains a prismatic primitive aligned with the Z axis, providing the frames with one translational degree of freedom along that axis. However, the two frames are also rotated with respect to each other about the common Z axis. This offset is invalid, since joint Pg contains no Revolute or Spherical primitives, and hence no rotational degrees of freedom about any axis. To correct the model assembly error, you must rotate either of the two frames so that all of their axes are parallel to each other.

Correct Assembly Error

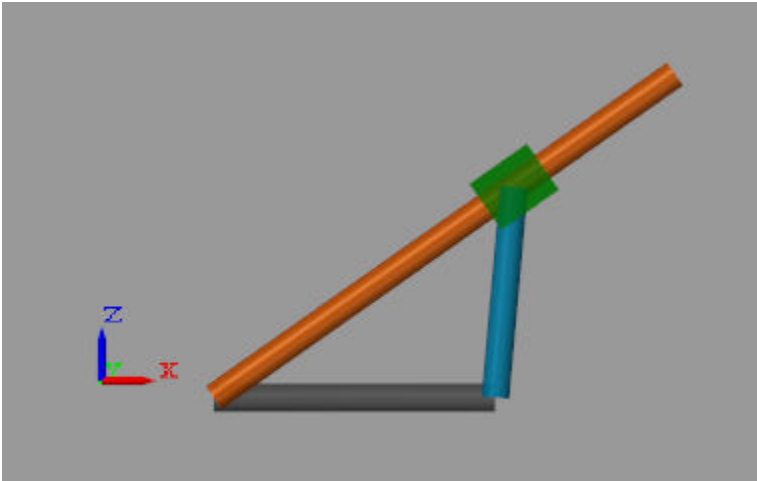
In this example, you apply a rotation transform to the follower frame so that its axes lie parallel to the base frame axes. Alternatively, you could apply an equivalent rotation transform to the base frame. This step enables joint Pg, and hence the model itself, to assemble successfully.

- 1 In the tree pane of Mechanics Explorer, right-click the Pg node and select **Go To Block**. Simscape Multibody brings the block diagram to the front and highlights the Pg block.
- 2 Right-click the Rigid Body C subsystem block and select **Mask > Look Under Mask**.
- 3 Double-click the **Slide Frame Transform** block and select the new parameter values that the table provides. Select **OK**.

Parameter	New Value
Rotation > Pair 2 > Follower	+X
Rotation > Pair 2 > Base	+Y

Simulate Model

You can now simulate the model. Mechanics Explorer opens with a 3-D animation of your model. The figure shows a snapshot of the animation. Rotate, roll, pan, and zoom to explore.



You can use the Model Report tool to verify the assembly status. To do this, in the Mechanics Explorer menu bar, select **Tools > Model Report**. In Model Report, check that the assembly status icons for the model and its joints are green circles. The green circles indicate that the model has assembled correctly.

Model Report - sm_dcrankaim_assembly_with_error

Assembly status: ●
 Joints: ●
 Constraints: ●

Joints | Constraints | Statistics

Joint	Assembled	Primitive	Position		Position		Velocity					
			Actual	Specified	Unit	Priority	Status	Actual	Specified	Units	Priority	Status
Pg	●	Pz	+0.3		m			+0		m/s		
Rg	●	Rz	+0		deg			+0		deg/s		
Ri	●	Rz	+0		deg			+0		deg/s		
Ro	●	Rz	+0		deg			+0		deg/s		

OK

See Also

Related Examples

- “Model an Open-Loop Kinematic Chain” on page 2-13
- “Model a Closed-Loop Kinematic Chain” on page 2-16

More About

- “Modeling Joint Connections” on page 2-4

Modeling Gear Constraints

In this section...

“Gear Constraints and Applications” on page 2-27

“Gear Assemblies as Kinematic Loops” on page 2-28

“Gear Assembly Restrictions” on page 2-29

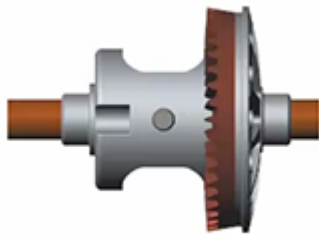
“Gear Pitch Circles” on page 2-30

“Modeling Gear Geometries” on page 2-30

“Limitations of Gear Constraints” on page 2-31

Gear Constraints and Applications

Gear assemblies are ubiquitous in rotating machinery. They appear in couplings and drives, often as gear trains, where they transmit torque at a ratio or at an angle between moving bodies. Some, like rack-and-pinion assemblies, serve special purposes, such as converting between rotational and translational motions.



Gears in an Automotive Differential

The kinematics of gears in mesh arise from what are, in computational terms, algebraic constraints between the gear rotations. Gear teeth cannot physically overlap and the gears must, at a contact point known as the pitch point, move with the same instantaneous linear velocity.

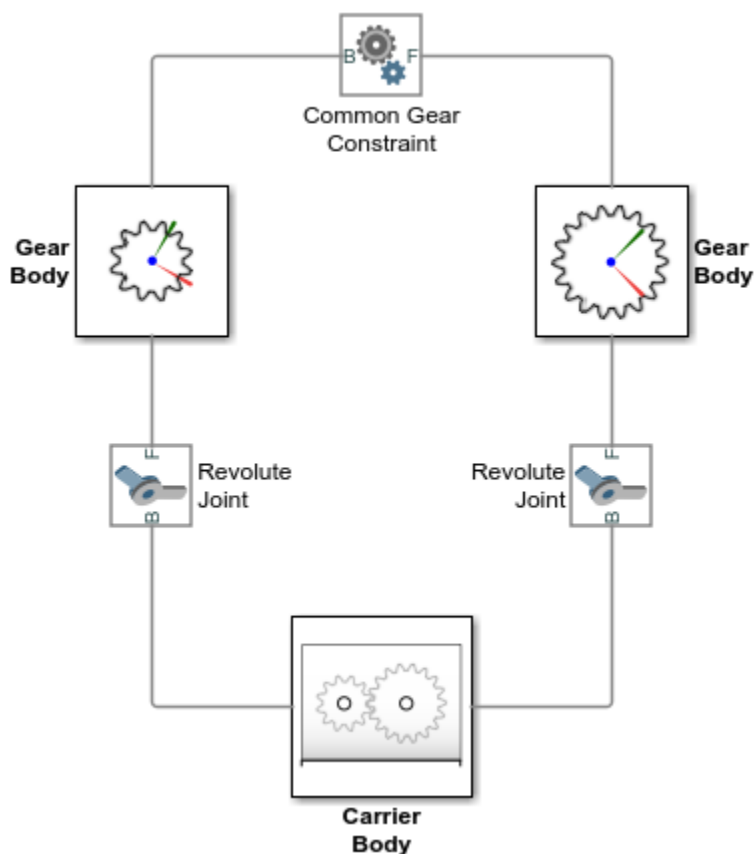
Gear constraint blocks capture the effects of these constraints in a model. The blocks, found in the **Gears and Couplings > Gears** library, include:

- **Bevel Gear Constraint** — Couple two gears, generally conical in cross-section, with intersecting rotation axes meeting at a right or general angle. Bevel gear assemblies are common in the drivetrains of rotorcraft, where they transmit torque between rotor shafts mounted at an angle.
- **Common Gear Constraint** — Couple two gears, generally cylindrical in cross-section, with internal or external meshing and parallel rotation axes. Common gear assemblies appear in automotive transmissions, often as planetary gear trains, that transmit power from engine to wheels at preset torque ratios.
- **Rack and Pinion Constraint** — Couple a rotating pinion to a translating rack with the respective motion axes facing at a right angle. Rack-and-pinion assemblies are common in power steering systems, where they transform a rotation of the steering wheel into a translation of the tie rods, causing the steering arms and wheels to turn.

- **Worm and Gear Constraint** — Couple a worm and a gear with nonintersecting rotation axes facing at a right angle. Worm-and-gear assemblies form the foundation of slew drives built into solar trackers that are designed to follow the sun and maximize the intensity of sunlight striking a solar panel array.

Gear Assemblies as Kinematic Loops

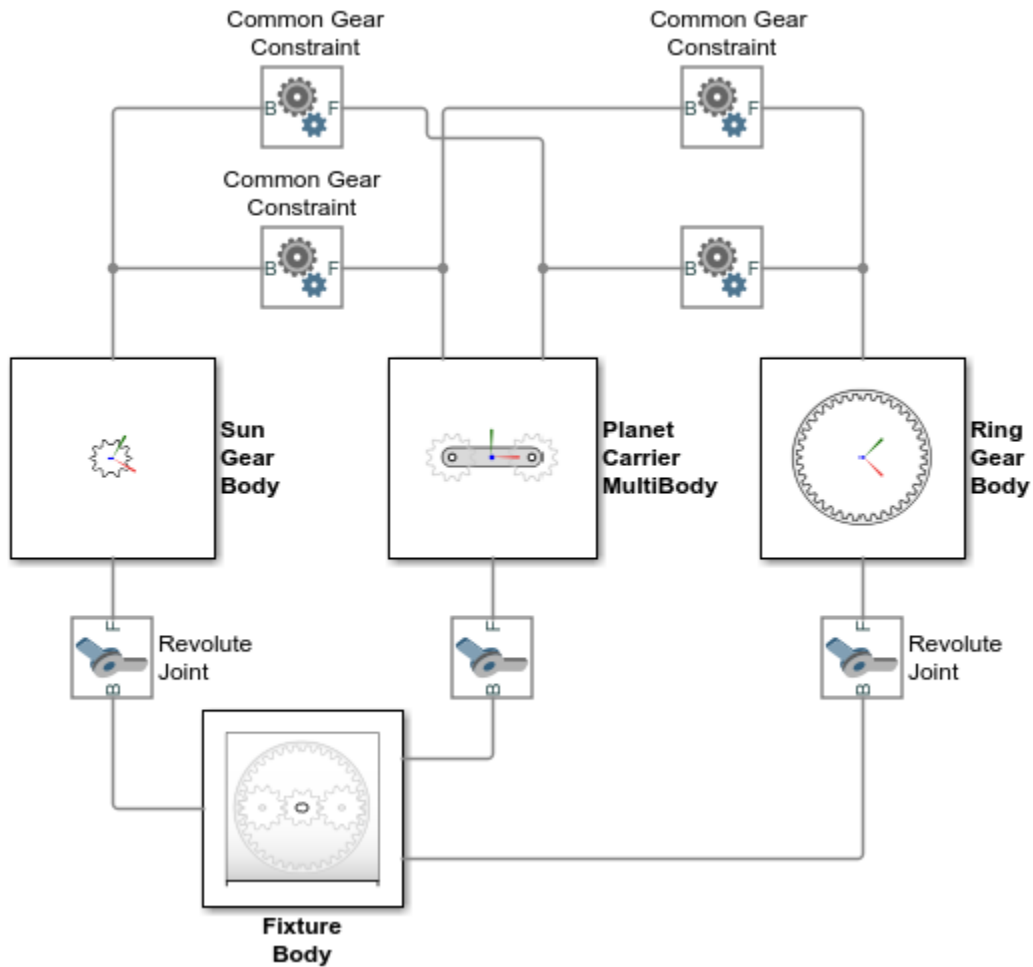
From a topological point of view, gear assemblies form closed kinematic chains, or *loops*. A simple loop comprises two or more gears—the term used loosely here to include worms, pinions, and racks—and a fixture, to hold the gears. The gears connect on one end to the fixture through joints, and on the other end to each other through a gear constraint.



Simple Gear Kinematic Chain

The joints define the degrees of freedom available to the gears before they are brought into mesh. The degrees of freedom encode the types of motion the gears are capable of and the respective motion axes. The gear constraint couples the gears so that they move as though in mesh at a speed ratio determined from the gear (pitch) radii or tooth counts.

More complex model topologies are possible. In a planetary gear train, a ring gear adds a second kinematic loop to the model. Planet gears attached to a carrier add still more kinematic loops. Still, no matter how unique the gear assembly, the model must by its nature comprise at least one kinematic loop.



Planetary Gear Kinematic Loops

Gear Assembly Restrictions

Gear constraints impose special restrictions on the positions and orientations of the gear connection frames. These restrictions are in addition to the meshing constraint, which couples the motions of the gears about the respective rotation axes, and serve to ensure that the gears are always arranged in mesh. For example, the Common Gear Constraint block requires that:

- The distance between the z-axes be equal to the distance between the gear centers.
- The follower frame origin lie on the xy plane of the base frame.
- The z-axes of the base and follower frames point in the same direction.

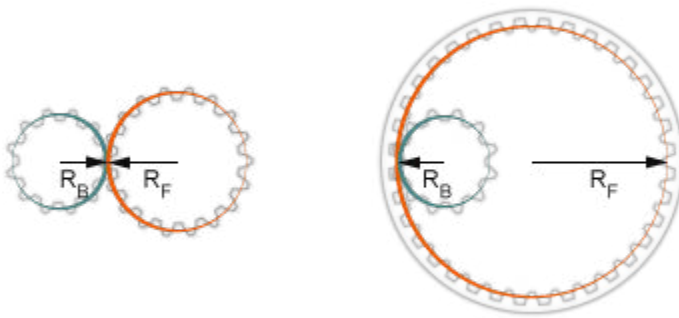
The gear constraint blocks enforce the assembly restrictions, but during model assembly only, when the gears are first placed in mesh. Once simulation starts, it is the task of the model to ensure that the gear placement still satisfies the assembly requirements. The gear constraint blocks then enforce the meshing constraint but merely monitor the assembly constraints, to ensure that the gears remain in a valid configuration.

For examples showing how to properly place the gear connection frames using Rigid Transform blocks, see:

- “Bevel Gear” on page 2-33
- “External Spur Gear” on page 2-36
- “Internal Spur Gear” on page 2-39
- “Rack and Pinion” on page 2-41
- “Worm and Gear” on page 2-44

Gear Pitch Circles

Gear constraints are parameterized in terms of pitch circle dimensions. A pitch circle is an imaginary circle concentric with a gear or worm and tangent to the tooth contact point. Every gear and worm has a pitch circle. The figure shows the pitch circles of spur gears with external and internal meshing. The parameters R_B and R_F denote the gear pitch radii.



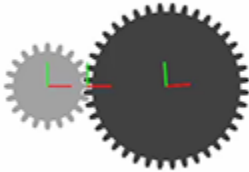
Modeling Gear Geometries

You can approximate gears, worms, and racks using standard solid shapes. Use cylinders with radii equal to the pitch radii for gears and worms. You can use cones for bevel gears and bricks for rack shapes. The figure shows an example with spur gear geometries reduced to cylinders. If you are new to modeling bodies using standard solid shapes, see “Model a Simple Link”.

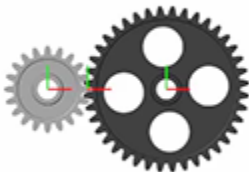


For more detailed geometries, use the Extruded Solid block. This shape enables you to specify the toothed cross-sectional shapes of gears and racks. The Extruded Solid block generates 3-D extrusions by sweeping the cross-sections along their normal axes. The figure shows an example with spur gear

geometries modeled as general extrusions. For an example showing how to model a simple body with a Extruded Solid block, see “Modeling Extrusions and Revolutions” on page 1-44.



For precise geometries, you can load 3-D shapes into File Solid blocks using STEP or STL files. You must obtain the STEP or STL files from external sources. If you have CAD models of gears, worms, and racks, you may be able to export them in STEP or STL format for use in Simscape Multibody software. The figure shows an example with spur gear geometries imported from CAD models via STEP files.



Limitations of Gear Constraints

The physical models provided by the gear constraint blocks are idealized. Gear friction, inertia, and backlash are ignored. You add viscous damping to the gear shafts by specifying damping coefficients in the joint blocks that represent the shaft joints. The shaft joint blocks are typically located between the gear shaft bodies and the gear carrier body. You add inertia to the gears by modeling the gear bodies using the various solid blocks, the Inertia, or General Variable Mass blocks.

See Also

Related Examples

- “Assemble a Gear Model” on page 2-32

Assemble a Gear Model

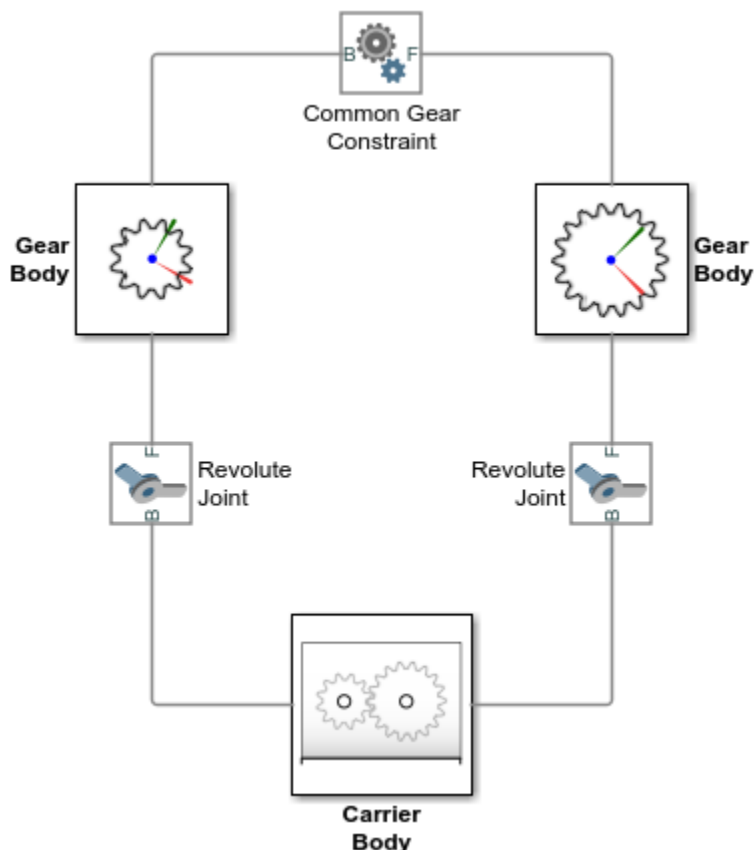
In this section...

“Gear Examples” on page 2-32
 “Bevel Gear” on page 2-33
 “External Spur Gear” on page 2-36
 “Internal Spur Gear” on page 2-39
 “Rack and Pinion” on page 2-41
 “Worm and Gear” on page 2-44

Gear Examples

The examples that follow show how to position and orient gear bodies so that they satisfy the assembly requirements of the various gear constraint blocks. Each example starts with an overview of relevant gear dimensions and frame placements. These attributes guide the selection of rigid transforms needed to ensure that the gears assemble in mesh.

The models share the same block diagram topology, with the model components—bodies, joints, and gear constraint—arranged in a kinematic loop in each case. The figure shows a simple loop. The carrier body is in the examples considered to be fixed to the world frame, with its inertia consequently reduced to a superfluous detail and the body altogether ignored.



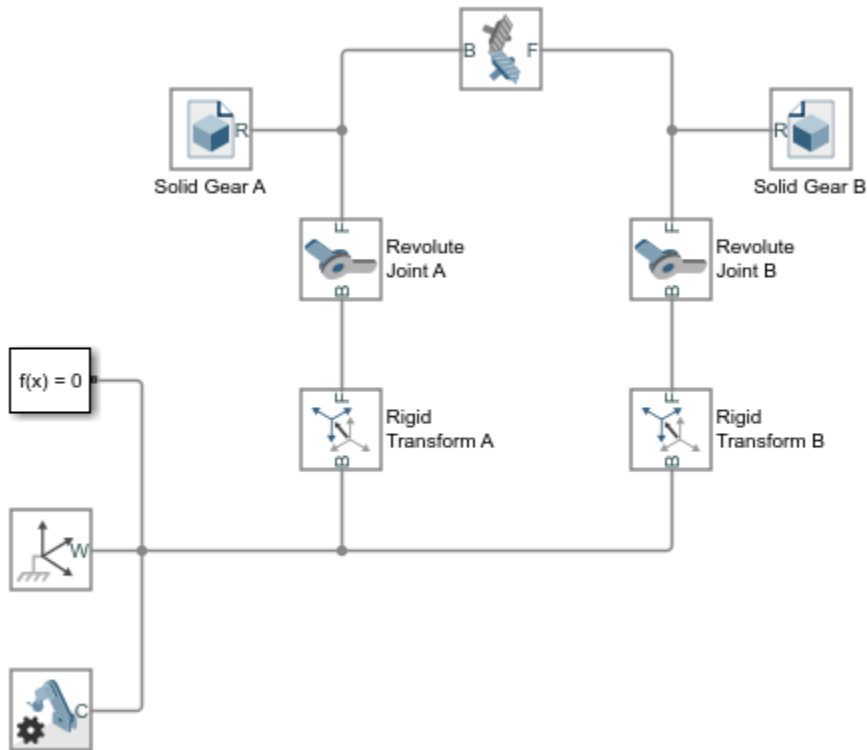
The models comprise four types of Simscape Multibody blocks:

- File Solid — Provides the gear geometries, inertias, and colors. The gear geometries, complete with teeth or threads to more clearly show the gears in mesh, are imported from STEP files. The poses of the gear reference frames relative to the gear geometries are obtained from the same files.
- Joint — Provides the gear bodies with the requisite degrees of freedom. Revolute Joint blocks enable rotation about a single axis. Prismatic Joint blocks enable translation along a single axis. Velocity state targets specified in the joint blocks set the gears in motion.
- Rigid Transform — Rotates and translates the joints and the attached gear bodies so that they are properly placed for meshing. Rigid Transform blocks provide the means to change the gear placements and therefore to satisfy the gear assembly requirements.
- Gear constraint — Couples the motions of the gear bodies. Gear constraint blocks eliminate one degree of freedom between the gears, causing them to move as though in mesh. The examples showcase, one by one, the various gear constraint blocks.

Bevel Gear



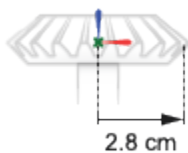
The `smdoc_bevel_gear_start` model, shown in the figure, provides an example of a bevel gear assembly. The model, based on the Bevel Gear Constraint block, is complete in every sense but one— all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

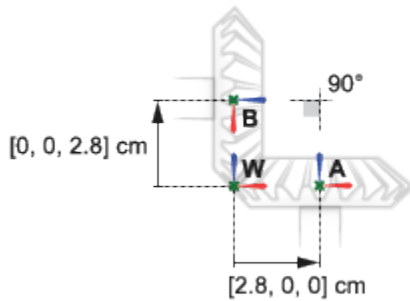
Gear Geometry

The bevel gears, **A** and **B**, are identical in size, with a pitch radius of 2.8 cm in each case. The gear reference frames are placed with origins at the gear centers and z-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint blocks, which allow rotation about the z-axis only.



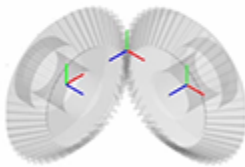
Gear Assembly

The gear rotation axes meet at a right angle. The reference frame of bevel gear **A** sits at an offset of $[2.8, 0, 0]$ cm, in Cartesian coordinates, relative to the world frame. The reference frame of bevel gear **B** sits at an offset of $[0, 0, 2.8]$ cm relative to the world frame and at an angle of 90 deg about the y-axis also of the world frame.



Complete the Model

Complete the bevel gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_bevel_gear_start` at the MATLAB command prompt.

- 1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of bevel gear **A** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[2.8, 0, 0] cm

- 2 In the Rigid Transform B block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of bevel gear **B** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[0, 0, 2.8] cm

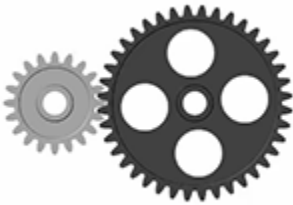
- 3 In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of bevel gear **B** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Standard Axis
Axis	+Y
Angle	90 deg

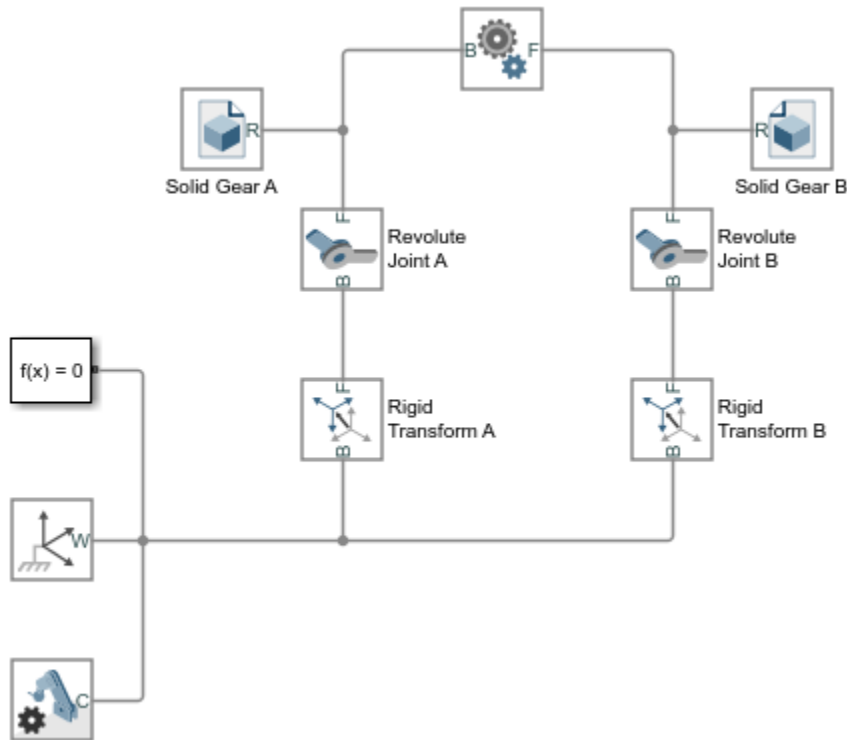
- 4 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete bevel gear model, at the MATLAB command prompt enter `smdoc_bevel_gear`. Simscape Multibody opens a bevel gear model with the rigid transforms described in this example.

External Spur Gear



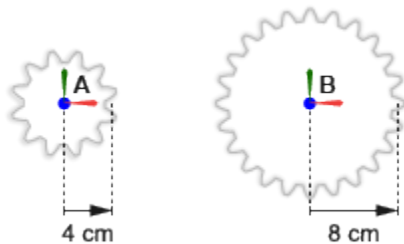
The `smdoc_common_gear_external_start` model, shown in the figure, provides an example of an external spur gear assembly. The model, based on the Common Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

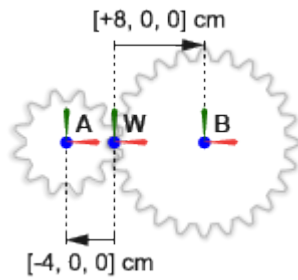
Gear Geometry

The small spur gear, **A**, has a pitch radius of 4 cm. The large spur gear, **B**, has a pitch radius of 8 cm. The gear reference frames are placed with origins at the gear centers and z-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint block, which allows rotation about the z-axis only.



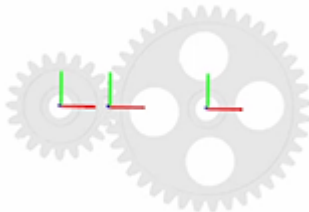
Gear Assembly

The spur gear rotation axes are parallel to each other. The reference frame of the small spur gear sits at an offset of $[-4, 0, 0]$ cm, in Cartesian coordinates, relative to the world frame. The reference frame of the large spur gear sits at an offset of $[-8, 0, 0]$ cm, also relative to the world frame.



Complete the Model

Complete the external spur gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_common_gear_external_start` at the MATLAB command prompt.

- 1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the small spur gear, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	$[-4, 0, 0]$ cm

- 2 In the Rigid Transform B block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the large spur gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	$[8, 0, 0]$ cm

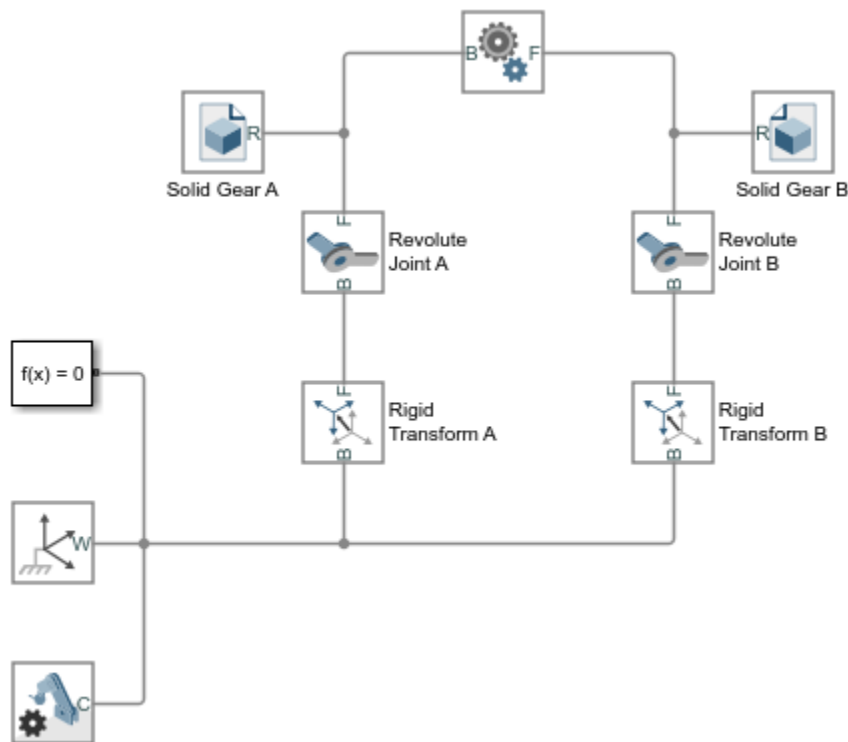
- 3 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete external spur gear model, at the MATLAB command prompt enter `smdoc_common_gear_external`.

Internal Spur Gear



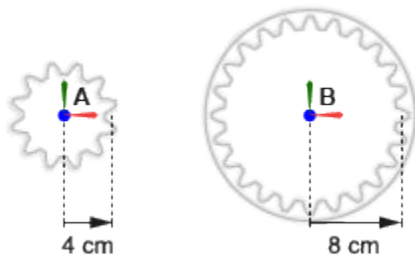
The `smdoc_common_gear_internal_start` model, shown in the figure, provides an example of an internal spur gear assembly. The model, based on the Common Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

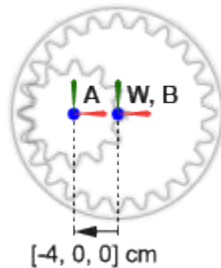
Gear Geometry

The spur gear, **A**, has a pitch radius of 4 cm. The ring gear, **B**, has a pitch radius of 8 cm. The gear reference frames are placed with origins at the gear centers and z-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint block, which allows rotation about the z-axis only.



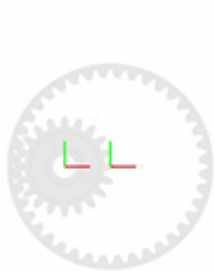
Gear Assembly

The gear rotation axes are parallel to each other. The spur gear reference frame sits at an offset of $[-4, 0, 0]$ cm, in Cartesian notation, relative to the world frame. The ring gear reference frame sits left with its origin and z-axis coincident with those of the world frame.



Complete the Model

Complete the internal spur gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_common_gear_internal_start` at the MATLAB command prompt.

- 1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the spur gear, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[-4, 0, 0] cm

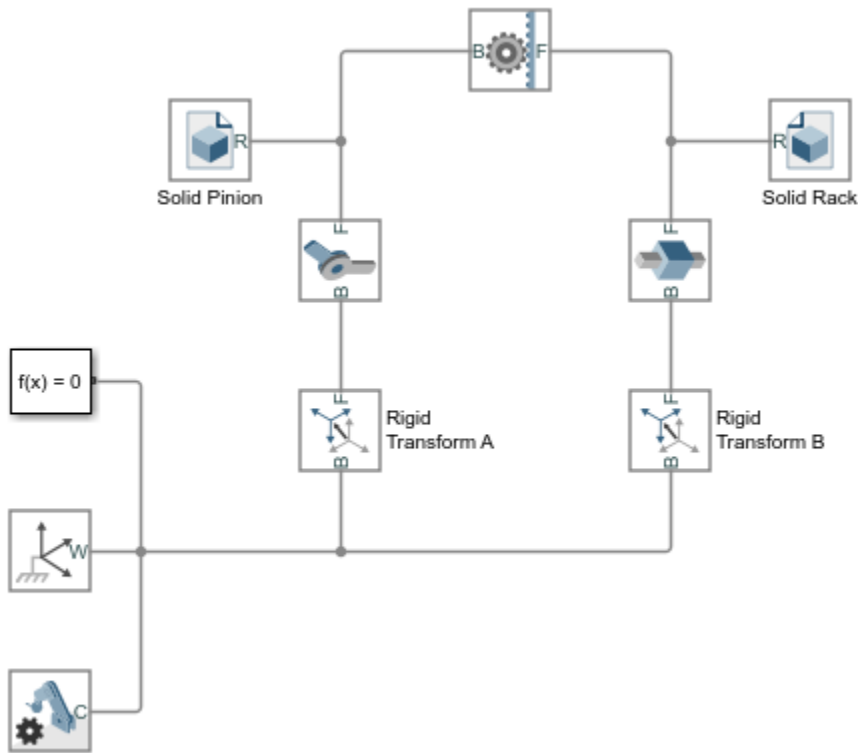
- 2 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete internal spur gear model, at the MATLAB command prompt enter `smdoc_common_gear_internal`.

Rack and Pinion



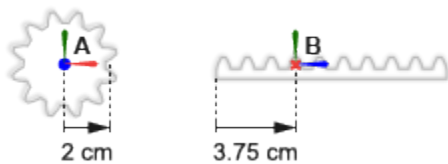
The `smdoc_rack_and_pinion_start` model, shown in the figure, provides an example of a rack-and-pinion assembly. The model, based on the Rack and Pinion Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

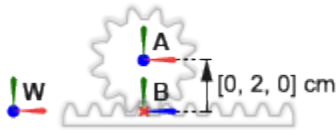
Gear Geometry

The pinion, **A**, has a pitch radius of 2 cm. The pinion reference frame is placed with origin at the pinion center and z-axis along the pinion axis. The rack reference frame is placed with origin 3.75 cm from the rack edge and z-axis along the rack length. The frame alignments are consistent with the Revolute Joint and Prismatic Joint blocks, which allow motion about or along the z-axis only.



Gear Assembly

The rack translation axis is at a right angle to the pinion rotation axis. The pinion reference frame sits at an offset of $[0, 2, 0]$ cm, in Cartesian notation, relative to the world frame. The rack reference frame sits at an angle of 90° deg relative to the positive y-axis of the world frame.



Complete the Model

Complete the rack-and-pinion model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_rack_and_pinion_start` at the MATLAB command prompt.

- 1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the pinion, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[0, 2, 0] cm

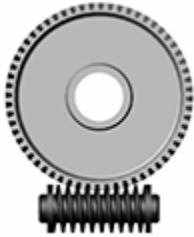
- 2 In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of the rack, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Standard Axis
Axis	+Y
Angle	90 deg

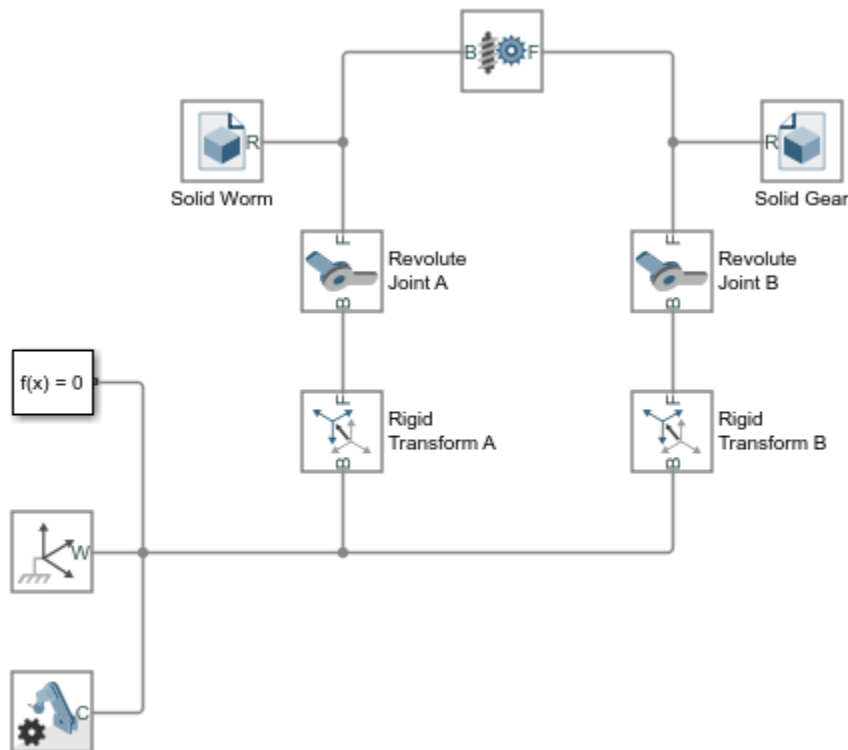
- 3 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete rack-and-pinion model, at the MATLAB command prompt enter `smdoc_rack_and_pinion`.

Worm and Gear



The `smdoc_worm_and_gear_start` model, shown in the figure, provides an example of a worm-and-gear assembly. The model, based on the Worm and Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.

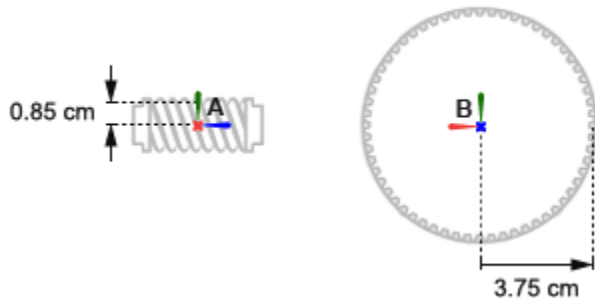


This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

Gear Geometry

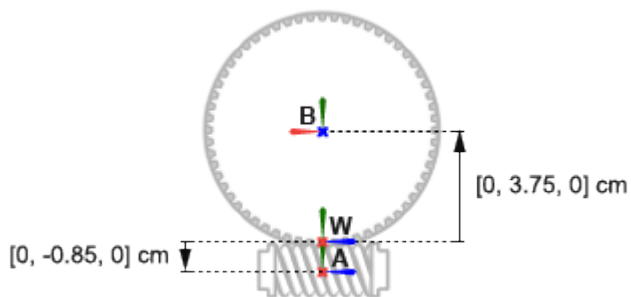
The worm, **A**, has a pitch radius of 0.85 cm. The gear, **B**, has a pitch radius of 3.75 cm. The worm and gear reference frames are placed with origins at the geometry centers and z -axes aligned with

the respective rotation axes. This alignment is consistent with the Revolute Joint block, which allows rotation about the z-axis only.



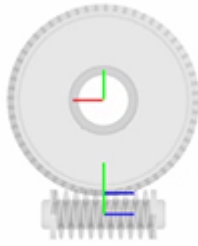
Gear Assembly

The worm rotation axis is at a right angle to the gear rotation axis. The worm reference frame sits at an offset of $[0, -0.85, 0]$ cm, in Cartesian notation, relative to the world frame. The gear reference frame sits at an offset of $[0, +3.75, 0]$ cm and at an angle of 90° deg about the positive y-axis relative to the world frame.



Complete the Model

Complete the worm-and-gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_worm_and_gear_start` at the MATLAB command prompt.

- 1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the worm, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[0, -0.85, 0] cm

- 2 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Cartesian
Offset	[0, 3.75, 0] cm

- 3 In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of the gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	Standard Axis
Axis	+Y
Angle	90 deg

- 4 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete worm-and-gear gear model, at the MATLAB command prompt enter `smdoc_worm_and_gear`.

Model a Compound Gear Train

In this section...

“Model Overview” on page 2-47
“Model Sun-Planet Gear Set” on page 2-47
“Constrain Sun-Planet Gear Motion” on page 2-50
“Add Ring Gear” on page 2-51
“Add Gear Carrier” on page 2-54
“Add More Planet Gears” on page 2-57

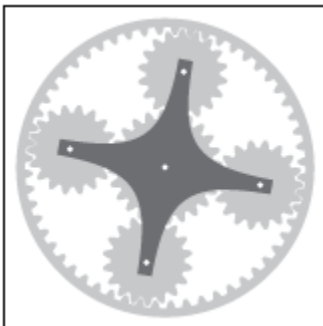
Model Overview

Planetary gear trains are common in industrial, automotive, and aerospace systems. A typical application is the automatic transmission system of car. From a kinematic point of view, what sets this mechanism apart is the kinematic constraint set between gear pairs. These constraints fix the angular velocity ratios of the gear pairs, causing the gears in each pair to move in sync.

In Simscape Multibody, you represent the kinematic constraint between meshed gears using blocks from the Gears sublibrary. This tutorial shows you how to use these blocks to model a planetary gear train. The gear train contains four bodies:

- Sun gear
- Planet gear
- Ring gear
- Planet carrier

Each body, including the planet carrier, can spin about its central axis. In addition, each planet gear can revolve about the sun gear. Joint blocks provide the required degrees of freedom, while gear constraint blocks ensure the gears move as if they were meshed.



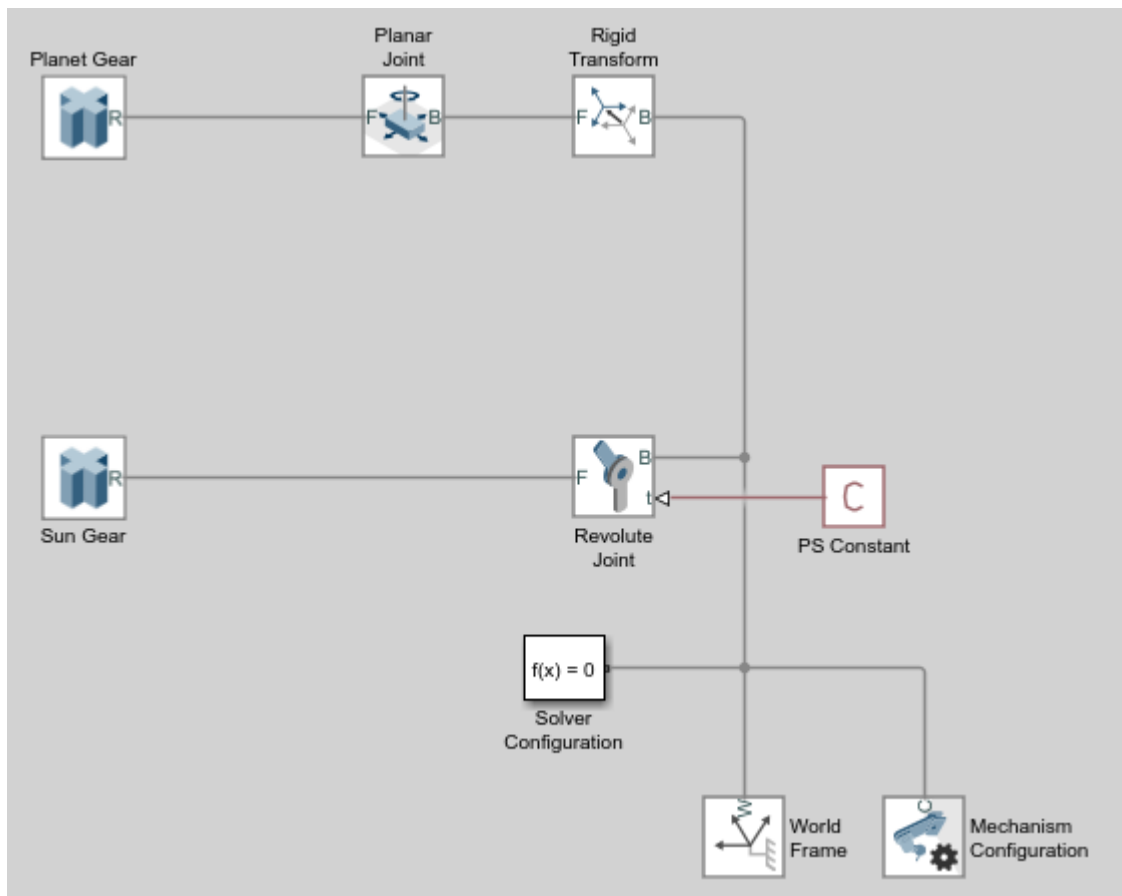
Model Sun-Planet Gear Set

Model the gear bodies and connect them with the proper degrees of freedom. In a later step, you add gear constraints to this model.

- 1 Drag these blocks to a new model.

Library	Block	Quantity
Body Elements	Extruded Solid	2
Joints	Revolute Joint	1
Joints	Planar Joint	1
Frames and Transforms	Rigid Transform	1
Frames and Transforms	World Frame	1
Utilities	Mechanism Configuration	1
Simscape > Utilities	Solver Configuration	1

2 Connect and name the blocks as shown.



3 In the Sun Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Cross-Section	Enter <code>simmechanics.demohelpers.gear_profile(2*Sun.R,Sun.N,A)</code> . Select units of cm.
Geometry > Length	Enter T. Select units of cm.
Inertia > Density	Enter Rho.

Parameter	Setting
Graphic > Visual Properties > Color	Enter Sun.RGB.

The `simmechanics.demohelpers.gear_profile` function generates the cross-section matrix for an external gear with an involute tooth profile. The cross-section is approximate. Use the function as an example only.

- 4 In the Planet Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Cross-Section	Enter <code>simmechanics.demohelpers.gear_profile(2*Planet.R,Planet.N,A)</code> . Select units of cm.
Geometry > Length	Enter T. Select units of cm.
Inertia > Density	Enter Rho.
Graphic > Visual Properties > Color	Enter Planet.RGB.

- 5 In the Rigid Transform block dialog box, specify these parameters.

Parameter	Setting
Translation > Method	Select Standard Axis.
Translation > Axis	Select +Y.
Translation > Offset	Enter <code>Sun.R + Planet.R</code> . Select units of cm.

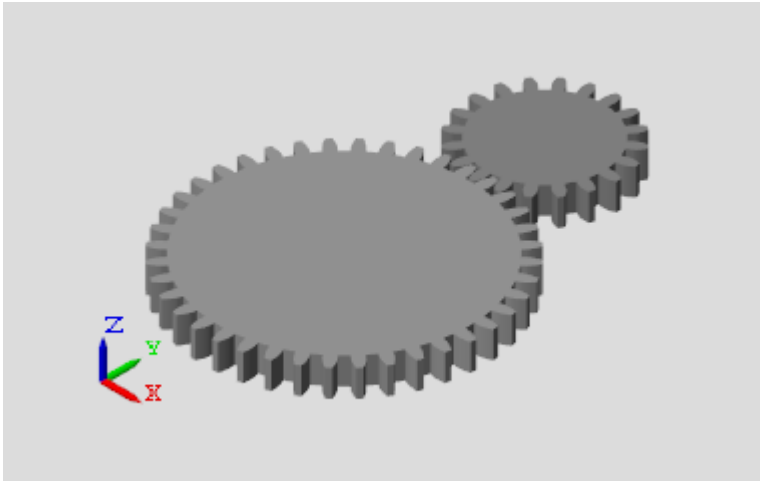
- 6 In the model workspace, define the block parameters using MATLAB code:

```
% Common Parameters
Rho = 2700;
T = 3;
A = 0.8; % Gear Addendum

% Sun Gear Parameters
Sun.RGB = [0.75 0.75 0.75];
Sun.R = 15;
Sun.N = 40;

% Planet Gear Parameters
Planet.RGB = [0.65 0.65 0.65];
Planet.R = 7.5;
Planet.N = Planet.R/Sun.R*Sun.N;
```

- 7 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun and planet gears move independently of each other. To constrain gear motion, you must add a gear constraint block between the gear solid blocks.



You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_a`.

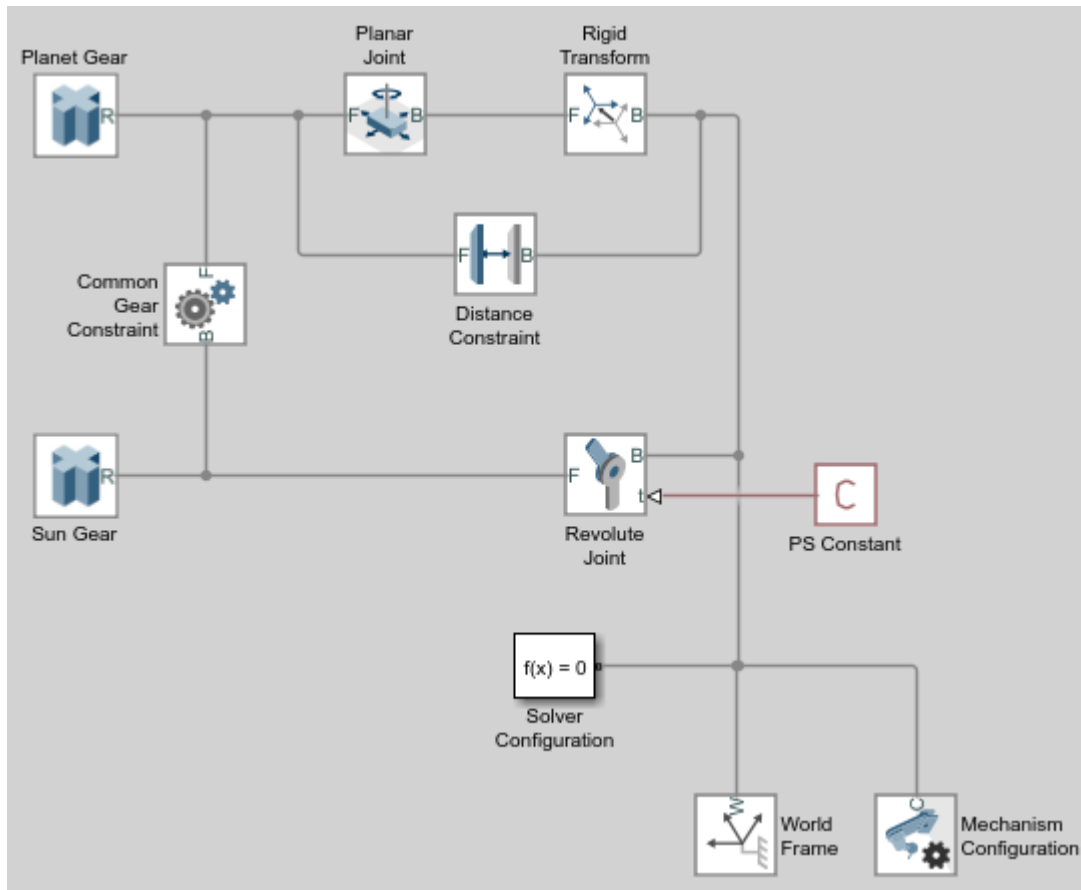
Constrain Sun-Planet Gear Motion

Specify the kinematic constraints acting between the sun and planet gears. These constraints ensure that the gears move in a meshed fashion.

- 1 Drag these blocks to the sun-planet gear model.

Library	Block
Constraints	Distance Constraint
Gears and Couplings > Gears	Common Gear Constraint

- 2 Connect the blocks as shown. The new blocks are highlighted.



- 3 In the Common Gear Constraint block dialog box, specify these parameters.

Parameter	Setting
Specification Method	Select Pitch Circle Radii.
Specification Method > Base Gear Radius	Enter Sun.R. Select units of cm.
Specification Method > Follower Gear Radius	Enter Planet.R. Select units of cm.

- 4 In the Distance Constraint block dialog box, specify this parameter:
- **Distance** — Enter Sun.R + Planet.R. Select units of cm.
- 5 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun and planet gears now move in sync.

You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_b`.

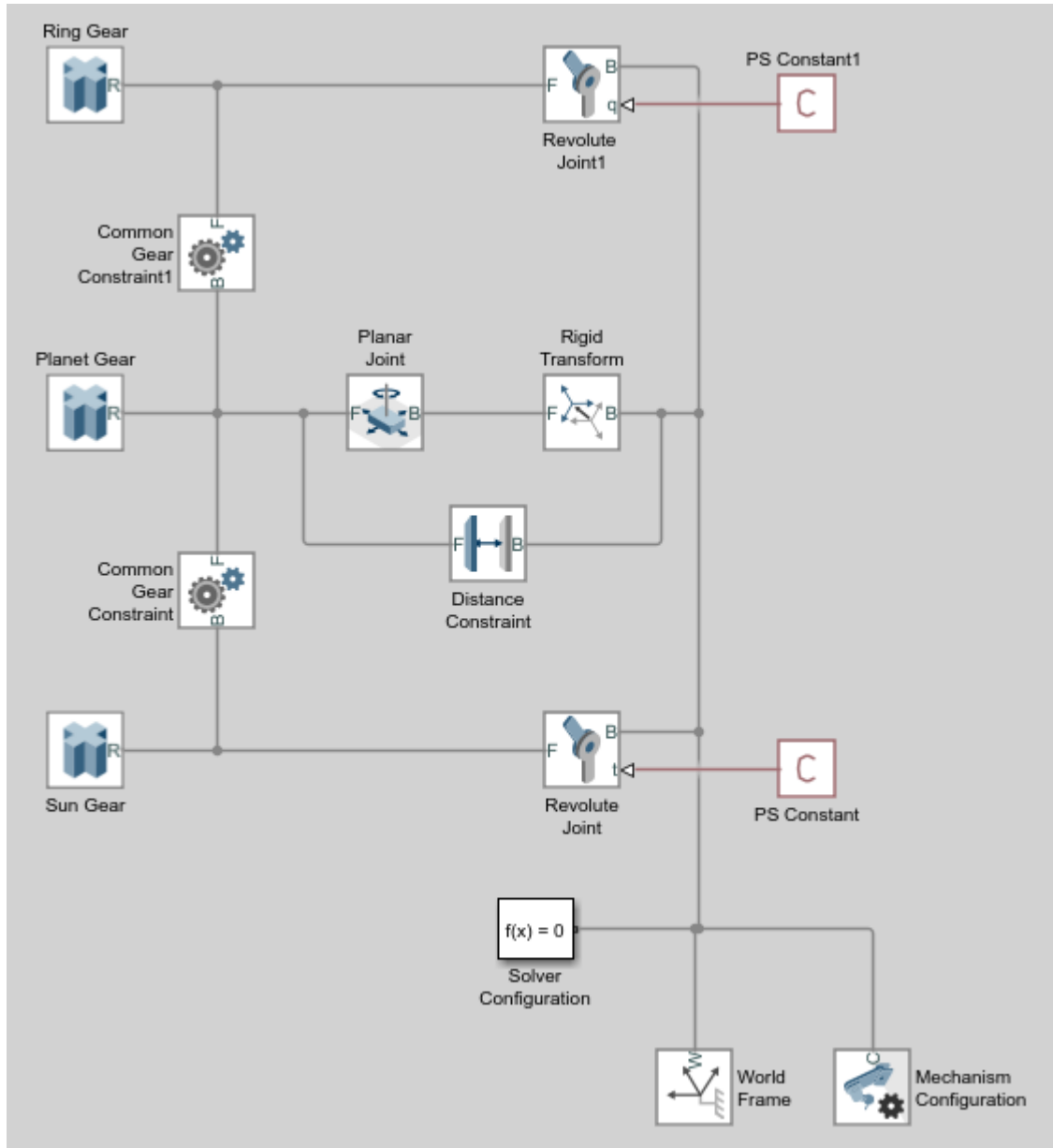
Add Ring Gear

Model the ring gear body, connect it with the proper degrees of freedom, and constrain its motion with respect to the planet gear.

- 1 Add these blocks to the sun-planet gear model.

Library	Block
Body Elements	Extruded Solid
Joints	Revolute Joint
Gears and Couplings > Gears	Common Gear Constraint

- 2 Connect and name the blocks as shown. The new blocks are highlighted.



- 3 In the Ring Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Cross-Section	Enter Ring.CS. Select units of cm.
Geometry > Length	Enter T.
Inertia > Density	Enter Rho.
Graphic > Visual Properties > Color	Enter Ring.RGB.

- 4 In the Common Gear Constraint1 block dialog box, specify these parameters.

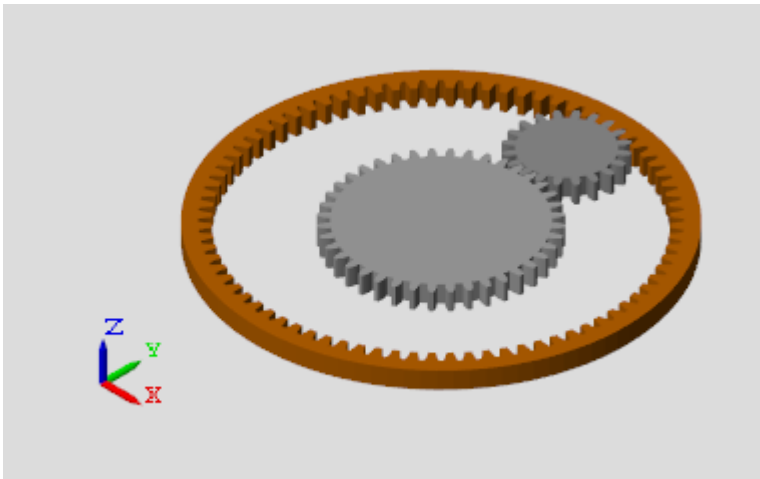
Parameter	Setting
Type	Select Internal.
Specification Method	Select Pitch Circle Radii.
Specification Method > Base Gear Radius	Enter Planet.R. Select units of cm.
Specification Method > Follower Gear Radius	Enter Ring.R. Select units of cm.

- 5 In the model workspace, define the Ring Gear block parameters using MATLAB code:

```
% Ring Gear Parameters
Ring.RGB = [0.85 0.45 0];
Ring.R = Sun.R + 2*Planet.R;
Ring.N = Ring.R/Planet.R*Planet.N;

Ring.Theta = linspace(-pi/Ring.N,2*pi-pi/Ring.N,100)';
Ring.R0 = 1.1*Ring.R;
Ring.CS0 = [Ring.R0*cos(Ring.Theta) Ring.R0*sin(Ring.Theta)];
Ring.CSI = simmechanics.demohelpers.gear_profile(2*Ring.R, Ring.N, A);
Ring.CSI = [Ring.CSI; Ring.CSI(1,:)];
Ring.CS = [Ring.CS0; flipud(Ring.CSI)];
```

- 6 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun, planet, and ring gears move in a meshed fashion.



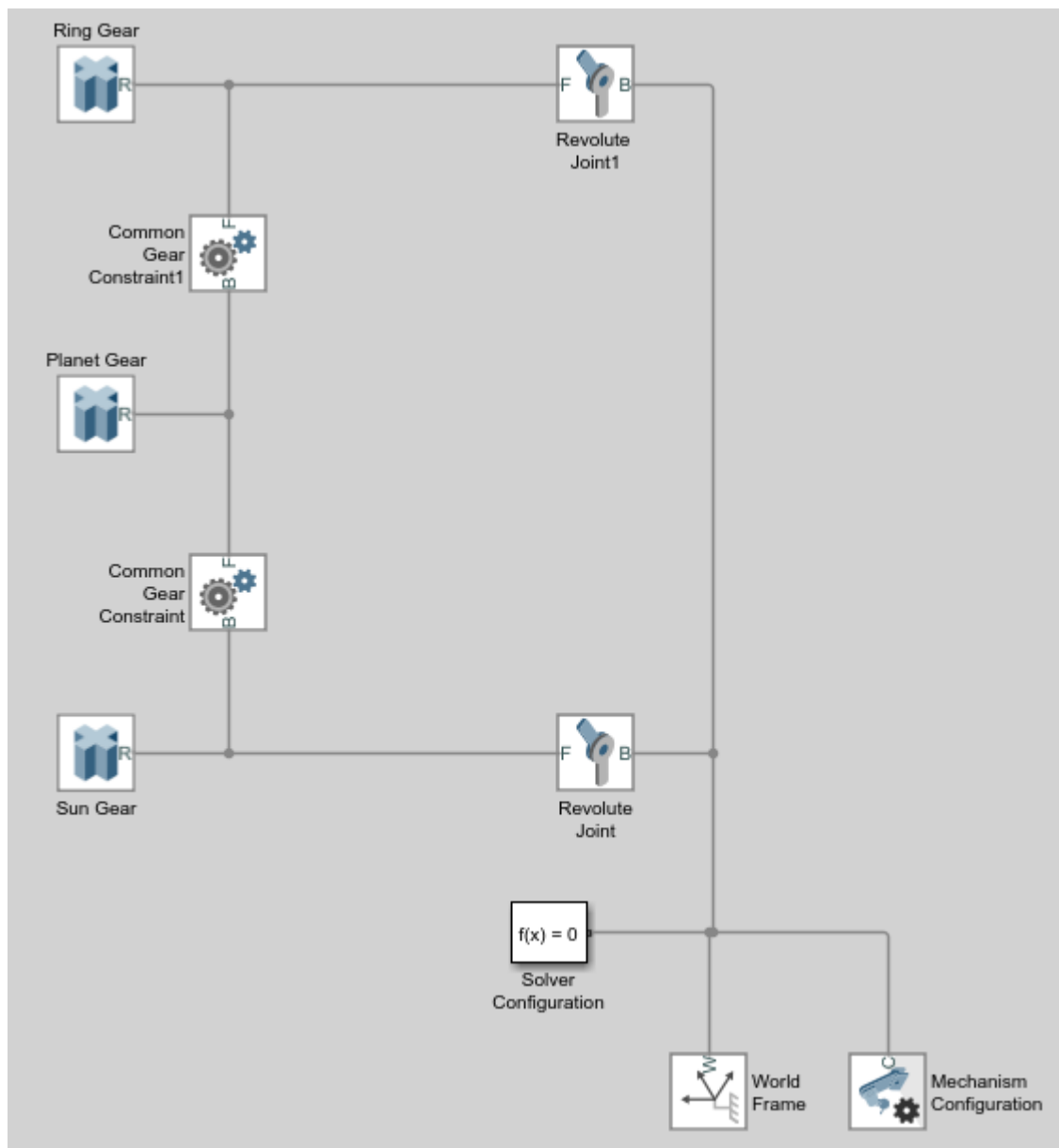
You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_c`.

Add Gear Carrier

Up to now, you have kept the sun and planet gears at a fixed distance using a Distance Constraint block. In an actual planetary gear, a gear carrier enforces this constraint. Model the gear carrier and connect it between the sun and planet gears.

1 Remove these blocks from the planetary gear model:

- Planar Joint
- Rigid Transform
- Distance Constraint

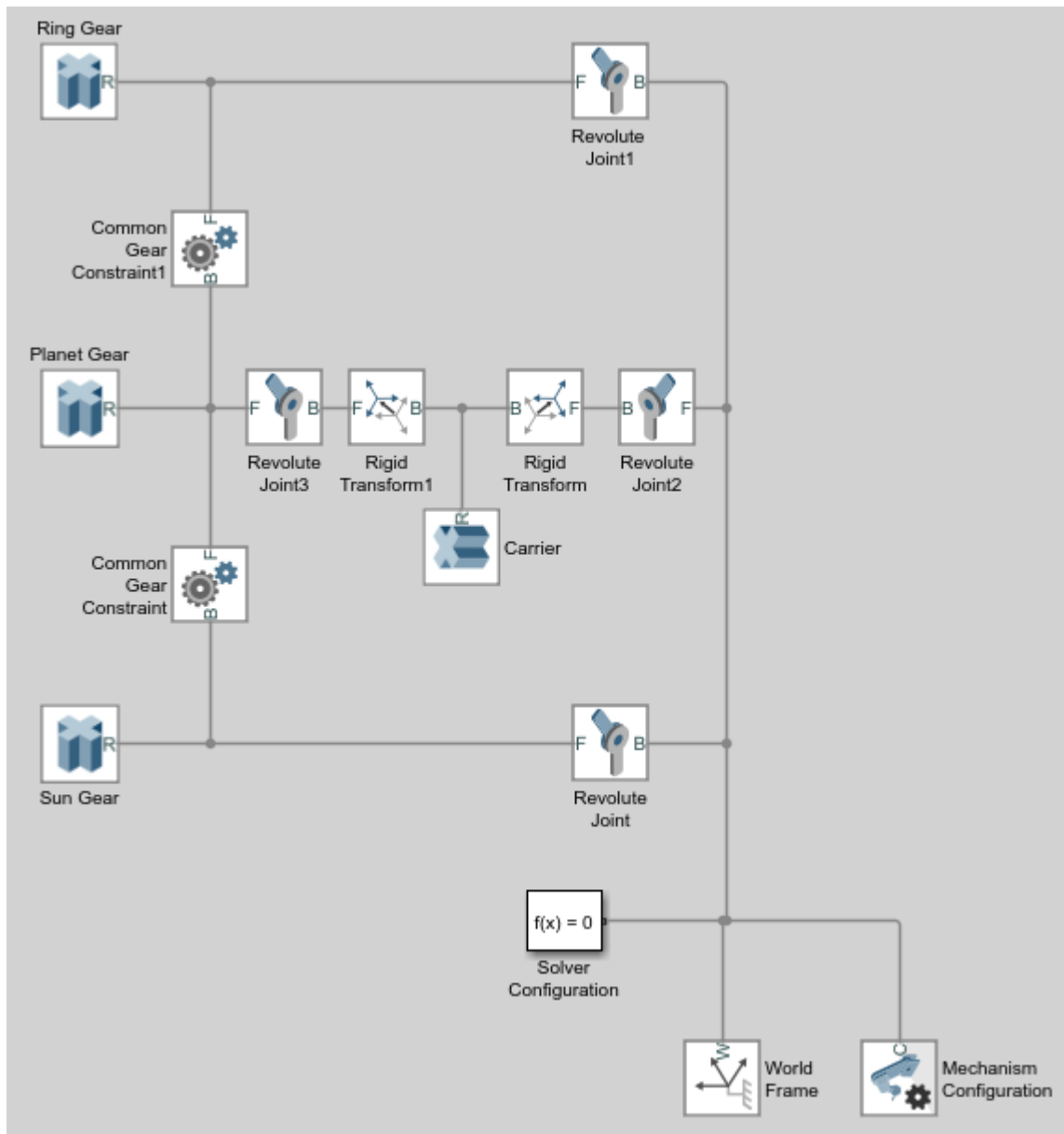


2 Add these blocks to the planetary gear model.

Library	Block	Quantity
Body Elements	Extruded Solid	1
Joints	Revolute Joint	2
Frames and Transforms	Rigid Transform	2

3 Connect and name the blocks as shown.

Pay close attention to the Rigid Transform block orientation: the B frame ports should face the Solid block. The new blocks are highlighted.



4 In the Carrier block dialog box, specify these parameters.

Parameter	Setting
Geometry > Cross-Section	Enter Carrier.CS. Select units of cm.
Geometry > Length	Enter Carrier.T.
Inertia > Density	Enter Rho.
Graphic > Visual Properties > Color	Enter Carrier.RGB.

- 5 In the Rigid Transform block dialog box, specify these parameters.

Parameter	Setting
Translation > Method	Select Cartesian.
Translation > Offset	Enter $[\text{Carrier.L}/2 \ 0 \ -(\text{Carrier.T} + \text{T})/2]$. Select units of cm.

- 6 In the Rigid Transform1 block dialog box, specify these parameters.

Parameter	Setting
Translation > Method	Select Cartesian.
Translation > Offset	Enter $[-\text{Carrier.L}/2 \ 0 \ -(\text{Carrier.T} + \text{T})/2]$. Select units of cm.

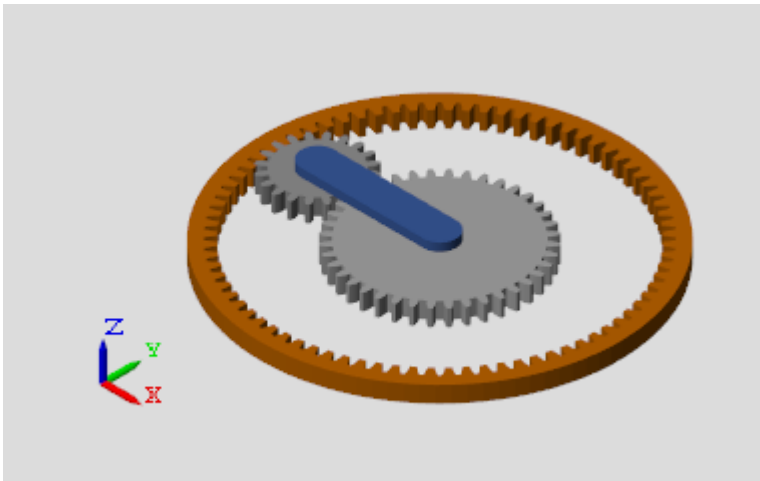
- 7 In the model workspace, define the Carrier block parameters using MATLAB code:

```
% Gear Carrier Parameters
Carrier.RGB = [0.25 0.4 0.7];
Carrier.L = Sun.R + Planet.R;
Carrier.W = 2*T;
Carrier.T = T/2;
```

```
Theta = (90:1:270)*pi/180;
Beta = (-90:1:90)*pi/180;
```

```
Carrier.CS = [-Carrier.L/2 + Carrier.W/2*cos(Theta) ...
Carrier.W/2*sin(Theta); Carrier.L/2 + Carrier.W/2*cos(Beta), ...
Carrier.W/2*sin(Beta)];
```

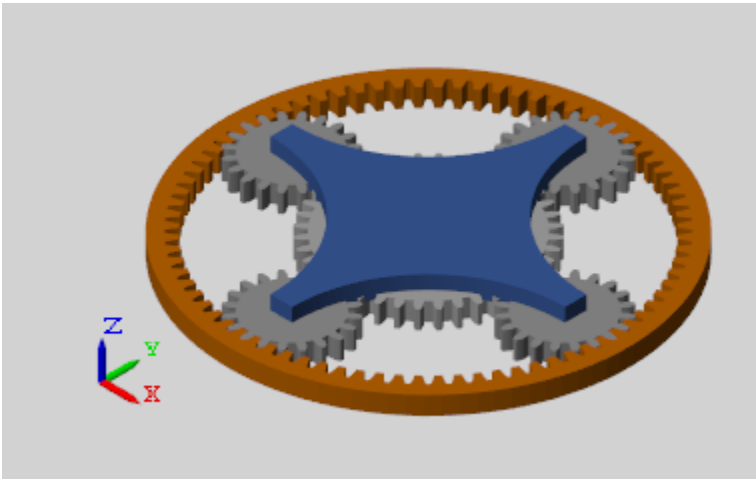
- 8 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the gear carrier now performs the task of the Distance Constraint block.



You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_d`.

Add More Planet Gears

Experiment with the model by adding more planet gears. Remember that you must change the Carrier body to accommodate any additional planet gears. To see an example with four planet gears, at the MATLAB command line enter `smdoc_planetary_gear_e`.



Constrain a Point to a Curve

In this section...

“Open the Flap Assembly Model” on page 2-58

“Effect of Constraints on the Model” on page 2-59

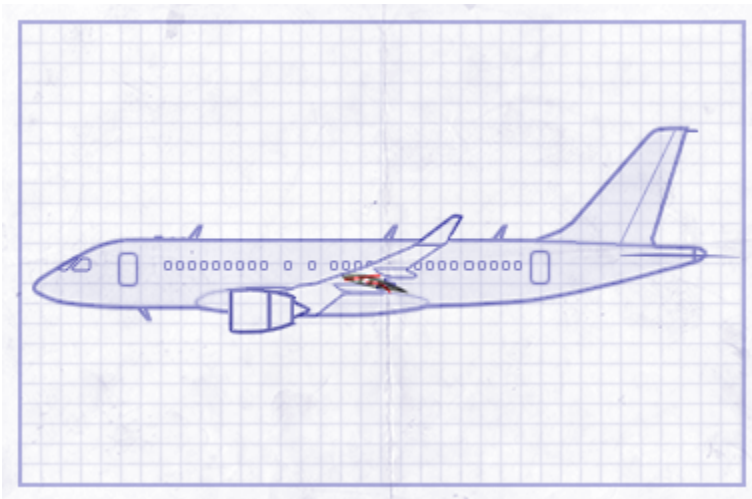
“Create the Connection Frames” on page 2-60

“Connect the Constraint Blocks” on page 2-61

“Specify the Flap Constraint Curves” on page 2-62

“Simulate the Model” on page 2-63

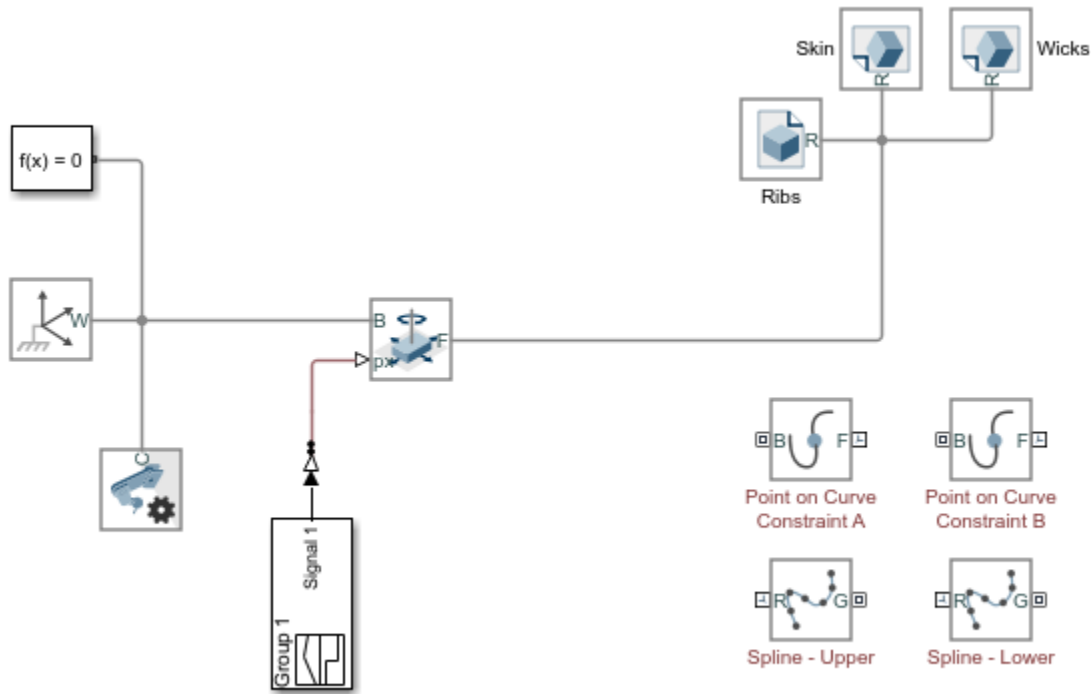
This example shows how to apply multiple point-on-curve constraints to a single body. The example is based on a partial model of an aircraft flap that extends and retracts by riding on curved tracks. You complete the model by adding point-on-curve constraints between two points on the flap and the flap track curves. The figure shows the visualization results of the flap model, once it is complete.



Open the Flap Assembly Model

At the MATLAB command prompt, enter `smdoc_poc_flap_start`. Simscape Multibody software opens the flap assembly model. Save the model with a different name in a convenient folder so that you do not inadvertently overwrite the model.

The model is missing key connection lines and block parameters and does yet not simulate. Three rigidly connected blocks, named Ribs, Skin, and Wicks, represent the flap body. A Planar Joint block connects this body to the world frame with three degrees of freedom—one rotational and two translational.

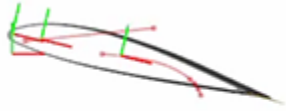


Effect of Constraints on the Model


The flap is at this point unconstrained. In this state, the degrees of freedom of the flap are those provided by the Planar Joint block. The flap is free to rotate about the z-axis, translate along the x-axis, and translate along the y-axis. The animated figure shows these motions. The frame axes are color-coded, with red denoting x, green y, and blue z. The z-axis points out of the screen is not visible in the figure.



The point-on-curve constraints couple the initially independent motions of the flap so that any one of these motions suffices to completely determine the flap trajectory. For example, in the fully constrained flap, a translation along the x-axis determines also the translation along the y-axis and the rotation about the z-axis. The animated figure shows the constrained motion that you obtain at the end of this example.

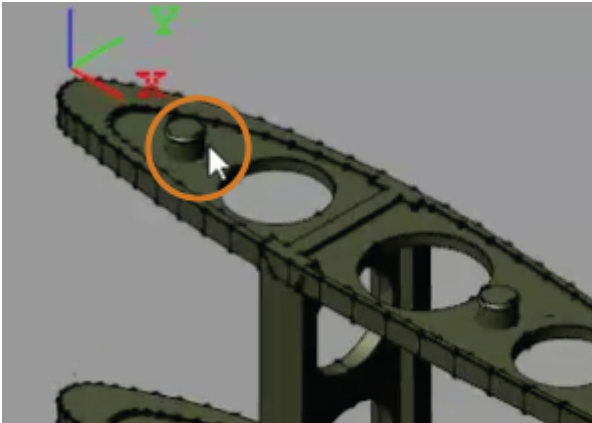


Create the Connection Frames

- 1 In the Ribs File Solid block dialog box, expand the **Frames** node and click the  button. The frame creation interface opens. You use this interface to create a new frame and set its position and orientation with respect to the solid geometry.
- 2 In the visualization pane, zoom in on the top portion of the flap and select the top surface of the rightmost cylindrical protrusion, as shown in the figure. The visualization pane highlights the selected surface and reveals its normal vector.

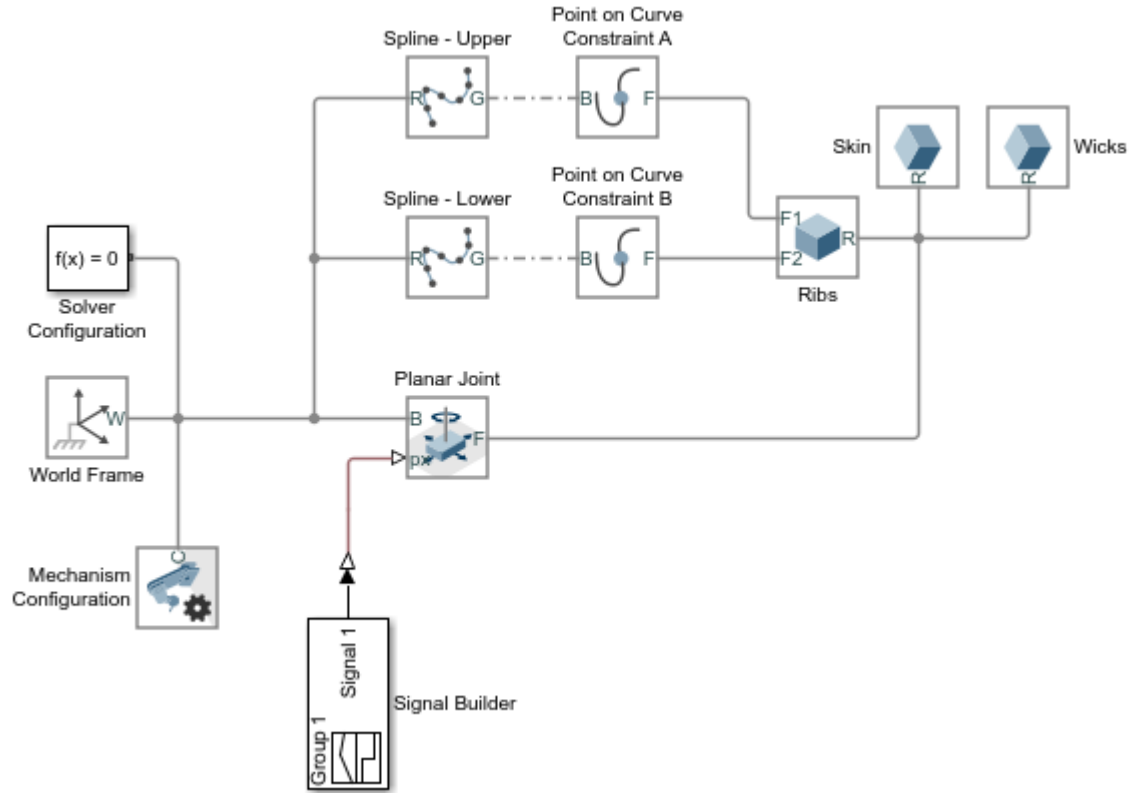


- 3 In the **Frame Origin** area of the frame creation interface, click the **Based on Geometric Feature** radio button and then the **Use Selected Feature** button. The frame moves to the center of the selected surface.
- 4 Click the **Save** button to add the new frame to the solid. The block exposes a new port, F1, corresponding to the new frame. You can rename the frame anything you want but, in this example, the default name suffices.
- 5 Repeat steps 1-4 to create a second frame with origin at the top surface of the leftmost cylindrical protrusion, as shown in the figure. The block exposes a new port, F2, corresponding to the new frame.



Connect the Constraint Blocks

- 1** Connect the frame port of each Spline block to the World Frame block. This connection ensures that the constraint curves specified in the Spline blocks are resolved in the World frame. The curve definitions are currently the block defaults. You later change these defaults to obtain a reasonable flap trajectory.
- 2** Connect the frame ports of the Point on Curve Constraint blocks to the frame ports that you added to the Ribs File Solid block (F1 and F2). The origins of these frames are the points that you constrain in this example.
- 3** Connect the geometry ports of the Spline blocks to the geometry ports of the Point on Curve Constraint blocks as shown in the figure. The geometry connection lines identify the spline curve definitions as the curves to constrain.



Specify the Flap Constraint Curves

- 1 Open Model Explorer and, in the **Model Hierarchy** pane, expand the **smdoc_poc_cam_start > Model Workspace** node. Model Explorer enables you to define workspace variables so that you can reference them in block dialog box parameters.
- 2 In the **Model Workspace** pane of Model Explorer, set the **Data source** parameter to MATLAB Code. An editable field appears with the still-incomplete variable definitions:

```
theta = linspace(4*pi/12,pi/8,4)';
lowerTrack = [];
upperTrack = [];
```

The variables `lowerTrack` and `upperTrack` are the constraint curves of the lower and upper flap tracks. The variable `theta` is an angle range used in the definition of the lower track curve.

- 3 Complete the `lowerTrack` and `upperTrack` definitions by specifying a few points on the curves as shown in the following code snippet. You must click the **Reinitialize from Source** button to apply the changes to the model. The new code portions are shown in blue.

```
theta = linspace(4*pi/12,pi/8,4)';
lowerTrack = [350 0; (640+100*cos(theta)) (130*sin(theta)-210)];
upperTrack = [50 50; 550 100];
```

- 4 In the dialog box of each Spline block, specify the parameters listed in the table. The **Interpolation Points** parameter is defined in terms of the MATLAB variables from the model workspace. Ensure that this parameter is in units of mm. The **Color** parameter is defined as a normalized RGB vector corresponding to a light shade of red.

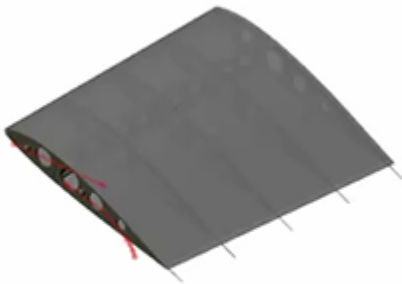
Block	Interpolation Points	Graphic > Visual Properties + Color
Spline - Upper	upperTrack	[0.8, 0, 0]
Spline - Lower	lowerTrack	[0.8, 0, 0]

- Update the block diagram. In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a static visualization of the model in its initial configuration. The figure shows the spline curves as they appear in Mechanics Explorer with the bodies hidden. You can hide a body by selecting its node in the tree view pane and selecting **Hide This**.



Simulate the Model

Simulate the model. Mechanics Explorer shows a dynamic visualization of the flap assembly. The motion of the flap is now constrained so that the origins of the frames you created always lie on the curves you defined. Switch between the standard views of Mechanics Explorer or rotate, pan, and zoom to better explore the flap and its motion.



To open a complete version of the flap assembly model, at the MATLAB command prompt, enter `smdoc_poc_flap`.

Internal Mechanics, Actuation and Sensing

- “Modeling and Sensing System Dynamics” on page 3-2
- “Modeling Gravity” on page 3-4
- “Model Gravity in a Planetary System” on page 3-8
- “Specifying Joint Actuation Inputs” on page 3-19
- “Joint Actuation Limitations” on page 3-26
- “Actuating and Sensing with Physical Signals” on page 3-28
- “Sensing” on page 3-31
- “Force and Torque Sensing” on page 3-33
- “Modeling Contact Force Between Two Solids” on page 3-36
- “Use Contact Proxies to Simulate Contact” on page 3-40
- “Model Wheel Contact in a Car” on page 3-49
- “Motion Sensing” on page 3-56
- “Rotational Measurements” on page 3-60
- “Translational Measurements” on page 3-65
- “Selecting a Measurement Frame” on page 3-69
- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specify Joint Actuation Torque” on page 3-84
- “Analyze Motion at Various Parameter Values” on page 3-94
- “Measure Forces and Torques Acting at Joints” on page 3-99
- “Sense Constraint Forces” on page 3-105
- “Specify Joint Motion Profile” on page 3-110
- “Specify Joint Motion in Planar Manipulator Model” on page 3-114

Modeling and Sensing System Dynamics

In this section...

“Provide Joint Actuation Inputs” on page 3-2

“Specify Joint Internal Mechanics” on page 3-2

“Model Body Interactions and External Loads” on page 3-2

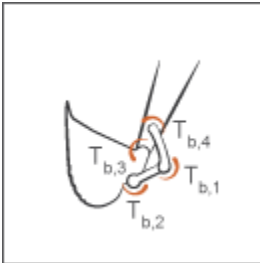
“Sense Dynamical Variables” on page 3-3

Provide Joint Actuation Inputs



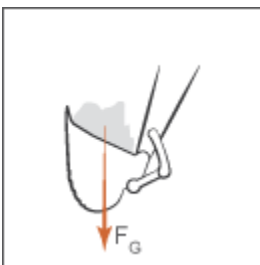
Identify the joints to actuate and the actuation type to use. Then, model the actuation inputs as time-varying physical signals and connect them to the various joints. See “Specify Joint Actuation Torque” on page 3-84 for an example.

Specify Joint Internal Mechanics



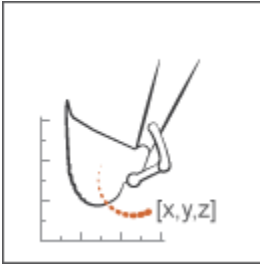
Model damping and spring behavior at joints. Specify joint damping coefficients to model energy dissipation and joint spring constants to model energy storage.

Model Body Interactions and External Loads



Identify the forces and torques acting at or between bodies not connected by joints. Model these forces and torques explicitly using Forces and Torques blocks. See “Model Gravity in a Planetary System” on page 3-8 for an example.

Sense Dynamical Variables



Identify the forces, torques, and motion variables to sense. You can sense these variables at joints through Joint blocks. You can also sense motion variables using the Transform Sensor block. See “Sense Motion Using a Transform Sensor Block” on page 3-79 for an example.

Modeling Gravity

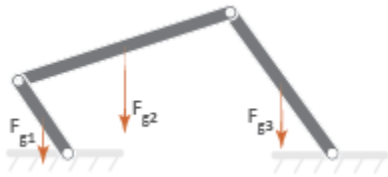
In this section...

- “Gravity Models” on page 3-4
- “Gravitational Force Magnitude” on page 3-5
- “Force Position and Direction” on page 3-5
- “Gravitational Torques” on page 3-6

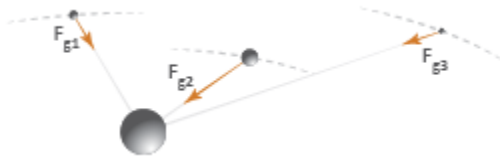
Gravity Models

Gravity influences motion in many natural and engineered systems. These range in scale from the very large, such as the planets orbiting the sun, to the relatively small, such as the shock absorbers damping gravity-driven oscillations in a car. In Simscape Multibody, you can add gravity to systems like these using three gravity models:

- Uniform gravity, as experienced by most earthbound systems. The force on each body due to uniform gravity depends only on its mass. This force is the same everywhere in space for a given body, though it can vary in time. You model uniform gravity using the Mechanism Configuration block.



- Gravitational field, as experienced by the planets in the solar system. The force on each body due to a gravitational field depends not only on its mass but also on its inverse square distance to the field origin. You model a gravitational field using the Gravitational Field block.



- Inverse-square law force pair, similar in nature to a gravitational field, but acting exclusively between one pair of bodies. You model an inverse-square law force pair using the Inverse Square Law Force block. You must specify the body masses and force constants explicitly.



Gravitational Force Magnitude

The force of gravity is an inverse-square law force—that is, one that decays with the square distance from the field origin to the target body. The magnitude of this force, F_g , follows from Newton’s law of universal gravitation which, for two bodies of mass M and m a distance R apart, states

$$F_g = -G \frac{Mm}{R^2}$$

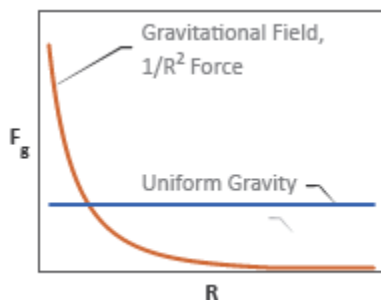
with G being the gravitational constant. This is the force that you model when you represent gravity through Gravitational Field or Inverse Square Law Force blocks. If the distance between source and target masses is constant, the gravitational force reduces to a simpler form,

$$F_g = -mg$$

with g being the nominal gravitational acceleration. Near the surface of the Earth, at a distance equal to Earth’s radius from the gravitational field origin, the nominal acceleration equals

$$g = \frac{GM}{R^2} \approx 9.80665 \text{ m/s}^2$$

This is the gravitational force that you model when you represent gravity through the Mechanism Configuration block. The figure shows how the magnitude of the gravitational force (F_g) varies with distance (R) for a given body under uniform gravity, a gravitational field, and an inverse-square law force pair.



Force Position and Direction

In a physical system, the force due to a gravitational field acts at a body’s center of mass—automatically computed during simulation—along the imaginary line connecting the field origin to the center of mass. These are also the application point and direction of gravity that the Gravitational Field block provides. See “Modeling Bodies” on page 1-4 for more information on how Simscape Multibody defines a body subsystem.

Far from the field origin, the field origin-center of mass line remains approximately constant at small-to-moderate displacements, and the force of gravity behaves as if its direction were fixed. This is the approximation used in the Mechanism Configuration block. Gravity still acts at each body’s center of mass, but its direction is now fixed along the gravity vector that you specify.

If you want to model the effects of gravity on a point other than a body’s center of mass, you can add a frame at the desired location and apply a gravitational force directly at that frame. You model the

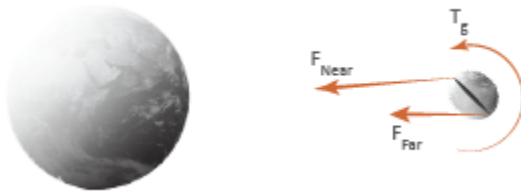
force using the Inverse Square Law Force block. This force points along the imaginary line between the two body frames that the Inverse Square Law Force block connects.

The table summarizes the application point and direction of gravity provided by the different blocks.

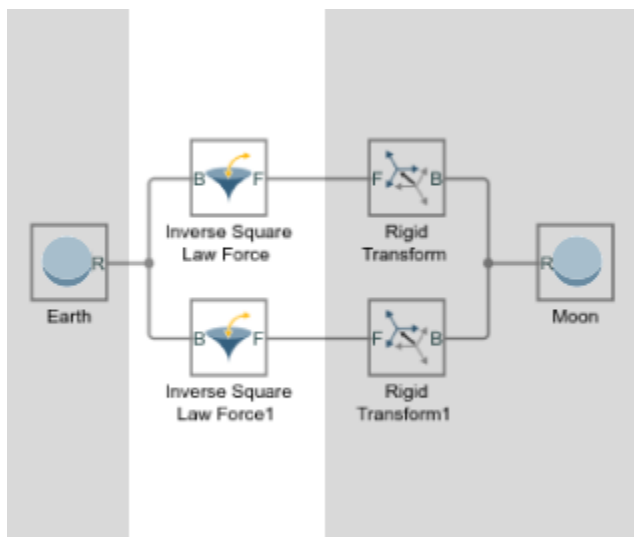
Block	Position	Direction
Mechanism Configuration	Center of Mass	Specified gravity vector
Gravitational Field	Center of Mass	Field origin-center of mass line
Inverse Square Law Force	Connection frames	Base-follower frame line

Gravitational Torques

A gravitational torque can arise in a large body immersed in a nonuniform gravitational field. The lemon-shaped moon, with its near end perpetually facing Earth, is one example. Being placed at different distances from Earth, the near and far elongated ends experience dissimilar gravitational forces, resulting in a net gravitational torque if the line between the two ends ever falls out of alignment with the center of the Earth.

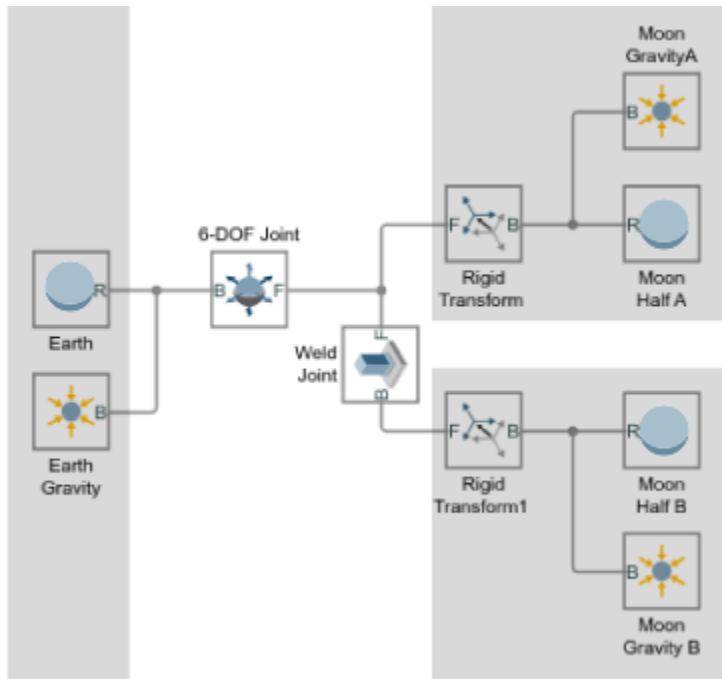


You can model such torques in Simscape Multibody by modeling the different gravitational forces acting on a body. You do this using the Inverse Square Law Force or Gravitational Field block. If you use the Inverse Square Law Force block, you must create additional frames in each body whose response to gravitational torque you want to model. You must then apply a gravitational force to each frame explicitly. The figure shows an example.



Torque on the moon due to dissimilar gravitational forces at the elongated ends

If you use the Gravitational Field block, you must split each body into discrete sections and connect them through Weld Joint blocks. The Gravitational Field block automatically applies a force at the center of mass of each section, approximating the compound effect of the different gravitational forces on the body—which in this case is treated as a rigid multibody system. The figure shows an example.



Torque on the moon due to dissimilar gravitational forces at the elongated ends

Model Gravity in a Planetary System

In this section...

“Model Overview” on page 3-8

“Step 1: Start a New Model” on page 3-8

“Step 2: Add the Solar System Bodies” on page 3-9

“Step 3: Add the Degrees of Freedom” on page 3-12

“Step 4: Add the Initial State Targets” on page 3-13

“Step 5: Add the Gravitational Fields” on page 3-16

“Step 6: Configure and Run the Simulation” on page 3-17

“Open an Example Model” on page 3-18

Model Overview

This tutorial shows how to simulate the gravity-driven orbits of the major solar system bodies. The model treats the sun and planets as perfect spheres each with three translational degrees of freedom. Planet spin is ignored. Gravitational fields generate the forces that keep the planets in orbit.



Spherical Solid blocks represent the solar system bodies and provide their geometries, inertias, and colors. Cartesian Joint blocks define the bodies’ degrees of freedom relative to the world frame, located at the solar system barycenter. Gravitational Field blocks add the long-range forces responsible for bending the initial planet trajectories into closed elliptical orbits.

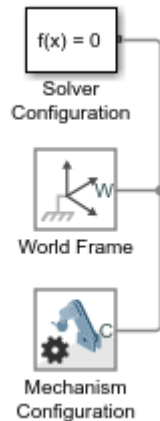
The Cartesian Joint blocks provide the initial states—positions and velocities—of the sun and planets relative to the world frame. The initial states correspond to the solar system configuration on Jun 20th, 2016. They are sourced from ephemeris databases maintained by the Jet Propulsion Laboratory (JPL).

You can query the databases through the JPL Horizons system using a web or telnet interface. Aerospace Toolbox users can alternatively obtain the ephemeris data at the MATLAB command prompt using the `planetEphemeris` function after installing the Aerospace Ephemeris Data support package.

Step 1: Start a New Model

Open the Simscape Multibody model template and remove all unnecessary blocks. Modify the gravity settings so that you can add gravitational fields to the model. The result provides a starting point for the solar system model.

- 1 At the MATLAB command prompt, enter `smnew`. MATLAB opens a model template with commonly used blocks and suitable solver settings for Simscape Multibody models.
- 2 Cut all but the Mechanism Configuration, Solver Configuration, and World Frame blocks. These three blocks provide the model with gravity settings, solver settings, and a global inertia reference frame.



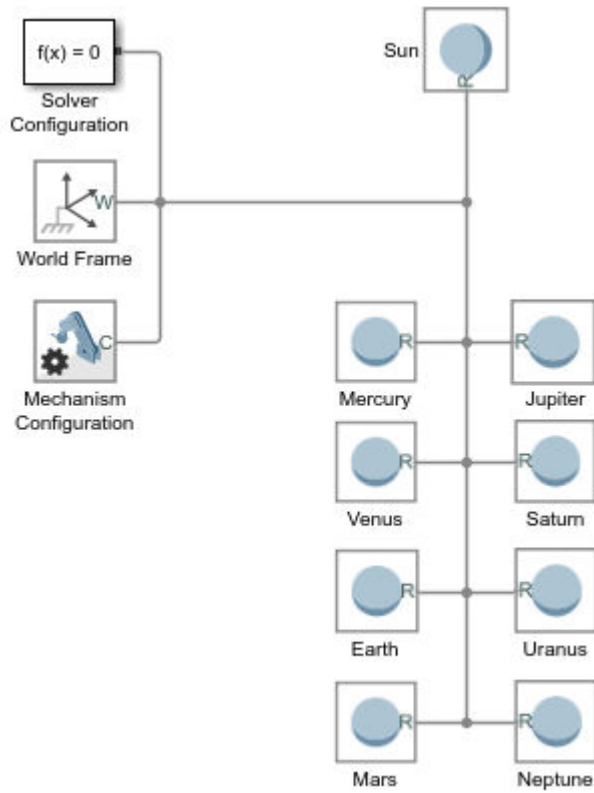
- 3 In the Mechanism Configuration block dialog box, set **Uniform Gravity** to **None**. This setting enables you to model gravity as an inverse-square law force using Gravitational Field blocks instead.

Step 2: Add the Solar System Bodies

Represent the solar system bodies using Spherical Solid blocks. Specify the geometry and inertia parameters in terms of MATLAB variables and initialize these variables in the model workspace using Model Explorer. The variables are data structures named after the solar system bodies using proper-noun capitalization.

Connect and Configure the Solid Blocks

- 1 Add to the model nine Spherical Solid blocks from the Body Elements library. The blocks represent the sun and eight known planets.
- 2 Connect and name the blocks as shown in the figure. The branched frame connection line between the world frame and the planets makes them rigidly connected and coincident in space. You later change this condition using Cartesian Joint blocks.



- 3 In the Spherical Solid block dialog boxes, set the **Inertia > Based on** parameter to **Mass**. The inertia parameter setting enables you to specify the solid mass directly so that you can scale the planet shapes without affecting the model dynamics.
- 4 Specify the following Spherical Solid block parameters in terms of MATLAB data structure fields. Enter the field names in the format *Structure.Field*, where *Structure* is the title-case name of the solar system body and *Field* is the string shown in the table—e.g., Sun.R or Earth.RGB.

Block Parameter	Field String
Geometry > Radius	R
Inertia > Mass	M
Graphic > Visual Properties > Color	RGB

You later define the new structure fields in the model workspace using Model Explorer.

Add the Solid Property Initialization Code

- 1 In the **Modeling** tab, click **Model Explorer**.
- 2 In the Model Hierarchy pane of Model Explorer, expand the node for your model and select **Model Workspace**. The Model Hierarchy pane is on the left side.
- 3 In the Model Workspace pane of Model Explorer, set **Data Source** to **MATLAB Code**. The Model Workspace pane is on the right side.
- 4 In the **MATLAB Code** field, add the initialization code for the sun and planet solid properties. The code is organized into sections named after the solar system bodies. You later add the initial position and velocity data to these sections.

```
% All values are in SI units.
% RGB color vectors are on a normalized 0-1 scale.
% Body dimensions are scaled for visualization purposes.
% Scaling has no impact on model dynamics.

% Scaling
SunScaling = 0.5e2;
TerrestrialPlanetScaling = 1.2e3;
GasGiantScaling = 2.5e2;

% Sun
Sun.M = 1.99e30;
Sun.R = 6.96e8*SunScaling;
Sun.RGB = [1 0.5 0];

% Mercury
Mercury.M = 3.30e23;
Mercury.R = 2.44e6*TerrestrialPlanetScaling;
Mercury.RGB = [0.5 0.5 0.5];

% Venus
Venus.M = 4.87e24;
Venus.R = 6.05e6*TerrestrialPlanetScaling;
Venus.RGB = [1 0.9 0];

% Earth
Earth.M = 5.97e24;
Earth.R = 6.05e6*TerrestrialPlanetScaling;
Earth.RGB = [0.3 0.6 0.8];

% Mars
Mars.M = 6.42e23;
Mars.R = 3.39e6*TerrestrialPlanetScaling;
Mars.RGB = [0.6 0.2 0.4];

% Jupiter
Jupiter.M = 1.90e27;
Jupiter.R = 6.99e7*GasGiantScaling;
Jupiter.RGB = [0.6 0 0.3];

% Saturn
Saturn.M = 5.68e26;
Saturn.R = 5.82e7*GasGiantScaling;
Saturn.RGB = [1 1 0];

% Uranus
Uranus.M = 8.68e25;
Uranus.R = 2.54e7*GasGiantScaling;
Uranus.RGB = [0.3 0.8 0.8];

% Neptune
Neptune.M = 1.02e26;
Neptune.R = 2.46e7*GasGiantScaling;
Neptune.RGB = [0.1 0.7 0.8];
```

- 5 Click **Reinitialize from Source**.

The Spherical Solid blocks now have all the numerical data they need to render the planet shapes and colors. Try opening a Spherical Solid block dialog box and verify that a sphere now appears in the solid visualization pane.

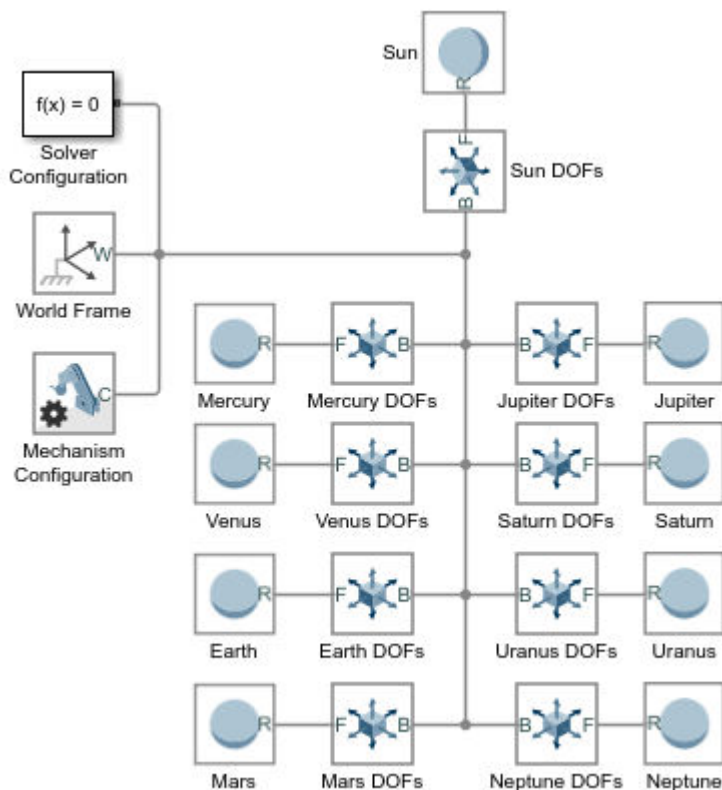


Earth Solid Visualization

Step 3: Add the Degrees of Freedom

Add three translational degrees of freedom between the solar system barycenter and each solar system body using Cartesian Joint blocks. You later use these blocks to specify the initial positions and velocities of the solar system bodies.

- 1 Add to the model nine Cartesian Joint blocks from the Joints library. The blocks provide the translational degrees of freedom of the sun and eight known planets.
- 2 Connect and name the blocks as shown in the figure. If you place a block on an existing connection line, Simscape Multibody software automatically connects the block to that line. Flip and rotate the joint blocks to ensure that Spherical Solid blocks connect only to follower (F) frame ports.



The sun and planets are no longer rigidly connected. They can now translate relative to each other. They are, however, still coincident in space. To place them at different initial positions and give them initial velocities, you must specify the joint state targets.

Step 4: Add the Initial State Targets

Specify the sun and planet initial states in terms of MATLAB variables using the Cartesian Joint blocks in your model. Then, initialize the new MATLAB variables in the model workspace using Model Explorer. You define the MATLAB variables as new fields in the existing data structures.

Configure the Cartesian Joint Blocks

- 1 In the Cartesian Joint block dialog boxes, check the **State Targets > Specify Position Target** and **State Targets > Specify Velocity Target** checkboxes for the X, Y, and Z prismatic joint primitives. These settings enable you to specify the desired initial states of the sun and planets.
- 2 Specify the Cartesian Joint state target values for the X, Y, and Z prismatic joint primitives in terms of MATLAB structure fields. Enter the field names in the format *Structure.Field*, where *Structure* is the title-case name of the solar system body and *Field* is the string shown in the table—e.g., Sun.Px or Earth.Vz.

Joint Primitive Axis	State Target	Field String
X	Position	Px
	Velocity	Vx
Y	Position	Py
	Velocity	Vy
Z	Position	Pz
	Velocity	Vz

You later define the new structure fields in the model workspace using Model Explorer.

Add the State Target Initialization Code

- 1 In the **Modeling** tab, click **Model Explorer**.
- 2 In the Model Hierarchy pane of Mechanics Explorer, expand the node for your model and select **Model Workspace**. The Model Hierarchy pane is on the left side.
- 3 In the Model Workspace pane of Model Explorer, set **Data Source** to **MATLAB Code**. The Model Workspace pane is on the right side.
- 4 In the **MATLAB Code** field, add the initialization code for the joint state targets. The new code, shown in blue, consists of the position and velocity components obtained from the JPL ephemeris databases. You can copy just the new code or replace your entire model workspace code with that shown.

```
% All values are in SI units.
% RGB color vectors are on a normalized 0-1 scale.
% Body dimensions are scaled for visualization purposes.
% Scaling has no impact on model dynamics.
```

```
% Scaling
SunScaling = 0.5e2;
TerrestrialPlanetScaling = 1.2e3;
```

```
GasGiantScaling = 2.5e2;

% Sun
Sun.M = 1.99e30;
Sun.R = 6.96e8*SunScaling;
Sun.RGB = [1 0.5 0];
Sun.Px = 5.5850e+08;
Sun.Py = 5.5850e+08;
Sun.Pz = 5.5850e+08;
Sun.Vx = -1.4663;
Sun.Vy = 11.1238;
Sun.Vz = 4.8370;

% Mercury
Mercury.M =3.30e23;
Mercury.R = 2.44e6*TerrestrialPlanetScaling;
Mercury.RGB = [0.5 0.5 0.5];
Mercury.Px = 5.1979e+10;
Mercury.Py = 7.6928e+09;
Mercury.Pz = -1.2845e+09;
Mercury.Vx = -1.5205e+04;
Mercury.Vy = 4.4189e+04;
Mercury.Vz = 2.5180e+04;

% Venus
Venus.M = 4.87e24;
Venus.R = 6.05e6*TerrestrialPlanetScaling;
Venus.RGB = [1 0.9 0];
Venus.Px = -1.5041e+10;
Venus.Py = 9.7080e+10;
Venus.Pz = 4.4635e+10;
Venus.Vx = -3.4770e+04;
Venus.Vy = -5.5933e+03;
Venus.Vz = -316.8994;

% Earth
Earth.M = 5.97e24;
Earth.R = 6.05e6*TerrestrialPlanetScaling;
Earth.RGB = [0.3 0.6 0.8];
Earth.Px = -1.1506e+09;
Earth.Py = -1.3910e+11;
Earth.Pz = -6.0330e+10;
Earth.Vx = 2.9288e+04;
Earth.Vy = -398.5759;
Earth.Vz = -172.5873;

% Mars
Mars.M = 6.42e23;
Mars.R = 3.39e6*TerrestrialPlanetScaling;
Mars.RGB = [0.6 0.2 0.4];
Mars.Px = -4.8883e+10;
Mars.Py = -1.9686e+11;
Mars.Pz = -8.8994e+10;
Mars.Vx = 2.4533e+04;
Mars.Vy = -2.7622e+03;
Mars.Vz = -1.9295e+03;

% Jupiter
```

```
Jupiter.M = 1.90e27;
Jupiter.R = 6.99e7*GasGiantScaling;
Jupiter.RGB = [0.6 0 0.3];
Jupiter.Px = -8.1142e+11;
Jupiter.Py = 4.5462e+10;
Jupiter.Pz = 3.9229e+10;
Jupiter.Vx = -1.0724e+03;
Jupiter.Vy = -1.1422e+04;
Jupiter.Vz = -4.8696e+03;

% Saturn
Saturn.M = 5.68e26;
Saturn.R = 5.82e7*GasGiantScaling;
Saturn.RGB = [1 1 0];
Saturn.Px = -4.2780e+11;
Saturn.Py = -1.3353e+12;
Saturn.Pz = -5.3311e+11;
Saturn.Vx = 8.7288e+03;
Saturn.Vy = -2.4369e+03;
Saturn.Vz = -1.3824e+03;

% Uranus
Uranus.M = 8.68e25;
Uranus.R = 2.54e7*GasGiantScaling;
Uranus.RGB = [0.3 0.8 0.8];
Uranus.Px = 2.7878e+12;
Uranus.Py = 9.9509e+11;
Uranus.Pz = 3.9639e+08;
Uranus.Vx = -2.4913e+03;
Uranus.Vy = 5.5197e+03;
Uranus.Vz = 2.4527e+03;

% Neptune
Neptune.M = 1.02e26;
Neptune.R = 2.46e7*GasGiantScaling;
Neptune.RGB = [0.1 0.7 0.8];
Neptune.Px = 4.2097e+12;
Neptune.Py = -1.3834e+12;
Neptune.Pz = -6.7105e+11;
Neptune.Vx = 1.8271e+03;
Neptune.Vy = 4.7731e+03;
Neptune.Vz = 1.9082e+03;
```

5 Click **Reinitialize from Source**.

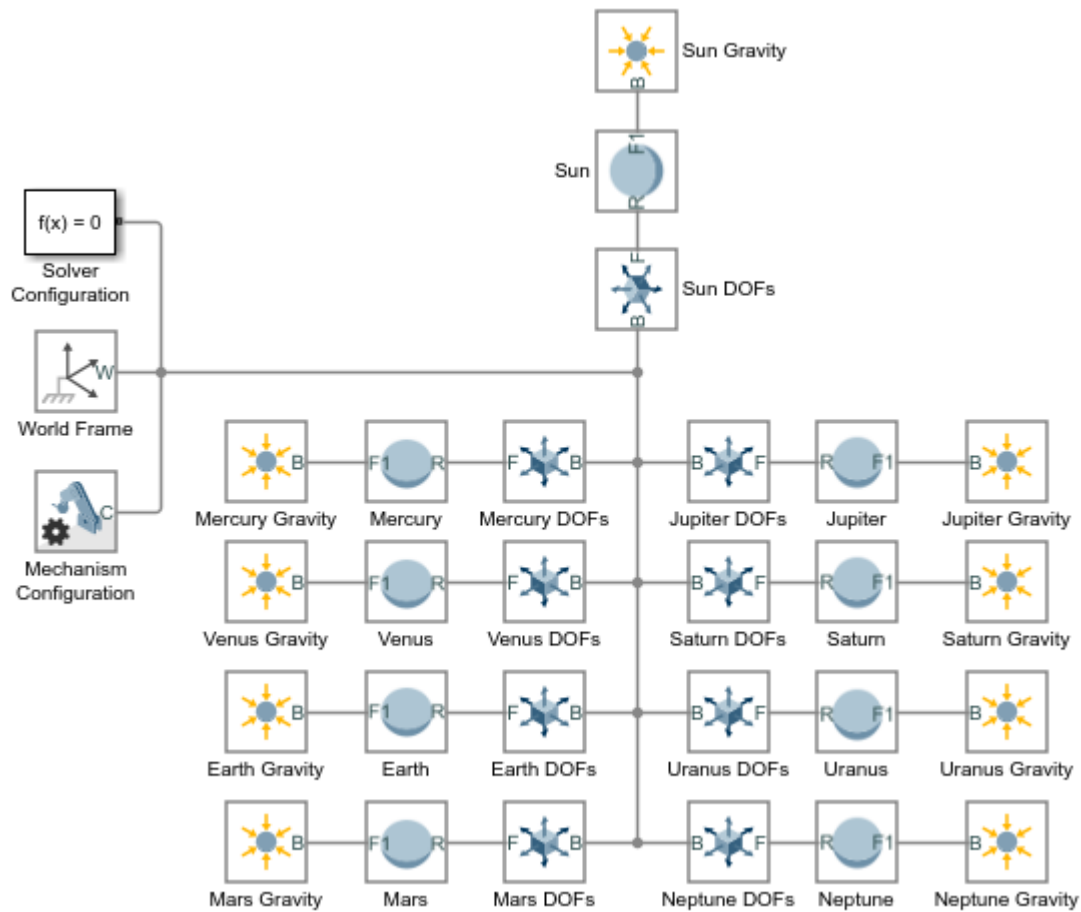
The model now has the numerical data it needs to assemble the planets in the position coordinates obtained from the JPL databases. However, a model simulation at this point would show the planets moving in straight-line trajectories. To obtain elliptical orbits, you must complete the model by adding the sun and planet gravitational fields.



Step 5: Add the Gravitational Fields

Model the gravitational pull of each solar system body using the Gravitational Field block. This block automatically computes the gravitational pull of a body on all other bodies using Newton's law of universal gravitation.

- 1 In each Spherical Solid block dialog box, expand the **Frames** area and click the **Create** button.
- 2 Set the **Frame Name** parameter to R2 and click the **Save** button. The new frame is an exact copy of the reference frame but has a separate frame port. You can use these ports to connect the gravitational field blocks while avoiding crossed connection lines.
- 3 Add to the model nine Gravitational Field blocks from the Forces and Torques library. The blocks provide the gravitational forces that each solar system body exerts on all other bodies.
- 4 Connect and name the blocks as shown in the figure. Ensure that the blocks connect directly to the Spherical Solid blocks. Such a connection ensures that the fields are centered on the solid spheres and rigidly connected to them.



- 5 In the Gravitational Field blocks, specify the **Mass** parameter as MATLAB structure field names. Enter the field names in the format *Structure.Field*, where *Structure* is the title-case name of the solar system body and *Field* is the string M—e.g., Sun.M or Earth.M. These fields have been previously defined in the model workspace.

Step 6: Configure and Run the Simulation

Configure the Simulink solver settings to capture ten earth revolutions in a single simulation. Then, simulate the model and view the resulting solar system animation. Configure the animation settings to play the ten-year animation in the period of a few seconds.

Configure the Solver Settings

- 1 Open the Configuration Parameters. In the **Modeling** tab, click **Model Settings**.
- 2 Set the **Stop time** parameter to $10 \times 365 \times 24 \times 60 \times 60$. This number, equal to ten years in seconds, allows you to simulate a full ten earth revolutions from Jun 20th, 2016 through Jun 20th, 2026.
- 3 Set the **Max step size** parameter to $24 \times 60 \times 60$. This number, equal to one day in seconds, is small enough to provide smooth animation results. Increase this number if you prefer faster simulation results.

Update and Simulate the Model

Update the block diagram. In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a static 3-D display of the model in its initial state. Check that the sun and planets appear in the visualization pane and that their relative dimensions and positions are reasonable.

Run the simulation.. Mechanics Explorer plays an animation of the solar system. Note that at the default base playback speed, the planets appear static. You must increase this speed in the Mechanics Explorer animation settings.

Configure the Animation Settings

- 1 In Mechanics Explorer, select **Tools > Animation Settings**.
- 2 In **Base(1X) Playback Speed**, enter 3153600. This speed corresponds to one earth revolution every ten seconds.
- 3 Pause and play the animation to apply the new base playback speed. The figure shows the animation results at the new speed.



Open an Example Model

You can open a complete solar system model by entering `smdoc_solar_system_wfield_b` at the MATLAB command prompt.

Specifying Joint Actuation Inputs

In this section...

“Actuation Modes” on page 3-19

“Motion Input” on page 3-21

“Input Handling” on page 3-22

“Assembly and Simulation” on page 3-23

“Specifying Motion Input Derivatives” on page 3-24

Actuation Modes

Joint blocks provide two actuation parameters. These parameters, **Force/Torque** and **Motion**, govern how the joint behaves during simulation. Depending on the parameter settings you select, a joint block can accept either actuation parameter as input or automatically compute its value during simulation.

An additional setting (**None**) allows you to set actuation force/torque directly to zero. The joint primitive is free to move during simulation, but it has no actuator input. Motion is due indirectly to forces and torques acting elsewhere in the model, or directly to velocity state targets.

Z Revolute Primitive (Rz)		Z Revolute Primitive (Rz)	
> State Targets		> State Targets	
> Internal Mechanics		> Internal Mechanics	
> Limits		> Limits	
▼ Actuation		▼ Actuation	
Torque	Automatically Computed	Torque	Automatically Computed
Motion	None Provided by Input	Motion	Automatically Computed
> Sensing	Automatically Computed	> Sensing	Provided by Input Automatically Computed

Like all joint block parameters, you select the actuation parameter settings for each joint primitive separately. Different joint primitives in the same block need not share the same actuation settings. Using a Pin Slot Joint block, for example, you can provide motion input and have actuation torque automatically computed for the **Z Revolute Primitive (Rz)**, while having motion automatically computed with no actuation force for the **X Prismatic Primitive (Px)**.

▼ X Prismatic Primitive (Px)	
> State Targets	
> Internal Mechanics	
> Limits	
▼ Actuation	
Force	None ▼
Motion	Automatically Computed ▼
> Sensing	
▼ Z Revolute Primitive (Rz)	
> State Targets	
> Internal Mechanics	
> Limits	
▼ Actuation	
Torque	Automatically Computed ▼
Motion	Provided by Input ▼
> Sensing	

By combining different **Force/Torque** and **Motion** actuation settings, you can achieve different joint actuation modes. Forward dynamics and inverse dynamics modes are two common examples. You actuate a joint primitive in forward dynamics mode by providing actuation force/torque as input while having motion automatically computed. Conversely, you actuate a joint primitive in inverse dynamics mode by providing motion as input while having actuation force/torque automatically computed.

Other joint actuation modes, including fully computed and fully specified modes, are possible. The table summarizes the different actuation modes that you can obtain by manipulating the actuation parameter settings.

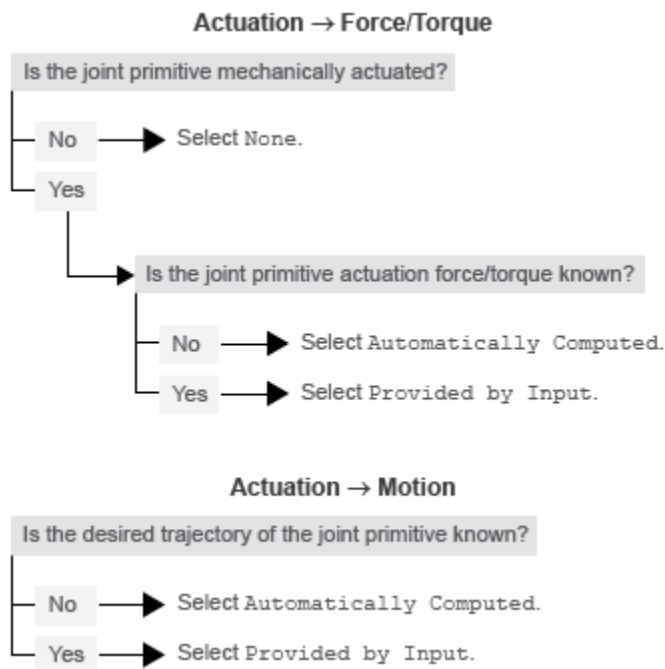
		Actuator Motion	
		Provided by Input	Automatically Computed
Actuator Force/Torque	None	Unactuated Motion	Passive
	Provided by Input	Fully Specified	Forward Dynamics
	Automatically Computed	Inverse Dynamics	Fully Computed

Joint Actuation Modes

More generally, thinking of joint actuation in terms of the specified or calculated quantities—i.e., force/torque and motion—provides a more practical modeling approach. You may not always know the appropriate mode for a joint but, having planned the model beforehand, you should always know the answers to two questions:

- Is the joint primitive mechanically actuated?
- Is the desired trajectory of the joint primitive known?

By selecting the joint actuation settings based on the answers to these questions, you can ensure that each joint is properly set for your application. The figure shows the proper settings depending on your answers.



Selecting Joint Primitive Actuation Settings

Motion Input

The motion input of a joint primitive is a timeseries object specifying that primitive's trajectory. For a prismatic primitive, that trajectory is the position coordinate along the primitive axis, given as a function of time. The coordinate provides the position of the follower frame origin with respect to the base frame origin. The primitive axis is resolved in the base frame.

For a revolute primitive, the trajectory is the angle about the primitive axis, given as a function of time. This angle provides the rotation of the follower frame with respect to the base frame about the primitive axis. The axis is resolved in the base frame.

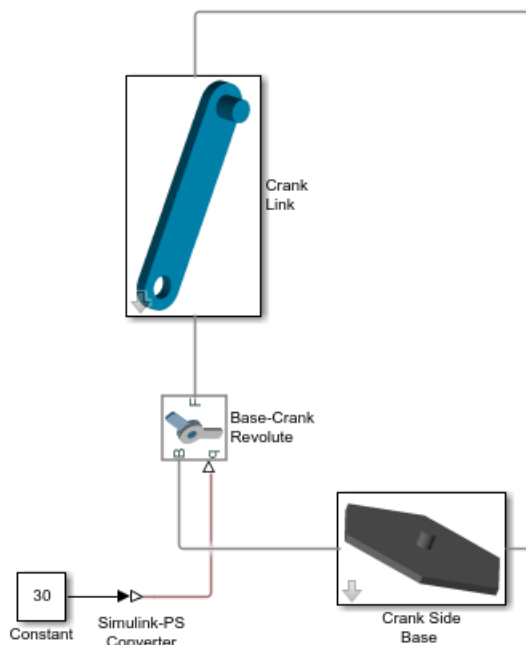
Spherical joint primitives provide no motion actuation options. You can specify actuation torque for these primitives, but you cannot prescribe their trajectories. Those trajectories are always automatically computed from the model dynamics during simulation.

Zero Motion Prescription

Unlike **Actuation > Force/Torque**, the **Actuation > Motion** parameter provides no zero input option, corresponding to a fixed joint primitive during simulation. You can, however, prescribe zero motion the same way you prescribe all other types of motion: using Simscape and Simulink blocks.

In Simscape Multibody, motion input signals are position-centric. You specify the joint primitive position and, if filtered to the second-order, the Simulink-PS Converter block smooths the signal while providing its two time-derivatives automatically. This behavior makes zero motion prescription straightforward: just provide a constant signal to the motion actuation input port of the joint primitive and simulate.

The figure shows an example of zero-motion prescription. A Simulink Constant block provides a constant position value. A Simulink-PS Converter block converts this Simulink signal into a Simscape signal compatible with the motion actuation input port of the Base-Crank Revolute Joint block. Assuming that assembly and simulation are successful, this joint will maintain a fixed angle of 30 degrees, corresponding to the value set in the Simulink Constant block and the units set in the Simulink-PS Converter block.



Input Handling

When prescribing a joint primitive trajectory, it is practical to specify a single input, the position, and filter that input using a Simulink-PS Converter block. This filter, which must be second-order, automatically provides the two time derivatives of the motion input. Because it also smooths the input signal, the filter can help prevent simulation issues due to sudden changes or discontinuities, such as those present when using a Simulink Step block.

Filtering smooths the input signal over a time scale of the order of the input filtering time constant. The larger the time constant, the greater the signal smoothing, and the more distorted the signal tends to become. The smaller the time constant, the closer the filtered signal is to the input signal, but also the greater the model stiffness—and, hence, the slower the simulation.

As a guideline, the input filtering time constant should be only as small as the smallest relevant time scale in a model. By default, its value is 0.001 s. While appropriate for many models, this value is often too small for Simscape Multibody models. For faster simulation, start with a value of 0.01 s. Decrease this value for greater accuracy.

If you know the two time derivatives of the motion input signal, you can specify them directly. This approach is most convenient for simple trajectories with simple derivatives. You must, however, ensure that the two derivative signals are compatible with the position signal. If they are not, even when simulation proceeds, results may be inaccurate.

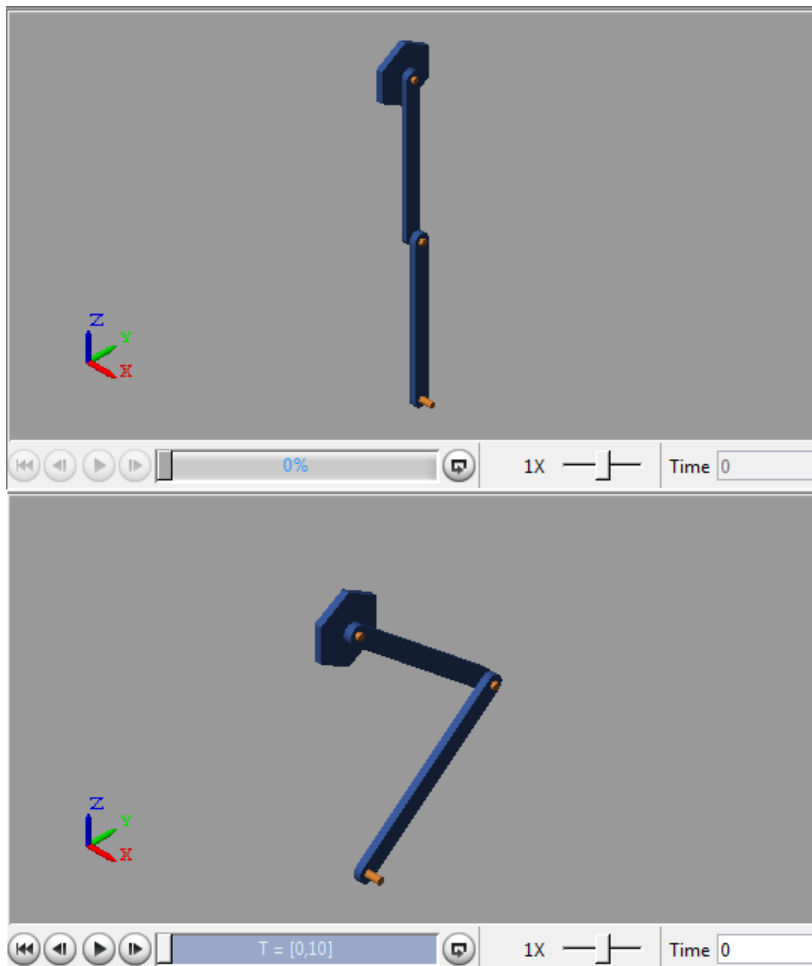
Assembly and Simulation

Simscape Multibody joints with motion inputs start simulation (**Ctrl+T**) at the initial position dictated by the input signal. This initial position may differ from the assembled state, which is governed by an assembly algorithm optimized to meet the joint state targets, if any. Even in the absence of joint state targets, the assembled state may differ from that at simulation time zero.

Note You obtain the assembled state each time you update the block diagram, e.g., by pressing **Ctrl+D**. You obtain the initial simulation state each time you run the simulation, e.g., by pressing **Ctrl+T**, and pausing at time zero.

Due to the discrepancy between the two states, Model Report provides accurate initial state data only for models lacking motion inputs. For models possessing motion inputs, that data is accurate only when the initial position prescribed by the motion input signal exactly matches the initial position prescribed in the joint state targets.

Similarly, Mechanics Explorer displays the initial joint states accurately only for models lacking motion inputs. As it transitions from the assembled state to the initial simulation state, Mechanics Explorer may show a sudden jump if a model contains motion inputs that are incompatible with the joint state targets. You can eliminate the sudden change by making the initial position prescribed by joint motion inputs equal to the initial position prescribed by the joint state targets.



Specifying Motion Input Derivatives

If filtering the input signal using the Simulink-PS Converter block, you need only to provide the position signal. The block automatically computes the derivatives. You must, however, select second-order filtering in the block dialog box:

- 1 Open the dialog box of the Simulink-PS Converter block and click **Input Handling**.
- 2 In **Filtering and derivatives**, select Filter input.
- 3 In **Input filtering order**, select Second-order filtering.
- 4 In **Input filtering time constant (in seconds)**, enter the characteristic time over which filter smooths the signal. A good starting value is 0.01 seconds.

If providing the input derivatives directly, you must first compute those derivatives. Then, using the Simulink-PS Converter block, you can provide them to the target joint block. To specify the input derivatives directly:

- 1 Open the Simulink-PS Converter block receiving the input signal and click the **Input Handling** tab.
- 2 In **Filtering and derivatives**, select Provide input derivative(s).

- 3** To specify both derivatives, in **Input derivatives**, select Provide first and second derivatives.

The block displays two additional physical signal ports, one for each derivative.

See Also

Related Examples

- “Specify Joint Motion in Planar Manipulator Model” on page 3-114
- “Specify Joint Motion Profile” on page 3-110
- “Specifying Motion Input Derivatives” on page 3-24

Joint Actuation Limitations

In this section...
“Closed Loop Restriction” on page 3-26
“Motion Actuation Not Available in Spherical Primitives” on page 3-26
“Redundant Actuation Mode Not Supported” on page 3-26
“Model Report and Mechanics Explorer Restrictions” on page 3-26
“Motion-Controlled DOF Restriction” on page 3-27

Closed Loop Restriction

Each closed kinematic loop must contain at least one joint block without motion inputs or computed actuation force/torque. This condition applies even if one of the joints acts as a virtual joint, e.g., the bushing joint in the “Specify Joint Motion in Planar Manipulator Model” on page 3-114 example. The joint without motion inputs or automatically computed actuation forces/torques can still accept actuation forces/torques from input.

In models not meeting this condition, you can replace a rigid connection line between two Solid blocks with a Weld Joint block. Since the Weld Joint block represents a rigid connection, this approach leaves the model dynamics unchanged. The advantage of this approach lies in its ability to satisfy the Simscape Multibody closed-loop requirement without altering model dynamics.

Motion Actuation Not Available in Spherical Primitives

Spherical joint primitives provide no motion actuation parameters. You can prescribe the actuation torque acting on the spherical primitive, but not its desired trajectory. For models requiring motion prescription for three concurrent rotational degrees of freedom, use joint blocks with three revolute primitives instead. These blocks include Gimbal Joint, Bearing Joint, and Bushing Joint.

Redundant Actuation Mode Not Supported

Redundant actuation, in which the end effector trajectory of a high-degree-of-freedom linkage is prescribed, is not allowed. Such linkages possess more degrees of freedom than are necessary to uniquely position the end effector and, as such, have no single solution. Models that have more degrees of freedom with automatically computed actuation forces/torques than with prescribed motion inputs cause simulation errors.

Model Report and Mechanics Explorer Restrictions

In models with motion input, the assembled state achieved by updating the block diagram (**Ctrl+D**) does not generally match the initial simulation state at time zero (**Ctrl+T**). This discrepancy is visible in Mechanics Explorer, where it can cause a sudden state change at time zero when simulating a model after updating it. It is also reflected in Model Report, whose initial state data does not generally apply to the simulation time zero when a model has motion inputs.

Motion-Controlled DOF Restriction

The number of degrees of freedom with prescribed trajectories must equal the number of degrees of freedom with automatically computed force or torque. In models not meeting this condition, simulation fails with an error.

See Also

Related Examples

- “Specify Joint Motion in Planar Manipulator Model” on page 3-114
- “Specify Joint Motion Profile” on page 3-110
- “Specifying Motion Input Derivatives” on page 3-24

More About

- “Specifying Joint Actuation Inputs” on page 3-19

Actuating and Sensing with Physical Signals

In this section...

“Exposing Physical Signal Ports” on page 3-28

“Converting Actuation Inputs” on page 3-28

“Obtaining Sensing Signals” on page 3-29

Some Simscape Multibody blocks provide physical signal ports for actuation input or sensing output. These ports accept or output only Simscape physical signals. If you wish to connect these ports to Simulink blocks, you must use the Simscape converter blocks. The table summarizes the converter blocks that Simscape provides. You can find both blocks in the Simscape Utilities library.

Block	Summary
PS-Simulink Converter	Convert Simscape physical signal into Simulink signal
Simulink-PS Converter	Convert Simulink signal into Simscape physical signal

Exposing Physical Signal Ports

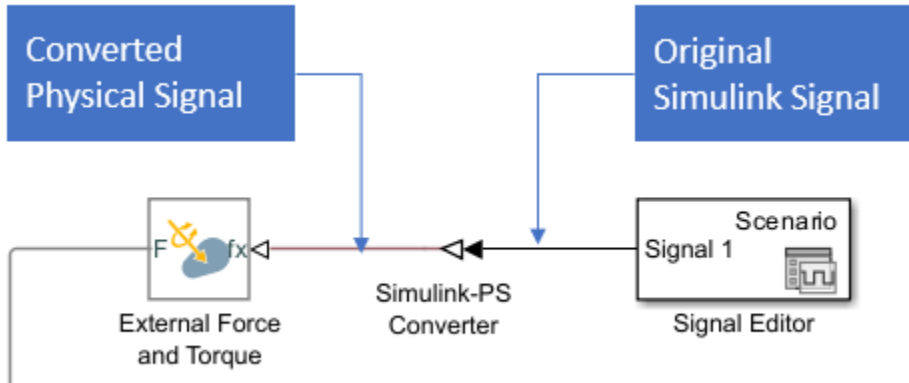
In Simscape Multibody, most physical signal ports are hidden by default. To expose them, you must select an actuation input or sensing output from the block dialog box. Blocks that provide physical signal ports include certain Forces and Torques blocks as well as Joint blocks. Each port has a unique label that identifies the actuation/sensing parameter. For the ports that a block provides, see the reference page for that block.

Converting Actuation Inputs

To provide an actuation signal based on Simulink blocks, you use the Simulink-PS Converter block:

- 1 Specify the desired actuation signal using Simulink blocks.
- 2 Connect the Simulink signal to the input port of a Simulink-PS Converter block.
- 3 Connect the output port of the Simulink-PS Converter block to the input port of the Simscape Multibody block that you want to provide the actuation signal to.

In the figure, the connection line that connects to the input port of the Simulink-PS Converter block represents the original Simulink signal. The connection line that connects to the output port of the same block represents the converted physical signal. This is the signal that you must connect to the actuation ports in Simscape Multibody blocks.

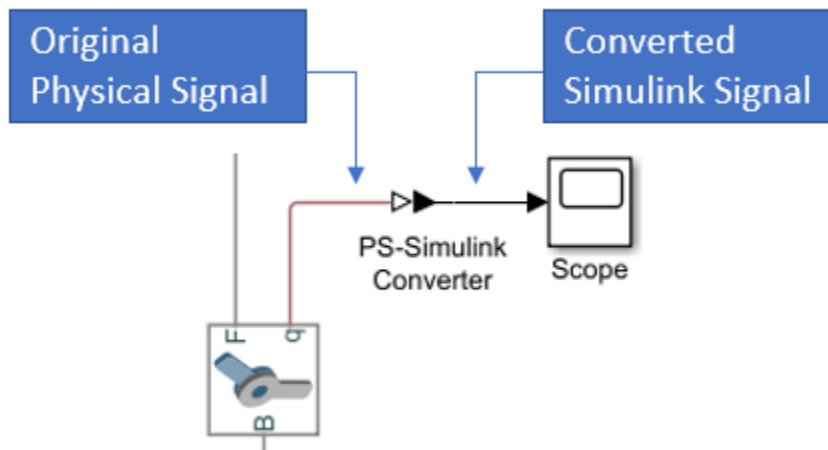


Obtaining Sensing Signals

To connect the sensing signal of a Simscape Multibody block to a Simulink block, you use the PS-Simulink Converter block:

- 1 Connect the Simscape Multibody sensing port to the input port of a PS-Simulink Converter block.
- 2 Connect the output port of the PS-Simulink Converter block to the Simulink block of your choice.

The figure shows how you can connect a Simscape Multibody sensing signal to a Simulink Scope block.



See Also

Related Examples

- “Specify Joint Motion in Planar Manipulator Model” on page 3-114
- “Specify Joint Motion Profile” on page 3-110
- “Specifying Motion Input Derivatives” on page 3-24

More About

- “Specifying Joint Actuation Inputs” on page 3-19

Sensing

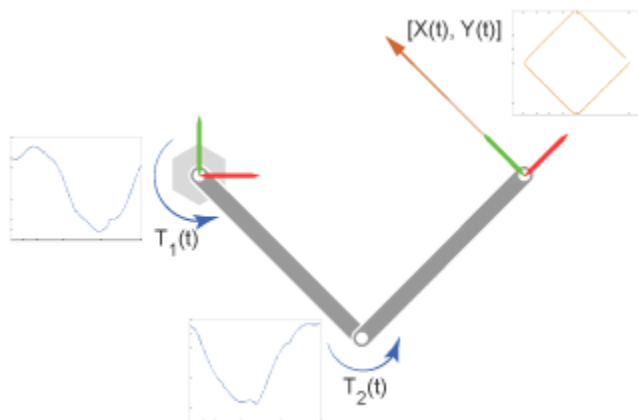
In this section...

“Sensing Overview” on page 3-31
 “Variables You Can Sense” on page 3-31
 “Blocks with Sensing Capability” on page 3-32
 “Sensing Output Format” on page 3-32

Sensing Overview

Sensing enables you to perform analytical tasks on a model. For example, you can perform inverse kinematic analyses of a robotic manipulator model. By prescribing the end-effector trajectory and sensing the joint actuation forces and torques, you can obtain the time-varying profile of each joint actuation input.

The variables you prescribe, the model inputs, and those you sense, the model outputs, determine which types of analysis you can perform. By changing the model inputs and outputs, you can perform numerous other analysis types. For example, to perform forward kinematic analysis on the robotic manipulator model, you can prescribe the manipulator joint trajectories and sense the resulting end-effector trajectory.



Variables You Can Sense

To support various analytical tasks, Simscape Multibody software provides a wide range of variables that you can sense. Each variable belongs to either of two categories:

- Motion variables — Linear and angular position, velocity, and acceleration. Linear variables are available in different coordinate systems, including Cartesian, spherical, and cylindrical. Angular variables are available in different formats, including quaternion, axis-angle, and transform matrix.
- Force and torque variables — Actuation, constraint, and total forces and torques acting at a joint, as well as certain forces and torques acting outside of a joint.

Blocks with Sensing Capability

The entire sensing capability spans multiple Simscape Multibody blocks. Two types of blocks provide motion sensing:

- Joint blocks — Motion sensing between the base and follower port frames of a joint block. Variables that you can sense are organized by joint primitive (prismatic, revolute, or spherical).
- Transform Sensor block — Motion sensing between any two frames in a model. This block provides the most comprehensive motion sensing capability in Simscape Multibody.

Three types of blocks provide force and torque sensing:

- Joint blocks — Actuation, constraint, and total force and torque sensing between the base and follower port frames. Actuation force and torque sensing is arranged by joint primitive.
- Constraint blocks — Constraint force and torque between the base and follower port frames.
- Certain Forces and Torques blocks — Total force the block exerts between the base and follower port frames. Only certain Forces and Torques blocks provide this type of sensing, such as the Spring and Damper Force and Inverse Square Law Force.

Sensing Output Format

Each sensing output is in a physical signal format. You can convert physical signals into Simulink signals using Simscape converter blocks, e.g., for plotting purposes using the Scope block. For information on how to use physical signals in Simscape Multibody models, see “Actuating and Sensing with Physical Signals” on page 3-28.

See Also

Inverse Square Law Force | Spring and Damper Force | Scope | Transform Sensor

More About

- “Actuating and Sensing with Physical Signals” on page 3-28

Force and Torque Sensing

In this section...

“Blocks with Force and Torque Sensing” on page 3-33

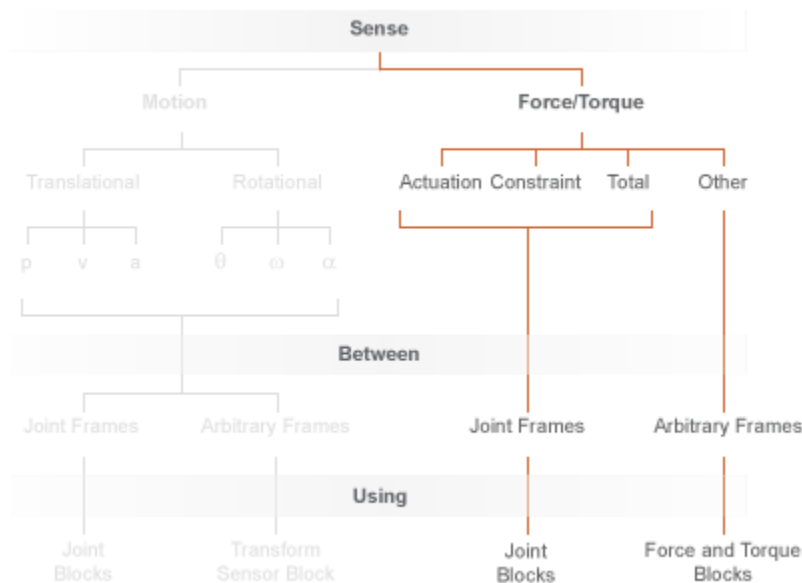
“Joint Forces and Torques You can Sense” on page 3-33

“Force and Torque Measurement Direction” on page 3-34

Blocks with Force and Torque Sensing

Blocks with force and torque sensing appear in two Simscape Multibody libraries:

- Forces and Torques — Sense the magnitude of certain forces not explicitly provided by input. Blocks with force sensing include Inverse Square Law Force and Spring and Damper Force. Each block can sense only the magnitude of its own force.
- Joints — Sense various forces and torques acting directly at a joint. All joint blocks provide force and torque sensing. However, the specific force and torque types that you can sense vary from joint to joint. Force and torque sensing is available strictly between the bodies the joint connects.



Force and Torque Sensing in Simscape Multibody

Joint Forces and Torques You can Sense

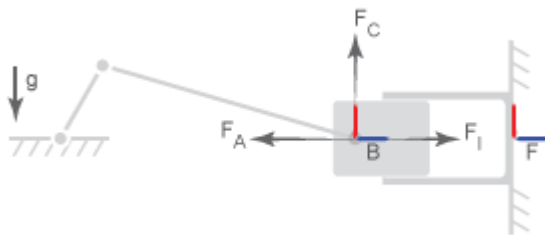
Forces and torques that you can sense at a joint fall into two categories:

- Joint primitive forces and torques. Each such force or torque is individually computed for a given joint primitive. Joint actuator forces and torques belong to this category.
- Composite forces and torques. Each such force or torque is computed in aggregate for an entire joint. Constraint and total forces and torques belong to this category.

The table summarizes the different joint forces and torques.

Force/Torque Type	Acts On	Measures
Actuator	Individual joint primitives	Force or torque driving an individual joint primitive. The sensed force or torque can be provided by input or it can be automatically computed based on joint motion inputs in a model.
Constraint	Entire joints	Aggregate constraint force or torque opposing motion normal to the joint degrees of freedom. By definition, these forces and torques act orthogonally to the joint primitive axes.
Total	Entire joints	Net sum of all forces or torques acting between the joint port frames. These include actuator, internal, and constraint forces and torques.

The figure shows a basic example of these forces acting on a crank-slider piston.



In the figure:

- F_A is the actuator force, which drives the piston toward the crank link.
- F_I is the internal spring and damper force, which resists motion of the piston with respect to the chamber.
- F_C is the constraint force, which opposes the effect of gravity on the piston, preventing it from falling.

The total force equals the net sum of F_A , F_I , and F_C .

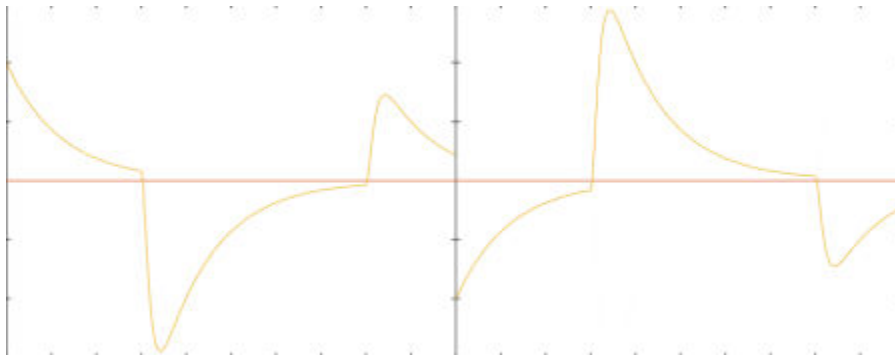
Force and Torque Measurement Direction

In accordance with Newton's third law of motion, a force or torque acting between two joint port frames accompanies an equal and opposite force or torque. If the base port frame of a Prismatic Joint block exerts a force on the follower port frame, then the follower port frame exerts an equal force on

the base frame. When sensing composite forces and torques in joint blocks, you can specify which of the two to sense:

- Follower on base — Sense the force or torque that the follower port frame exerts on the base port frame.
- Base on follower — Sense the force or torque that the base port frame exerts on the follower port frame.

The figure shows the effect of reversing the measurement direction. Reversing this direction changes the measurement sign.



Modeling Contact Force Between Two Solids

In this section...

“Spatial Contact Force Block Forces” on page 3-37

“Sensing” on page 3-37

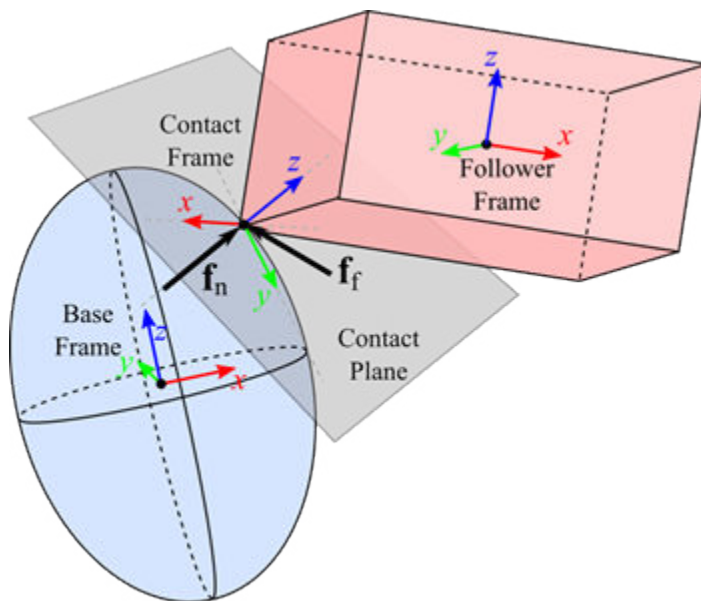
“Connect to Solid Blocks” on page 3-38

“Considerations for Contact Modeling” on page 3-38

When modeling solid blocks in contact with each other, contact forces play an important role in how the solid blocks behave. Both the normal force, f_n , and the frictional force, f_f , can cause the behavior of a model to drastically change. Contact forces come into play in many different modeling situations, such as:

- Package conveyors
- Robotic movement
- Race car dynamics

The Spatial Contact Force block models forces between base and follower frame solid bodies. When the two solid blocks are connected, the Spatial Contact Force block applies equal and opposite forces along a common contact plane. These forces conform to Newton's Third Law. The normal force is applied based on the penetration depth and the penetration velocity. If applied, the frictional force is based on the normal force and the relative velocities at the point of contact.



Simscape Multibody uses a *penalty method* for modeling contact between bodies, which allows the bodies to penetrate a small amount. Contact forces in the normal direction are computed using a spring-damper force law: the deeper the penetration and the greater the relative velocity in the penetration direction, the greater the normal contact forces.

Spatial Contact Force Block Forces

The Spatial Contact Force block properties are divided into three expandable nodes: **Normal Force**, **Frictional Force**, and **Sensing**. Set these properties when using the Spatial Contact Force block to model the contact forces between two solid bodies.

For more information about these properties, see Spatial Contact Force.

Normal Force

Parameters in the Normal Force section are used to determine the normal force, f_n , that the two solid bodies exert on each other. You can specify the **Stiffness**, **Damping**, and **Transition Region Width**.

Frictional Force

Parameters in the Frictional Force section are used to determine the frictional force, f_f , that the two solid bodies exert on each other. If you set **Method** to Smooth Stick-Slip, then you can specify the **Coefficient of Static Friction**, **Coefficient of Dynamic Friction**, and **Critical Velocity**. These options are not available when the **Method** is set to None.

Sensing

Besides modeling contact between two solid blocks, you can also use the Spatial Contact Force block to sense:

- Separation Distance—The distance between two solid bodies
- Normal Force—The normal force exerted by each solid body on the other.
- Frictional Force Magnitude—The frictional force exerted by each solid body on the other.

To enable these options, open the Spatial Contact Force block properties. Under **Sensing**, select the properties that you want the block to sense. For each property, a port is exposed on the block. Connect these ports to a viewer of your choice.

Normal Force

Stiffness	1e6	N/m	Compile-time
Damping	1e3	N/(m/s)	Compile-time
Transition Region Width	1e-4	m	Compile-time

Frictional Force

Method	Smooth Stick-Slip		
Coefficient of Static Friction	0.5		Compile-time
Coefficient of Dynamic Fricti...	0.3		Compile-time
Critical Velocity	1e-3	m/s	Compile-time

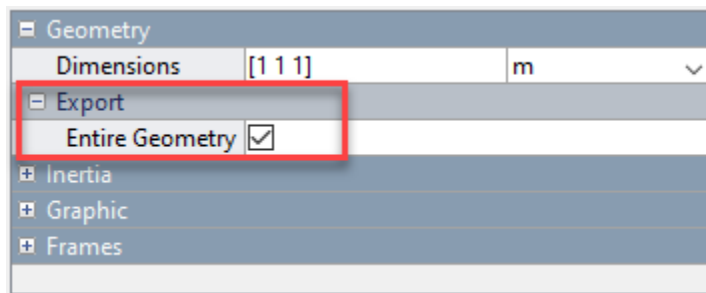
Sensing

<input checked="" type="checkbox"/> Separation Distance
<input checked="" type="checkbox"/> Normal Force Magnitude
<input checked="" type="checkbox"/> Frictional Force Magnitude

Zero-Crossings

Connect to Solid Blocks

The Spatial Contact Force block does not inherently know information about the two connecting solid bodies. In the solid block parameters, enable the **Export: Entire Geometry** option.



Once this option is enabled a new geometry port, **G**, appears on your solid block.



Connect the geometry port to the base or follower port on the Spatial Contact Force block. Follow the same steps for your other solid block and connect it to the remaining port. To learn more about base and follower frames, see “Selecting a Measurement Frame” on page 3-69.

Considerations for Contact Modeling

Compared to modeling the mechanical relationships in a multibody system, modeling contact presents more choices, and the chosen methods can impact simulation speed, accuracy, and model

maintainability. Simscape Multibody provides basic contact modeling constructs, but there are many ways to use them.

Solver Parameters

The computed normal contact forces are continuous functions of penetration depth and penetration velocity. Simscape Multibody computes these normal forces by reducing the spring and damper forces when the depth is less than the **Transition Region Width**. Increasing the **Transition Region Width** reduces the sharpness of the contact force, making the system easier for the solver to advance, but allowing greater amounts of penetration. Decreasing the **Transition Region Width** gives the contact forces a sharper profile, closer to idealized rigid-body contact. However, decreasing the **Transition Region Width** may also degrade solver performance.

Setting the maximum step size to 1e-3 seconds or lower improves accuracy in many models. Reducing the relative solver tolerance can produce a similar effect. However, reducing the step size also reduces the simulation speed.

Explicit solvers, such as `ode45`, are usually better for systems with many rapid collisions or contact changes. Implicit solvers may struggle under such circumstances. If the contact changes are infrequent and more stable, implicit solvers may offer a speed advantage due to their ability to handle the stiffness introduced by the contact forces.

Limit Penetration Depth

The separation distance between two solid blocks is positive when the two geometries are not in contact, zero when they are touching, and negative when there is nontrivial penetration between the geometries. When the separation distance is negative, its magnitude is also known as the *penetration depth*. Penetration depth is a continuous function of the positions and orientations of the geometries.

Simscape Multibody uses geometry analysis algorithms that are tuned for small contact-modeling applications. Some of the algorithms assume that the penetration depth is small compared to the size of the geometries. If this is not the case, the computed (negative) separation distances may only be approximate, or may exhibit discontinuities, and the resulting contact forces may be erratic. For best results, limit penetration depth in your model.

See Also

Spatial Contact Force | Brick Solid | Cylindrical Solid | Spherical Solid

More About

- “Model Wheel Contact in a Car” on page 3-49
- “Train Humanoid Walker” on page 8-114
- “Zero-Crossing Detection”
- “Use Contact Proxies to Simulate Contact” on page 3-40

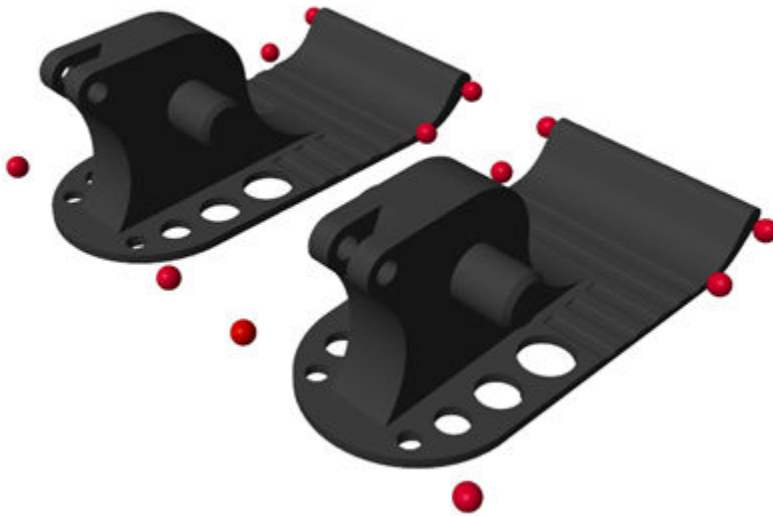
Use Contact Proxies to Simulate Contact

In this section...

“How to Use Contact Proxies” on page 3-40

“Examples” on page 3-41

Contact proxies are simple shapes that are used to represent the contact parts of actual bodies. For example, in the “Train Humanoid Walker” on page 8-114 example, red spheres are used to represent the bottoms of the robotic feet.



By using contact proxies, you can increase the speed and robustness of a contact simulation. For example, using contact proxies can speed up models that involve complex geometries and prevent the discontinuous jumping of contact force locations in static contact simulations.

How to Use Contact Proxies

Contact proxies can be used in many cases. However, adding contact proxies requires a greater modeling effort, and it is hard to leverage the proxies in several cases. If the contact interaction involves all the features of the actual bodies, then proxies may not be able to completely represent every element of the bodies. If your model is simple and you only need to run it once, you can use the actual bodies to model contact.

How to Choose Proxies

To choose the appropriate proxies for a contact problem, you need to indicate what parts of the actual bodies will interact and then select the simplest proxies for these parts. Note that the proxies should be enough to cover all the contact regions.

Simscape Multibody supports a variety of geometries and bodies for contact modeling. Based on complexity, these geometries and bodies can be categorized into three groups.

- The geometries defined by the Point, Infinite Plane, and Spherical Solid blocks. These geometries are the best candidates to act as proxies due to their simple shapes and high modeling efficiency.

- The bodies defined by Brick Solid, Cylindrical Solid, and Ellipsoidal Solid blocks. They are more computationally expensive than the geometries of the first group. If you can't use the geometries in the first group, try to use bodies in this group as proxies.
- The geometries created by extrusions, revolutions, and CAD imports. Typically, these geometries are not suitable to be used as proxies due to their complex shapes, high computational costs, and simplified representations. Note that Simscape Multibody represents some of these geometries with convex hulls instead of actual bodies when modeling contacts.

Tips for Using Contact Proxies in Complex Models

Simscape Multibody models contact between bodies by using the Spatial Contact Force block. Each pair of potentially contacting parts needs one Spatial Contact Force block. Consequently, a complex model that includes many proxies could lead to a plethora of Spatial Contact Force blocks and geometry lines. Use the following techniques to keep your block diagram organized. See the first example for how to use these techniques.

- Place each actual body and its proxies in one subsystem. The subsystem should also include all the relevant Rigid Transform blocks needed to properly place the proxies relative to the actual body.
- Copy and paste subsystems or use referenced subsystems to create identical subsystems.
- Use the Simscape Bus block to bundle the geometry lines of a complex model.

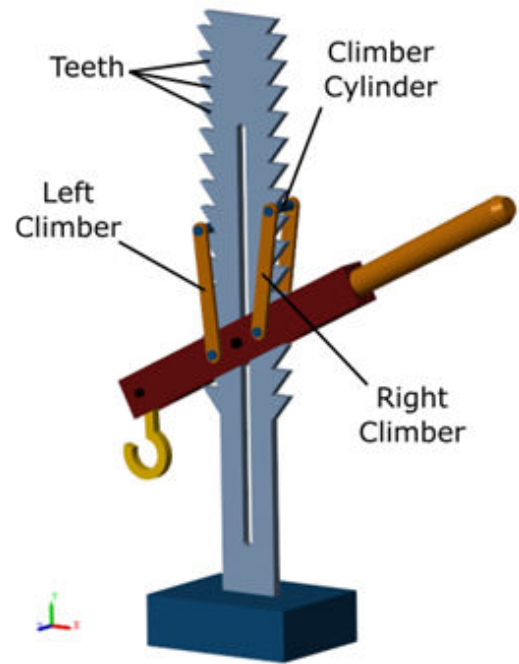
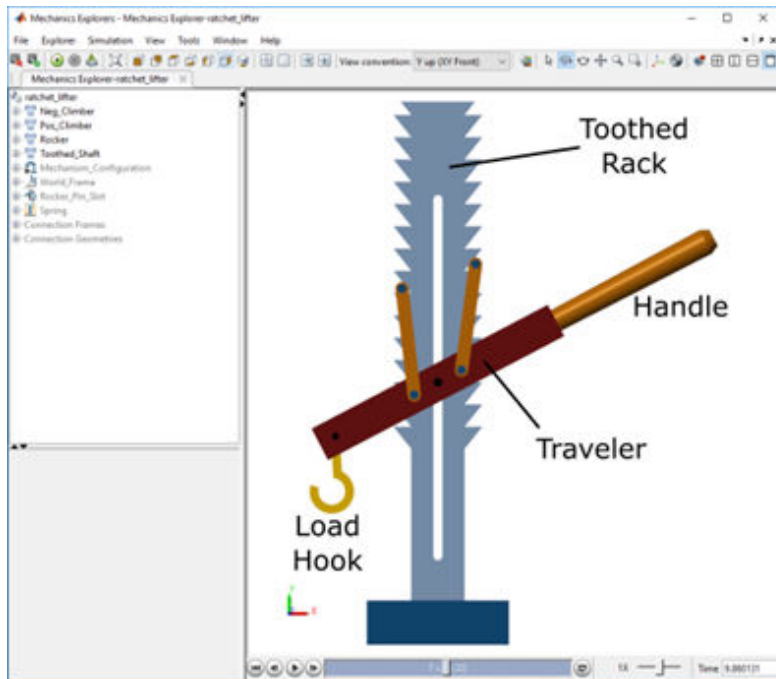
Usually, the actual body and its proxies occupy some common regions in the 3-D space. You should render only the actual body and hide the proxies in the final version of the simulation. However, during a modeling or debugging step, it is helpful to view the proxies to verify whether all the contacts are modeled as expected. Consider defining two variables to adjust the transparency of the actual body and proxies. Use the mass of the actual body for simulation and set the mass or density of the contact proxies to zero to avoid any effects on dynamics.

Examples

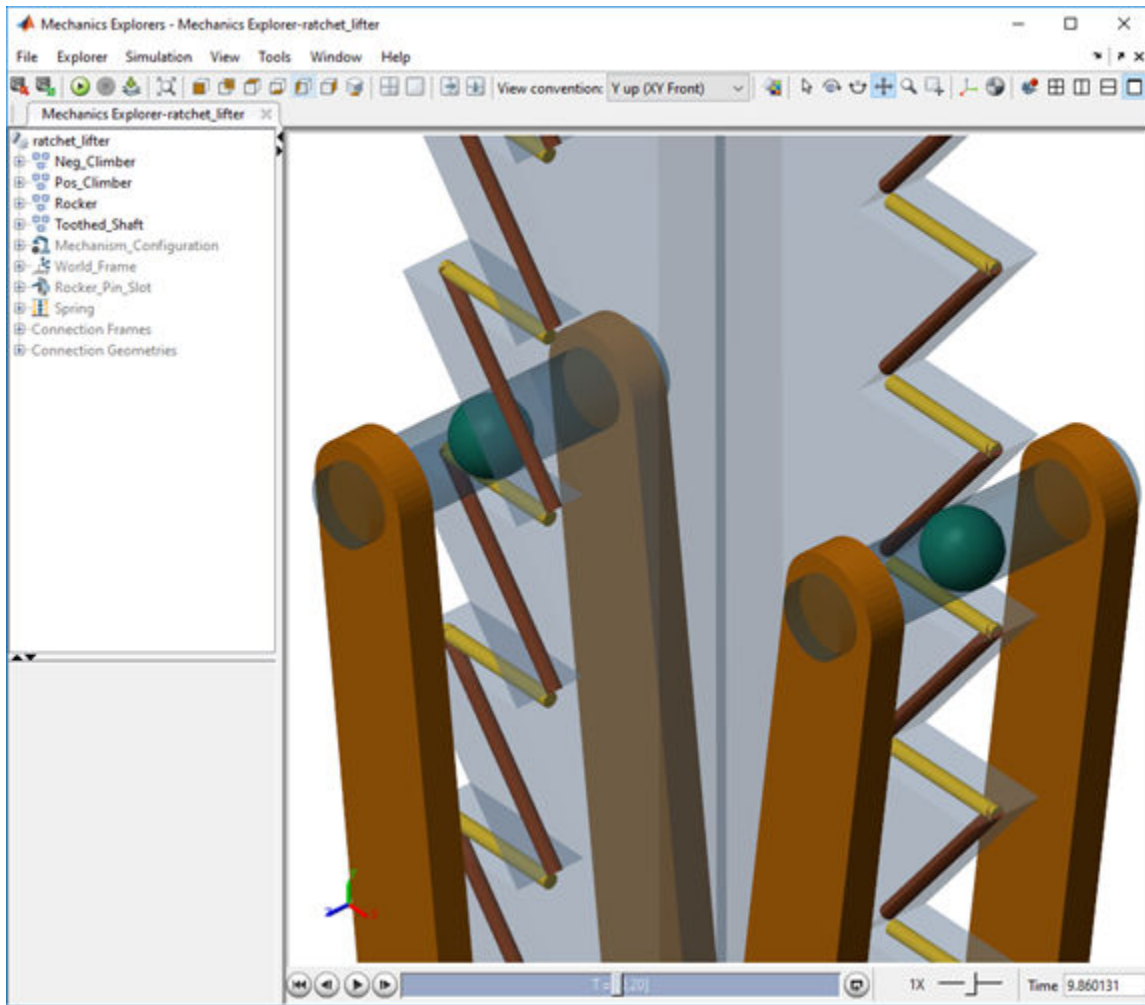
Using Proxies in Contact Models that Include Complex Shapes

Modeling contact between bodies with complex shapes is computationally expensive and time consuming. To speed up the model, you can decompose the complex shapes into simpler parts and use proxies to represent the parts involved in contact, then model the contacts between these proxies.

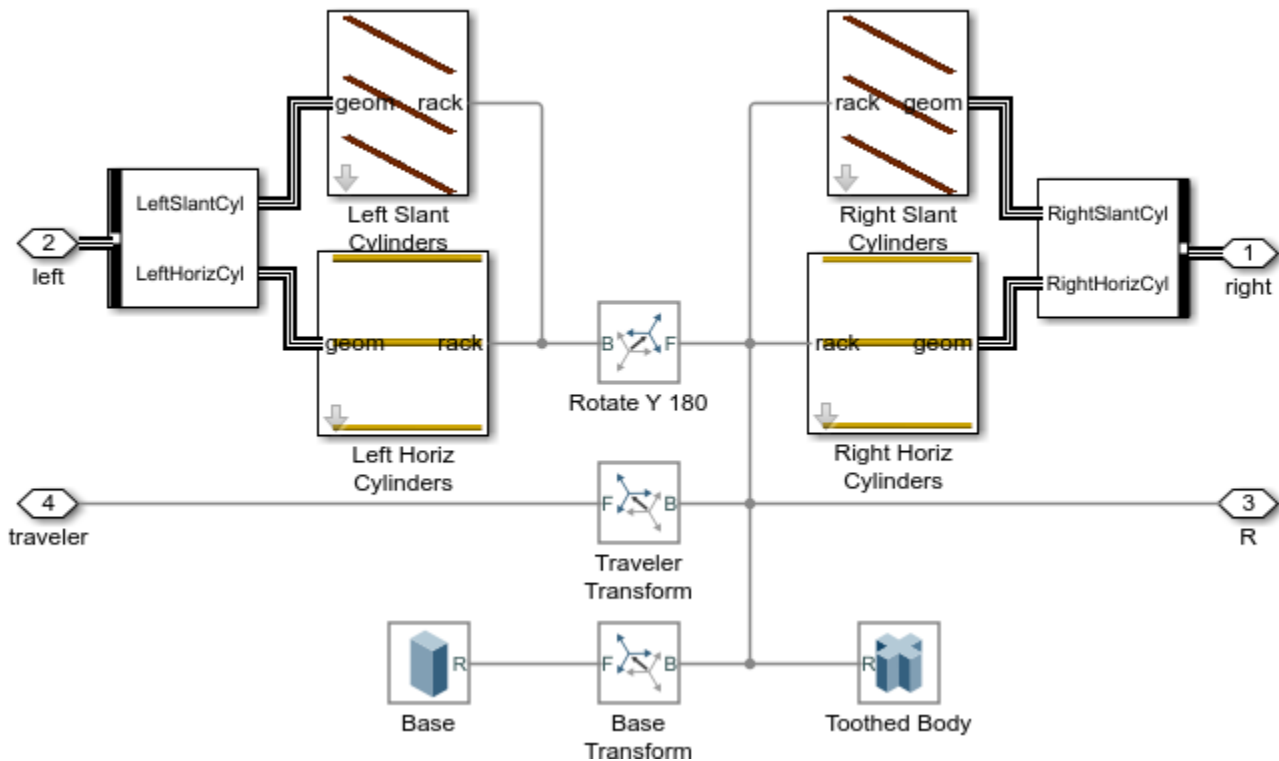
In the “Ratchet Lifter” on page 8-124 example, the traveler climbs up along the toothed rack as the traveler handle is pumped up and down. Only the horizontal and slanted surfaces of the rack teeth and climber cylinders are involved in the contact. Also, the traveler only exhibits planar motion. In other words, only the center parts of the climber cylinders are involved in the contact interactions. Therefore, the example uses the proxies to represent only these parts and models contacts among the proxies.



This example uses cylinders to represent the horizontal and slanted surfaces of the rack teeth and spheres to represent the center parts of the climber cylinders. The advantage of using a sphere instead of the entire climber cylinder is that spheres are simpler than cylinders and provide more efficient contact modeling.

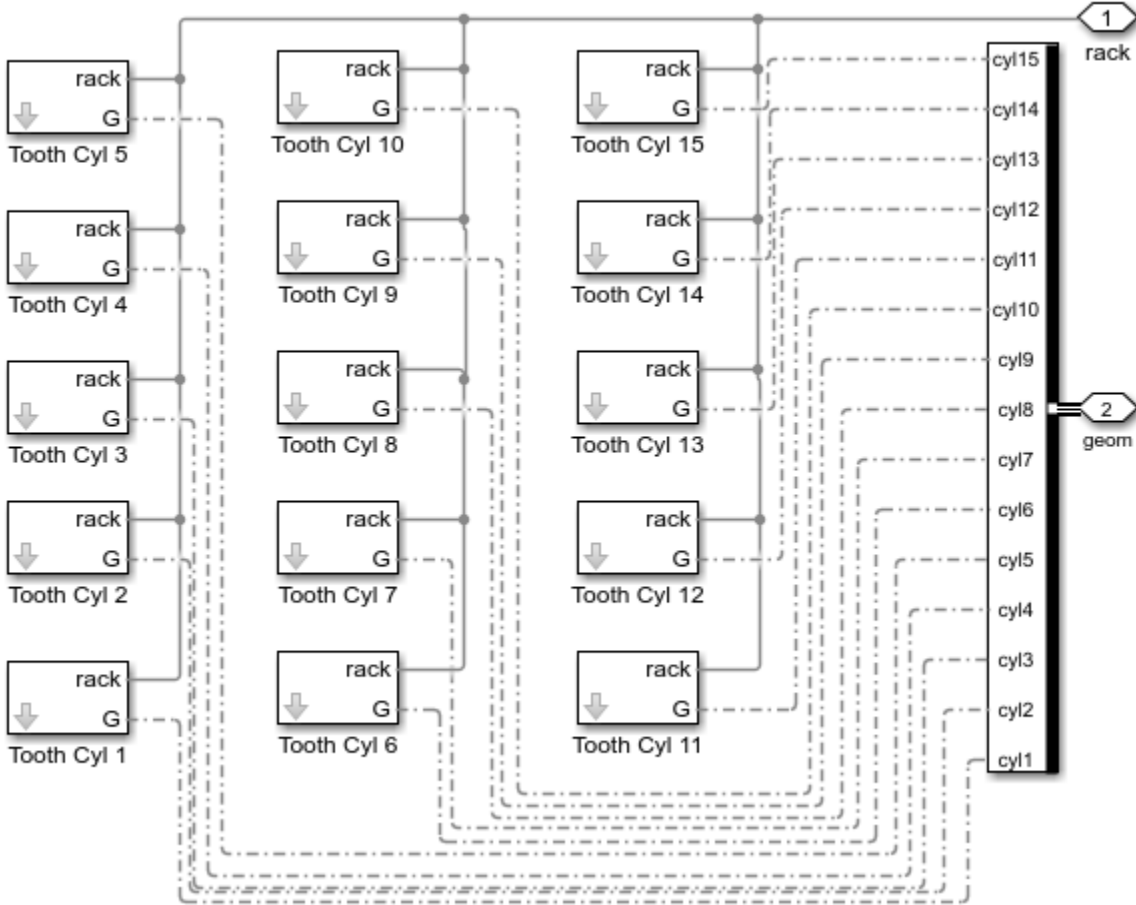


The toothed rack is symmetrical and has 15 teeth on each side. Consequently, this example uses 60 cylinders to represent the teeth. The model includes a subsystem called Toothed Rack that includes the actual body of the rack, all the proxies of the teeth, and relevant Rigid Transform blocks. The following figure shows the Toothed Rack subsystem.

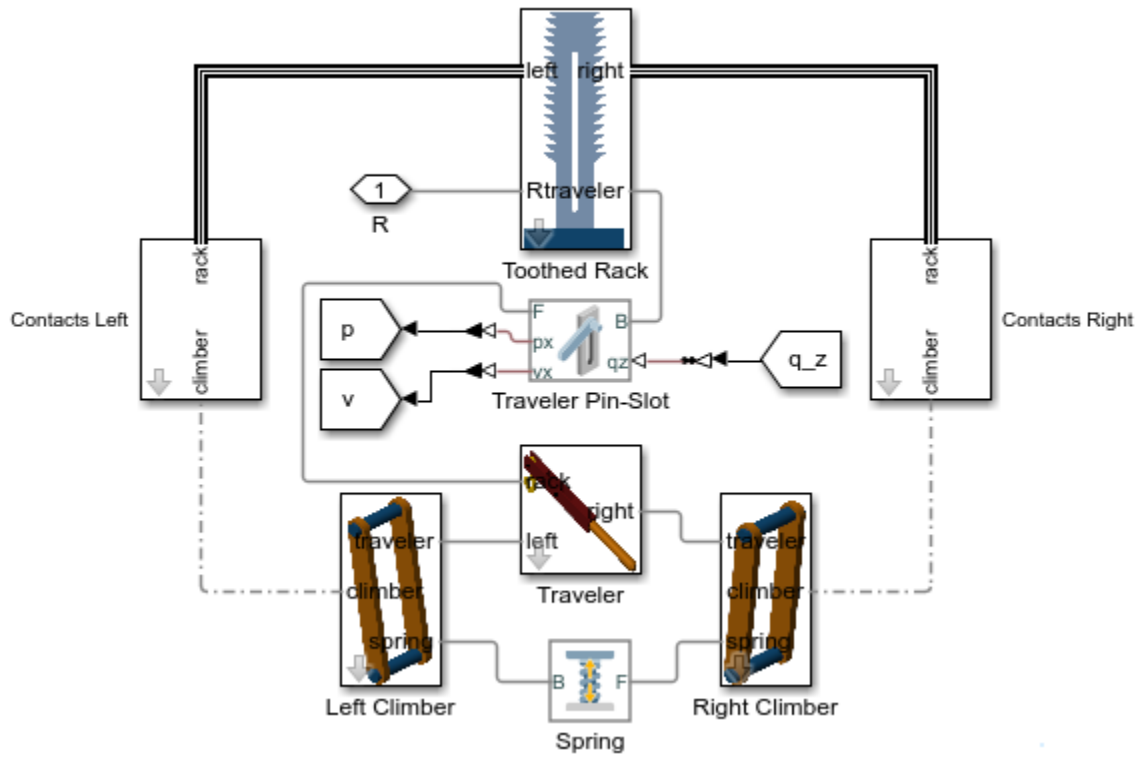


Because the toothed rack is symmetrical, only the proxies on one side of the rack were created manually and grouped into the Left Slant Cylinders and Left Horiz Cylinders subsystems. Then these subsystems were copied, pasted, and then rotated 180 degrees around the y-axis to represent the right-side teeth.

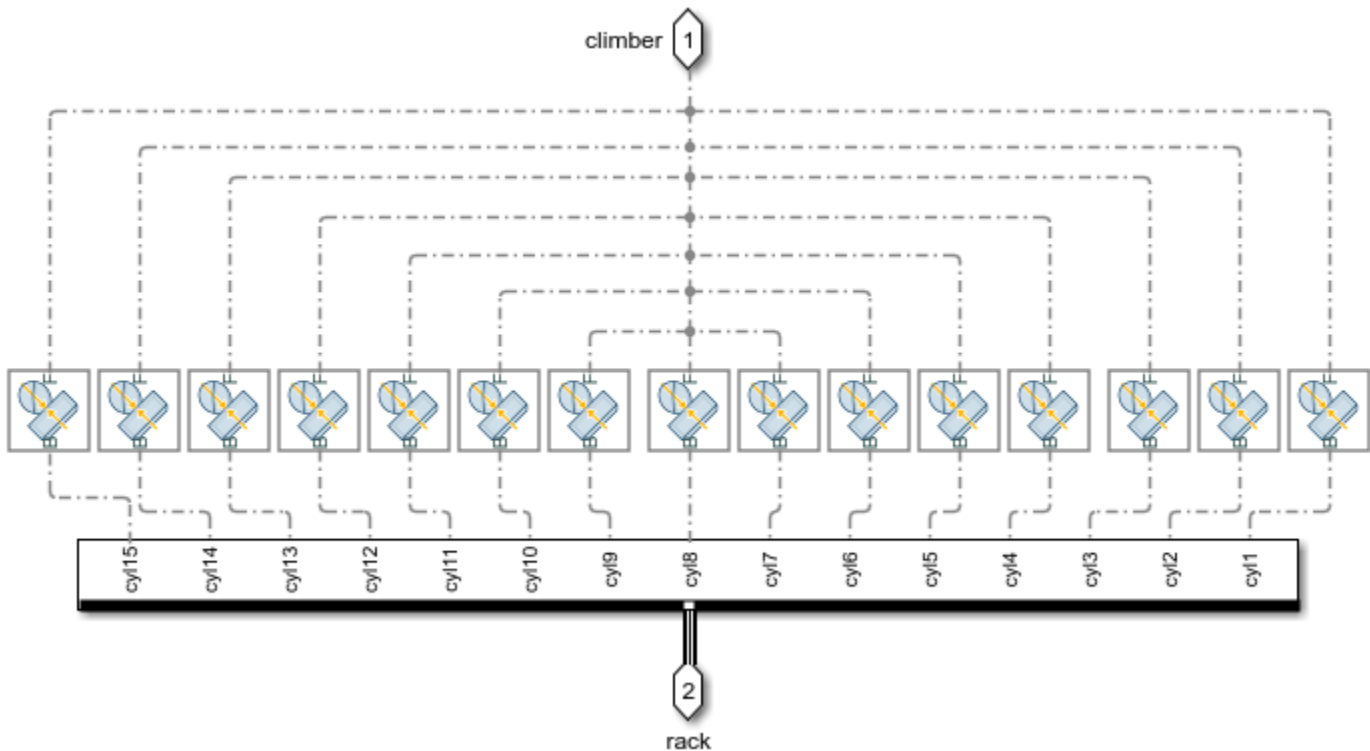
The following image shows the Left Horiz Cylinders subsystem. The model uses 15 identical cylinders to represent the horizontal surfaces of the left-side teeth. To minimize the number of blocks that were manually created, one parameterized referenced subsystem was created to model one of the cylinders. Then the parameterized referenced subsystem was copied 14 times. The parameterized referenced subsystems use the cylinders' indices to specify the locations of the cylinders. Finally, the geometry lines of the cylinders are bundled using a Simscape Bus block. The same method is used to model the subsystem of the teeth's slant surfaces.



The Simscape Bus block helps avoid a complex web of lines throughout the model. The following image shows that two Simscape Bus lines connect the 60 proxies of the rack teeth with the climbers' proxies via the Contacts Left and Contacts Right subsystems.



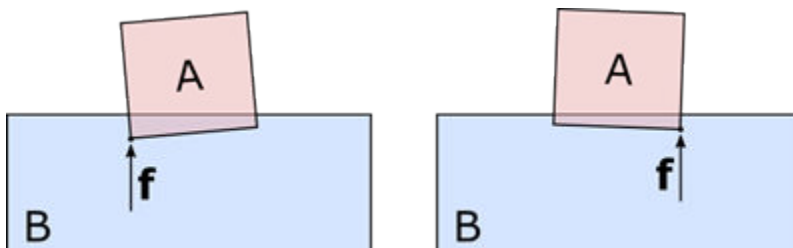
The Contacts Left and Contacts Right subsystems include all the Spatial Contact Force blocks for this simulation. The following image shows the diagram of the contact between the horizontal surface of the left-side teeth and the cylinder of the left climber.



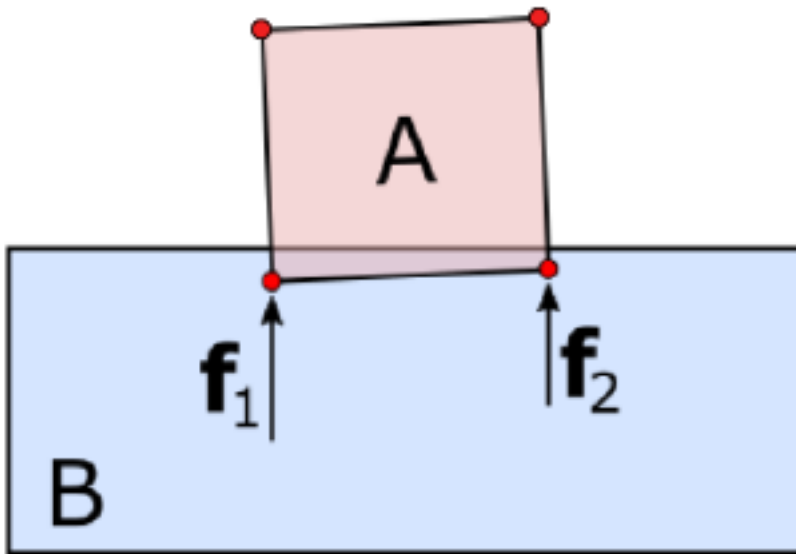
Using Proxies to Model Static Contact

Simscape Multibody employs a point-based penalty method for modeling contact between bodies. This method means that the Spatial Contact Force block applies necessary contact forces to its connected bodies at the points with the maximum penetration between the two bodies. Each Spatial Contact Force block only applies a single contact force for each body at each time step.

This point-based method has challenges when modeling static contact between the flat surfaces of bodies. For example, consider modeling the contact between two bricks. For the purposes of simplicity, the following image shows the 2-D version of the problem. Suppose Brick A is dropped onto Brick B, which has a fixed position. The left image shows the configuration of the bricks when the contact is first detected. At this point, the Spatial Contact Force block applies a contact force to the lower-left corner of Brick A because it has the maximum penetration. Over time, the force decelerates the downward motion of Brick A and rotates it in a clockwise direction. Then, the configuration appears like the right image, and the force is applied to the lower right corner of Brick A. As Brick A settles into static contact with Brick B, the location of the contact force rapidly and discontinuously jumps among the corners of Brick A. This behavior is challenging for the solver and can significantly decrease the simulation speed.



Using contact proxies is one effective way to avoid the above modeling challenges. Eight small spheres can be rigidly attached to the corners of Brick A. With these proxies, the contact is modeled as eight brick-sphere pairs instead of a single brick-brick pair. As the contact stabilizes, the normal force at each bottom corner will be one-fourth of the weight of Brick A.



See Also

Spatial Contact Force | Brick Solid | Cylindrical Solid | Spherical Solid | Point | Infinite Plane | Ellipsoidal Solid | Simscape Bus | Extruded Solid | File Solid

More About

- “Modeling Contact Force Between Two Solids” on page 3-36
- “Model Wheel Contact in a Car” on page 3-49
- “Train Humanoid Walker” on page 8-114
- “Zero-Crossing Detection”

Model Wheel Contact in a Car

In this section...

“Model a Rolling Wheel” on page 3-49

“Model a Bumper Car” on page 3-52

This example shows how to create a system that models a wheel rolling down an inclined plane by using the Spatial Contact Force block.

Model a Rolling Wheel

To create a new Simscape Multibody model, at the MATLAB command prompt, enter:

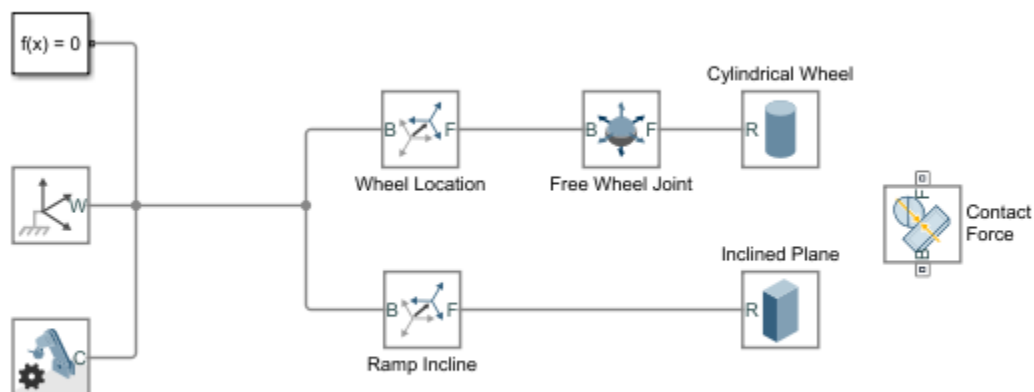
```
smnew % create new Simscape Multibody model
```

Save your model.

In the model, add:

- One Rigid Transform blocks
- One 6-DOF Joint block
- One Cylindrical Solid block
- One Spatial Contact Force block

Delete Scope, PS-Simulink Converter, and Simulink-PS Converter blocks. Rename and connect the blocks as showing in following figure.



Assign these properties to Wheel Location:

Property	Value
Rotation > Method	Aligned Axes

Property	Value
Rotation > Pair 1 > Follower	+Z
Rotation > Pair 1 > Base	-Y
Rotation > Pair 2 > Follower	+X
Rotation > Pair 2 > Base	+X
Translation > Method	None

Assign these properties to Ramp Incline:

Property	Value
Rotation > Method	Standard Axis
Rotation > Axis	+Y
Rotation > Angle	5 deg
Translation > Method	Cartesian
Translation > Offset	[30 0 -15] cm

Assign these properties to Cylindrical Wheel:

Property	Value
Geometry > Radius	5 cm
Geometry > Length	4 cm
Geometry > Export > Entire Geometry	selected
Inertia > Type	Calculate from Geometry
Inertia > Based on	Density
Inertia > Density	650 kg/m ³
Graphic > Type	From Geometry
Graphic > Visual Properties	Simple
Graphic > Visual Properties > Color	[0.6 0.0 0.0]
Graphic > Visual Properties > Opacity	1.0
Frames > Show Port R	selected

Assign these properties Inclined Plane:

Property	Value
Geometry > Dimensions	[90 20 5] cm
Geometry > Export > Entire Geometry	selected
Inertia > Type	Calculate from Geometry
Inertia > Based on	Density
Inertia > Density	1000 kg/m ³
Graphic > Type	From Geometry
Graphic > Visual Properties	Simple

Property	Value
Graphic > Visual Properties > Color	[0.4196 0.5569 0.1373]
Graphic > Visual Properties > Opacity	1.0
Frames > Show Port R	selected

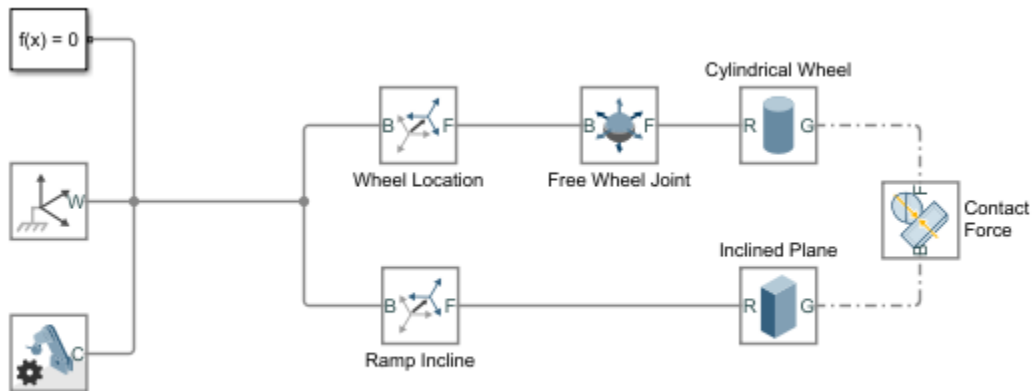
Assign these properties to Contact Force:

Property	Value
Normal Force > Stiffness	1e6 N/m
Normal Force > Damping	1e3 N/(m/s)
Normal Force > Normal Force: Transition Region Width	1e-4 m
Frictional Force > Method	Smooth Stick-Slip
Frictional Force > Coefficient of Static Friction	0.3
Frictional Force > Coefficient of Dynamic Friction	0.3
Frictional Force > Critical Velocity	0.01 m/s
Sensing > Separation Distance	unselected
Sensing > Normal Force	unselected
Sensing > Frictional Force Magnitude	unselected

On the **Modeling** tab, select **Model Settings > Model Settings** to open the Configuration Parameters. In the Solver pane, under **Solver details**, update the following:

Max step size:	1e-3
Absolute tolerance:	1e-3

At this point, both the Cylindrical Wheel block and Inclined Plane block should have a geometry port. As shown in the figure, connect the geometry ports of the Inclined Plane and Cylindrical Wheel blocks to the base and follower ports of the Spatial Contact Force block, respectively.



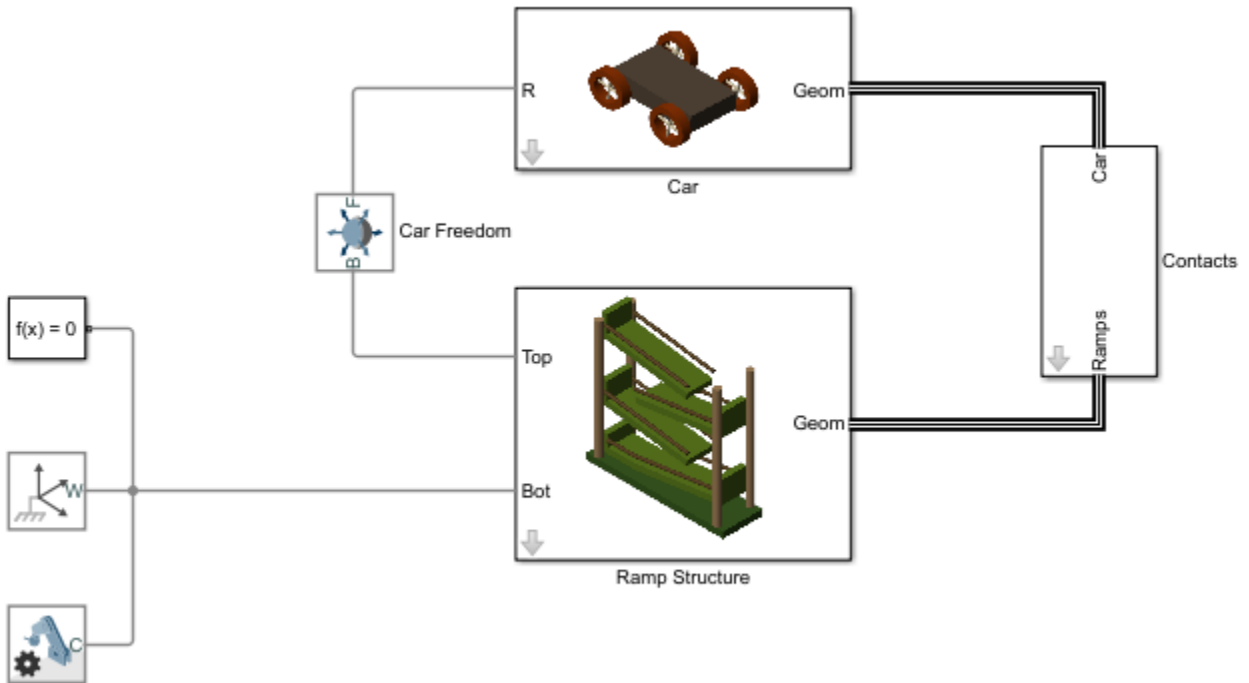
On the **Simulation** tab, click **Run**. In the MATLAB window, the Mechanics Explorer pane opens, and you see the cylindrical wheel roll down the surface.

Model a Bumper Car

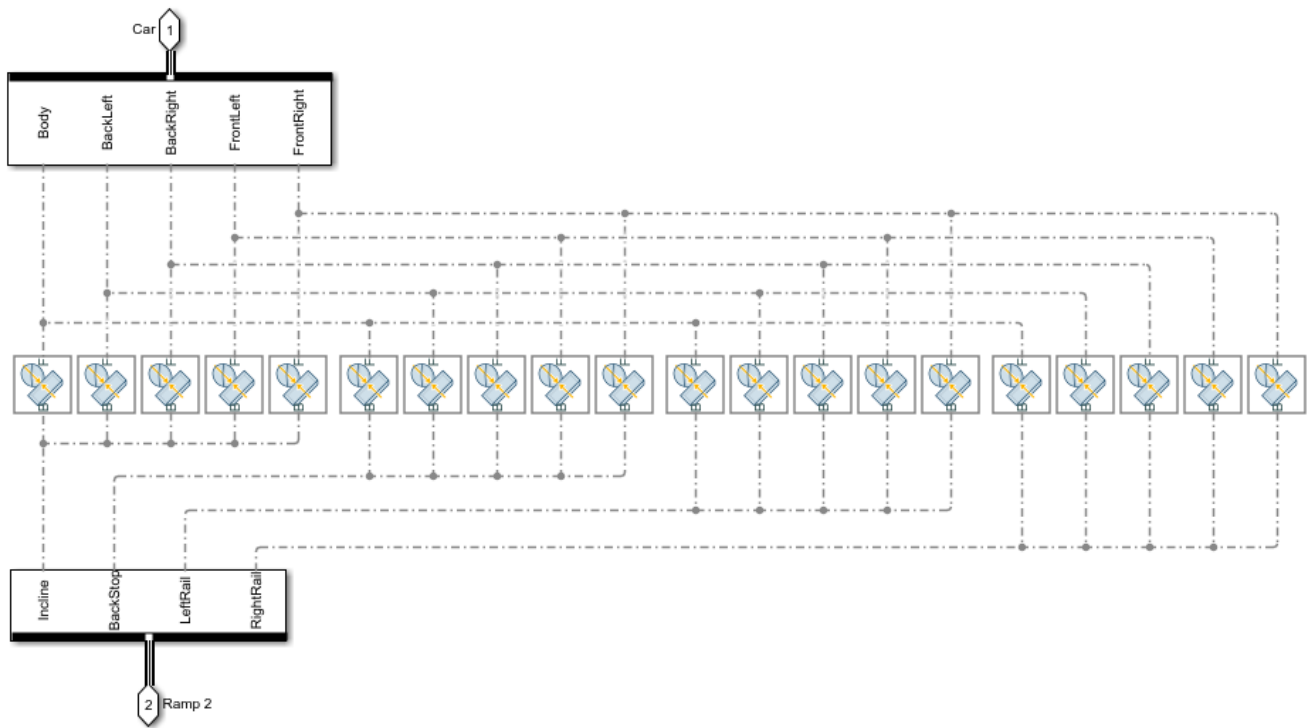
The modeling process described above can be used to develop more complicated models that include contact forces. To see a more complicated model, at the MATLAB command prompt, enter:

```
sm_bumper_car % open Simscape Multibody bumper car model
```

Open Bumper Car Playset.



The model is made of two structures: the Car and the Ramp Structure. The Spatial Contact Force blocks that are used to model the contact forces between each wheel of the car and the ramps are housed in the Car to Ramp subsystems.



To simulate the model, in the **Simulation** tab, click **Run**. In the Mechanics Explorer, you can see the bumper car rolling down multiple ramps.



See Also

More About

- Spatial Contact Force
- Brick Solid
- Cylindrical Solid
- Spherical Solid

Motion Sensing

In this section...

“Sensing Spatial Relationships Between Joint Frames” on page 3-56

“Sensing Spatial Relationships Between Arbitrary Frames” on page 3-57

In Simscape Multibody, you can sense the spatial relationship between two frames using two types of blocks:

- **Transform Sensor** — Sense the spatial relationship between any two frames in a model. Parameters that you can sense with this block include position, velocity, and acceleration of the linear and angular types. This block provides the most extensive motion sensing capability in the Simscape Multibody libraries.
- **Joint blocks** — Sense the spatial relationship between the base and follower frames of a Joint block. Parameters that you can sense with a Joint block include the position and its first two time derivatives (velocity and acceleration) for each joint primitive.

These blocks output a physical signal for each measurement that you specify. You can use the sensing output of these blocks for analysis or as input to a control system in a model.

Sensing Spatial Relationships Between Joint Frames

To sense the spatial relationship between the base and follower frames of a Joint block, you can use the Joint block itself. For each joint primitive, the dialog box provides a **Sensing** menu with basic parameters that you can measure. These parameters include the position, velocity, and acceleration of the follower frame with respect to the base frame. If the sensing menu of the dialog box does not provide the parameters that you wish to sense, use the Transform Sensor block instead. See “Sensing Spatial Relationships Between Arbitrary Frames” on page 3-57.

The sensing capability of a joint block is limited to the base and follower frames of that joint block. Every measurement provides the value of a parameter for the joint follower frame with respect to the joint base frame. If sensing the spatial relationship with a spherical joint primitive, you can also select the frame to resolve the measurement in. To sense the spatial relationship between any other two frames, use the Transform Sensor block instead.

If the joint primitive is of the revolute or spherical type, the parameters correspond to the rotation angle, angular velocity, and angular acceleration, respectively. If the joint primitive is of the prismatic type, the parameters correspond to the offset distance, linear velocity, and linear acceleration, respectively.

Regardless of joint primitive type, each parameter that you select applies only to the joint primitive it belongs to. For example, selecting **Position** in the **Z Revolute Primitive (Rz) > Sensing** menu exposes a physical signal port that outputs the rotation angle of the follower frame with respect to the base frame *about the base frame Z axis*.

The table lists the port label for each parameter that you can sense using a joint block. The first column of the table identifies the parameters that you can select. The remaining three columns identify the port labels for the three joint primitive menus that the dialog box can contain: **Spherical**, **Revolute**, and **Prismatic**.

Note For parameter descriptions, see the reference pages for Spherical Joint, Revolute Joint, and Prismatic Joint blocks.

Parameter	Spherical	Revolute	Prismatic
Position	Q	q	p
Velocity	w	w	v
Velocity (X/Y/Z)	wx/wy/wz	N/A	N/A
Acceleration	b	b	a
Acceleration (X/Y/Z)	bx/by/bz	N/A	N/A

A joint block can contain multiple revolute and prismatic joint primitives. For blocks with multiple primitives of the same type, the port labels include an extra letter identifying the joint primitive axis. For example, the **Position** port label for the Z prismatic primitive of a Cartesian Joint block is pz.

Select Joint Parameters To Sense

To select the spatial relationship parameters that you wish to sense:

- 1 Open the dialog box for the joint block to sense the spatial relationship across.
- 2 In the **Sensing** menu of the block dialog box, select the parameters to sense.

The block exposes one physical signal port for each parameter that you select. The label of each port identifies the parameter that port outputs.

Sensing Spatial Relationships Between Arbitrary Frames

To sense the spatial relationship between two arbitrary frames in a model, you use the Transform Sensor block. The dialog box of this block provides a set of menus that you can use to select the parameters to sense. These parameters include position, velocity, and acceleration of the linear and angular types.

Every measurement provides the value of a parameter for the follower frame with respect to the base frame, resolved in the measurement frame that you choose. You can connect the base and follower frame ports of the Transform Sensor block to any two frames in a model. To measure a parameter for a different frame, connect the follower frame port to the frame line or port that identifies that frame. Likewise, to measure a parameter for the same frame but with respect to a different frame, connect the base frame port to the frame line or port that identifies that frame. Finally, to resolve a measurement in a different frame, select a different measurement frame in the block dialog box. For more information about measurement frames, see “Selecting a Measurement Frame” on page 3-69. For more information about frame lines and ports, see “Working with Frames” on page 1-24.

Selecting a parameter from the block dialog box exposes the corresponding physical signal port in the block. Use this port to output the measurement for that parameter. To identify the port associated with each parameter, each port uses a unique label.

The table lists the port labels for each angular parameter that you can sense. The first column of the table identifies the parameters that you can select. The remaining three columns identify the port labels for the three angular parameter menus in the dialog box: **Rotation**, **Angular Velocity**, and **Angular Acceleration**. Certain parameters belong to one menu but not to others. N/A identifies the

parameters that do not belong to a given menu—e.g. Angle, which is absent from the Angular Velocity.

Note For parameter descriptions, see the Transform Sensor reference page.

Parameter	Rotation	Angular Velocity	Angular Acceleration
Angle	q	N/A	N/A
Axis	axs	N/A	N/A
Quaternion	Q	Qd	Qdd
Transform	R	Rd	Rdd
Omega X/Omega Y/ Omega Z	N/A	wx/wy/wz	N/A
Alpha X/Alpha Y/Alpha Z	N/A	N/A	bx/by/bz

The table lists the port labels for each linear parameter that you can sense. As in the previous table, the first column identifies the parameters that you can select. The remaining three columns identify the port labels for the three linear parameter menus in the dialog box: **Translation**, **Velocity**, and **Acceleration**.

Parameter	Rotation Port	Angular Velocity Port	Angular Acceleration Port
X/Y/Z	x/y/z	vx/vy/vz	ax/ay/az
Radius	rad	vrad	arad
Azimuth	azm	vazm	aazm
Distance	dst	vdst	adst
Inclination	inc	vinc	ainc

Select Transform Sensor Parameters To Sense

To select the spatial relationship parameters that you wish to sense:

- 1 Open the Transform Sensor dialog box.
- 2 Expand the menu for the parameter group that parameter belongs to.
E.g. **Rotation** for parameter **Angle**.
- 3 Select the check box for that parameter.

The block exposes one physical signal port for each parameter that you select. The label of each port identifies the parameter that port outputs.

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-79

- “Specify Joint Actuation Torque” on page 3-84

More About

- “Rotational Measurements” on page 3-60
- “Translational Measurements” on page 3-65
- “Selecting a Measurement Frame” on page 3-69

Rotational Measurements

In this section...

“Rotation Sensing Overview” on page 3-60

“Measuring Rotation” on page 3-60

“Axis-Angle Measurements” on page 3-60

“Quaternion Measurements” on page 3-61

“Transform Measurements” on page 3-61

“Rotation Sequence Measurements” on page 3-62

Rotation Sensing Overview

You can measure frame rotation in different formats. These include axis-angle, quaternion, transform, and rotation sequence. The different formats are available through the Transform Sensor block and, to a limited extent, in joint blocks ¹. The choice of measurement format depends on the model. Select the format that is most convenient for the application.

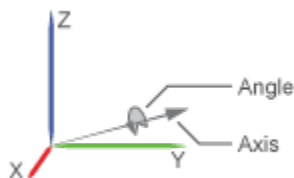
Measuring Rotation

Rotation is a relative quantity. The rotation of one frame is meaningful only with respect to another frame. As such, blocks with rotation sensing capability require two frames to make a measurement: measured and reference frames. In these blocks, the follower frame port identifies the measured frame; the base frame port identifies the reference frame of the measurement.

Simscape Multibody defines the rotation formats according to standard conventions. In some cases, more than one convention exists. This is the case, for example, of the quaternion. To properly interpret rotation measurements, review the definitions of the rotation formats.

Axis-Angle Measurements

Axis-angle is one of the simpler rotation measurement formats. This format uses two parameters to completely describe a rotation: axis vector and angle. The usefulness of the axis-angle format follows directly from Euler’s rotation theorem. According to the theorem, any 3-D rotation or rotation sequence can be described as a pure rotation about a single fixed axis.



To measure frame rotation in axis-angle format, use the Transform Sensor block. The block property inspector contains separate **Axis** and **Angle** parameters that you can select to expose the

¹ Weld Joint is an exception

corresponding physical signal (PS) ports (labeled a_x and q , respectively). Because the axis-angle parameters are listed separately, you can choose to measure the axis, the angle, or both.

The axis output is a 3D unit vector in the form $[a_x, a_y, a_z]$. This unit vector encodes the rotation direction according to the right-hand rule. For example, a frame spinning in a counterclockwise direction about the +X axis has rotation axis $[1\ 0\ 0]$. A frame spinning in a clockwise direction about the same axis has rotation axis $[-1\ 0\ 0]$.

The angle output is a scalar number in the range $0-\pi$. This number encodes the extent of rotation about the measured axis. By default, the angle is measured in radians. You can change the angle units in the PS-Simulink Converter block used to interface with Simulink blocks.

Quaternion Measurements

The quaternion is a rotation representation based on hypercomplex numbers. The quaternion is made up of a scalar part, S , and a vector, V , part. The scalar part encodes the angle of rotation, and the vector part encodes the rotational axis.

A key advantage of quaternions is the singularity-free parameter space. Mathematical singularities, which are present in Euler angle sequences, result in the loss of rotational degrees of freedom. This phenomenon is known as gimbal lock. In Simscape Multibody, gimbal lock causes numerical errors that lead to simulation failure. The absence of singularities means that quaternions are more robust for simulation purposes.

To measure frame rotation in quaternion format, use:

- A Transform Sensor block when measuring rotation between two general frames. The **Rotation** menu of the property inspector contains a **Quaternion** parameter that you can select to expose the corresponding physical signal port (labeled **Q**).
- A joint block that has a spherical primitive when measuring the 3-D rotation between the two joint frames. The **Sensing** menu of the property inspector contains a **Position** parameter that you can select to expose the corresponding physical signal port (which is also labeled **Q**). For more information, see the Spherical Joint block reference page.

The quaternion output is a four-element row vector, $Q = (S\ V)$, where:

$$S = \cos\left(\frac{\theta}{2}\right)$$

and

$$V = [U_x\ U_y\ U_z]\sin\left(\frac{\theta}{2}\right)$$

θ is the angle of rotation and $[U_x, U_y, U_z]$ is the unit vector of the rotational axis. Note that for any given rotation, there are two quaternions. They are negatives of each other, but represent the same rotation. For example, the quaternions $[1\ \theta\ \theta\ \theta]$ and $[-1\ \theta\ \theta\ \theta]$ both represent the identity rotation.

Transform Measurements

The rotation transform is a 3×3 matrix that encodes frame rotation. In terms of base frame axes $[x, y, z]_B$, the follower frame axes $[x, y, z]_F$ are:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_B = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_F$$

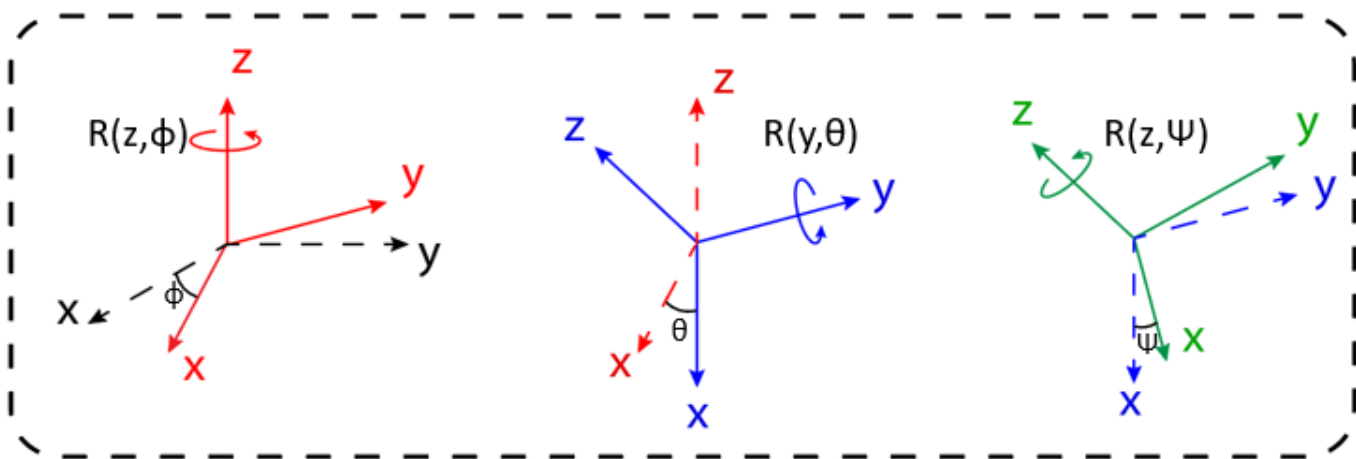
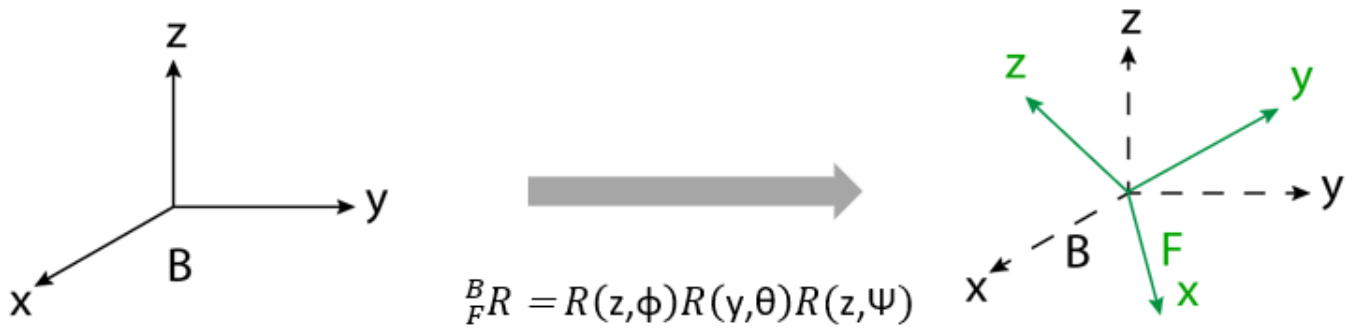
Each matrix column contains the coordinates of a follower frame axis resolved in the base frame. For example, the first column contains the coordinates of the follower frame X-axis, as resolved in the base frame. Similarly, the second and third columns contain the coordinates of the Y and Z-axes, respectively. Operating on a vector with the rotation matrix transforms the vector coordinates from the follower frame to the base frame.

You can sense frame rotation in terms of a rotation matrix using the Transform Sensor block. The property inspector for this block contains a **Transform** option that when selected exposes a physical signal port labeled **R**. Use this port to output the rotation matrix signal, for example, for processing and analysis in a Simulink subsystem—after converting the output physical signal to a Simulink signal through the PS-Simulink Converter block.

Rotation Sequence Measurements

You can use the Transform Sensor block to sense the rotation of the follower frame with respect to the base frame and represent the rotation as three successive elementary rotations.

The three elementary rotations are about the axes of an intermediate frame that rotates with each elementary rotation. In other words, the Transform Sensor block measures only intrinsic rotations. The block outputs a 3-by-1 vector that contains three angles in the range $[-\pi, \pi]$. You can output angles based on 12 different rotation sequences including X-Y-X, X-Y-Z, X-Z-X, X-Z-Y, Y-X-Y, Y-X-Z, Y-Z-X, Y-Z-Y, Z-X-Y, Z-X-Z, Z-Y-X, and Z-Y-Z. The image shows an example that represents the rotation of the follower frame with respect to the base frame by using rotations based on the Z-Y-Z sequence.



To compute extrinsic rotations, which are elementary rotations with respect to the base frame, you can manually convert the intrinsic rotations. Extrinsic rotations are equivalent to intrinsic rotations with the same angles, but with an inverted sequence order. For example, the extrinsic rotations Z-Y-X by angles ϕ , θ , and ψ are equivalent to the intrinsic rotations X-Y-Z by the angles ψ , θ , and ϕ .

In general, there are two sets of solutions for a rotation sequence whose rotation angles are in the range $(-\pi, \pi)$. If a rotation sequence has different letters, such as X-Y-Z and Z-X-Y, the Transform Sensor block outputs the solution whose θ is in the range $(-\pi/2, \pi/2)$. If the first and last letters of a rotation sequence are the same, such as X-Y-X and Z-Y-Z, the Transform Sensor block outputs the solution whose θ is in the range $(0, \pi)$.

For these cases, there are infinite number of solutions for the ϕ and ψ angles:

- A rotation sequence has different letters and the second rotation angle θ equals $-\pi/2$ or $\pi/2$.
- The first and last letters of a rotation sequence are the same and the second rotation angle θ equals 0 or π .

In these cases, the Transform Sensor block outputs only one set of solutions where the magnitudes of the ϕ and ψ angles are the same.

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specify Joint Actuation Torque” on page 3-84

More About

- “Motion Sensing” on page 3-56
- “Translational Measurements” on page 3-65
- “Selecting a Measurement Frame” on page 3-69

Translational Measurements

In this section...

“Translation Sensing Overview” on page 3-65
 “Measuring Translation” on page 3-65
 “Cartesian Measurements” on page 3-66
 “Cylindrical Measurements” on page 3-66
 “Spherical Measurements” on page 3-67

Translation Sensing Overview

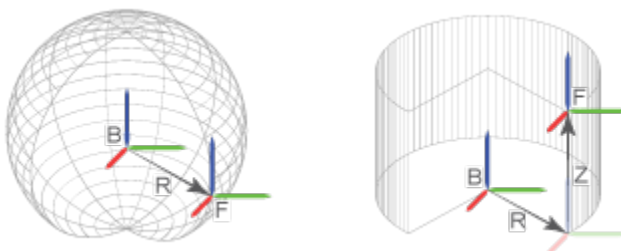
You can measure frame translation in different coordinate systems. These include Cartesian, cylindrical, and spherical systems. The different coordinate systems are available through the Transform Sensor block and, to a limited extent, through the Joint blocks. The choice of coordinate system depends on the model. Select the coordinate system that is most convenient for your application.

Measuring Translation

Translation is a relative quantity. The translation of one frame is meaningful only with respect to another frame. As such, blocks with translation sensing capability require two frames to make a measurement: measured and reference frames. In these blocks, the follower frame port identifies the measured frame; the base frame port identifies the reference frame of the measurement.

Some measurements are common to multiple coordinate systems. One example is the Z-coordinate, which exists in both Cartesian and cylindrical systems. In the Transform Sensor dialog box, coordinates that make up more than one coordinate system appear only once. Selecting **Z** outputs translation along the Z-axis in both Cartesian and cylindrical coordinate systems.

Other measurements are different but share the same name. For example, radius is a coordinate in both spherical and cylindrical systems. The spherical radius is different from the cylindrical radius: the former is the distance between two frame origins; the latter is the distance between one frame origin and a frame Z-axis.



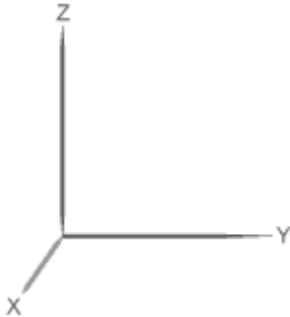
To differentiate between the two radial coordinates, Simscape Multibody uses the following convention:

- Radius — Cylindrical radial coordinate

- Distance — Spherical radial coordinate

Cartesian Measurements

The Cartesian coordinate system uses three linear coordinates—X, Y, and Z—corresponding to three mutually orthogonal axes. Cartesian translation measurements have units of distance, with meter being the default. You can use the PS-Simulink Converter block to select a different physical unit when interfacing with Simulink blocks.



Transform Sensor

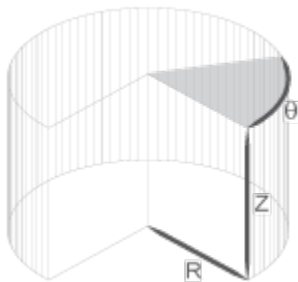
You can select any of the Cartesian axes in the Transform Sensor for translation sensing. This is true even if translation is constrained along any of the Cartesian axes. Selecting the Cartesian axes exposes physical signal ports x, y, and z, respectively.

Joints

With joint blocks, you can sense translation along each prismatic primitive axis. Selecting a sensing parameter from a prismatic primitive menu exposes the corresponding physical signal port. For example, if you select **Position** from the **Z Prismatic Primitive (Pz)** of a Cartesian Joint block, the block exposes physical signal port z.

Cylindrical Measurements

The cylindrical coordinate system uses one angular and two linear coordinates. The linear coordinates are the cylinder radius, R, and length, Z. The angular coordinate is the azimuth, ϕ , about the length axis. Linear coordinates have units of distance, with meter being the default. The angular coordinate has units of angle, with radian being the default. You can use the PS-Simulink Converter block to select a different physical unit when interfacing with Simulink blocks.



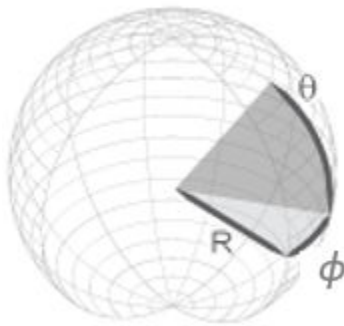
Transform Sensor

Only the Transform Sensor block can sense frame translation in cylindrical coordinates. In the dialog box of this block, you can select one or more cylindrical coordinates to measure. The cylindrical coordinates are named **Z**, **Radius**, and **Azimuth**. Selecting the cylindrical coordinates exposes physical signal ports z, rad, and azm, respectively.

Note **Z** belongs to both Cartesian and cylindrical systems.

Spherical Measurements

The spherical coordinate system uses two angular coordinates and one linear coordinate. The linear coordinate is the spherical radius, R . The angular coordinates are the azimuth, ϕ , and inclination, θ . The linear coordinate has units of distance, with meter being the default. The angular coordinates have units of angle, with radian being the default. You can use the PS-Simulink Converter block to select a different physical unit when interfacing with Simulink blocks.



Transform Sensor

Only the Transform Sensor block can sense frame translation in spherical coordinates. In the dialog box of this block, you can select one or more spherical coordinates to measure. The spherical coordinates are named **Azimuth**, **Distance**, and **Inclination**. Selecting the spherical coordinates exposes physical signal ports azm, dst, and inc, respectively.

Note **Azimuth** belongs to both cylindrical and spherical systems. **Distance** is the spherical radius.

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specify Joint Actuation Torque” on page 3-84

More About

- “Motion Sensing” on page 3-56
- “Rotational Measurements” on page 3-60

- “Selecting a Measurement Frame” on page 3-69

Selecting a Measurement Frame

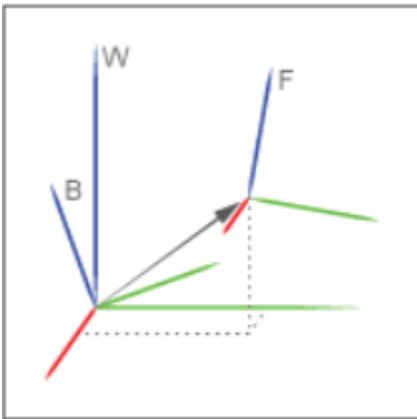
In this section...

“Measurement Frame” on page 3-69

“Example” on page 3-70

You can use the Transform Sensor block to measure the relative relationship between two arbitrary frames that are connected to the **B** and **F** frame ports of the block. The relationship includes relative rotation, translation, and their first and second time derivatives. These measurements are 3-D vectors or higher dimensioned quantities, such as rotation matrices.

To do computation with the measured vectors, the vectors must be resolved in coordinates. The setting of **Measurement Frame** parameter determines where to resolve the measured vectors; the vectors are resolved in the selected frame's coordinates. For example, in the figure, because **Measurement Frame** was set to `World`, the Transform Sensor block resolves a translation vector, shown as a black arrow, in the world frame's coordinates.



Note The rotation measurement of the Transform Sensor block is independent of the **Measurement Frame** parameter.

Measurement Frame

You can set **Measurement Frame** parameter to `World`, `Base`, `Follower`, `Non-Rotating Base`, or `Non-Rotating Follower`.

World

The Transform Sensor block resolves the measured vectors in the world frame's coordinates.

The world frame is an inertial frame.

Base or Follower

The Transform Sensor block resolves the measured vectors in the coordinates of the selected frame that is the base or follower frame.

The base or follower frame is the frame that connects to the block's **B** or **F** port, respectively. The base and follower frames are non-inertial. Therefore, the vectors resolved in the base or follower frame may involve centripetal and Coriolis terms.

Non-Rotating Base or Non-Rotating Base

The Transform Sensor block maps the vectors resolved in the world frame to the selected frame that is non-rotating base or non-rotating follower frame. In other words, the block calculates the rotation matrix from the world frame to the current base or follower frame then multiplies the matrix with the vectors resolved in the world frame.

The non-rotating base or non-rotating follower frame is an instantaneous frame that is coincident and aligned with the corresponding base or follower frame at the current time. The measurements resolved in the non-rotating frames do not involve centripetal and Coriolis terms.

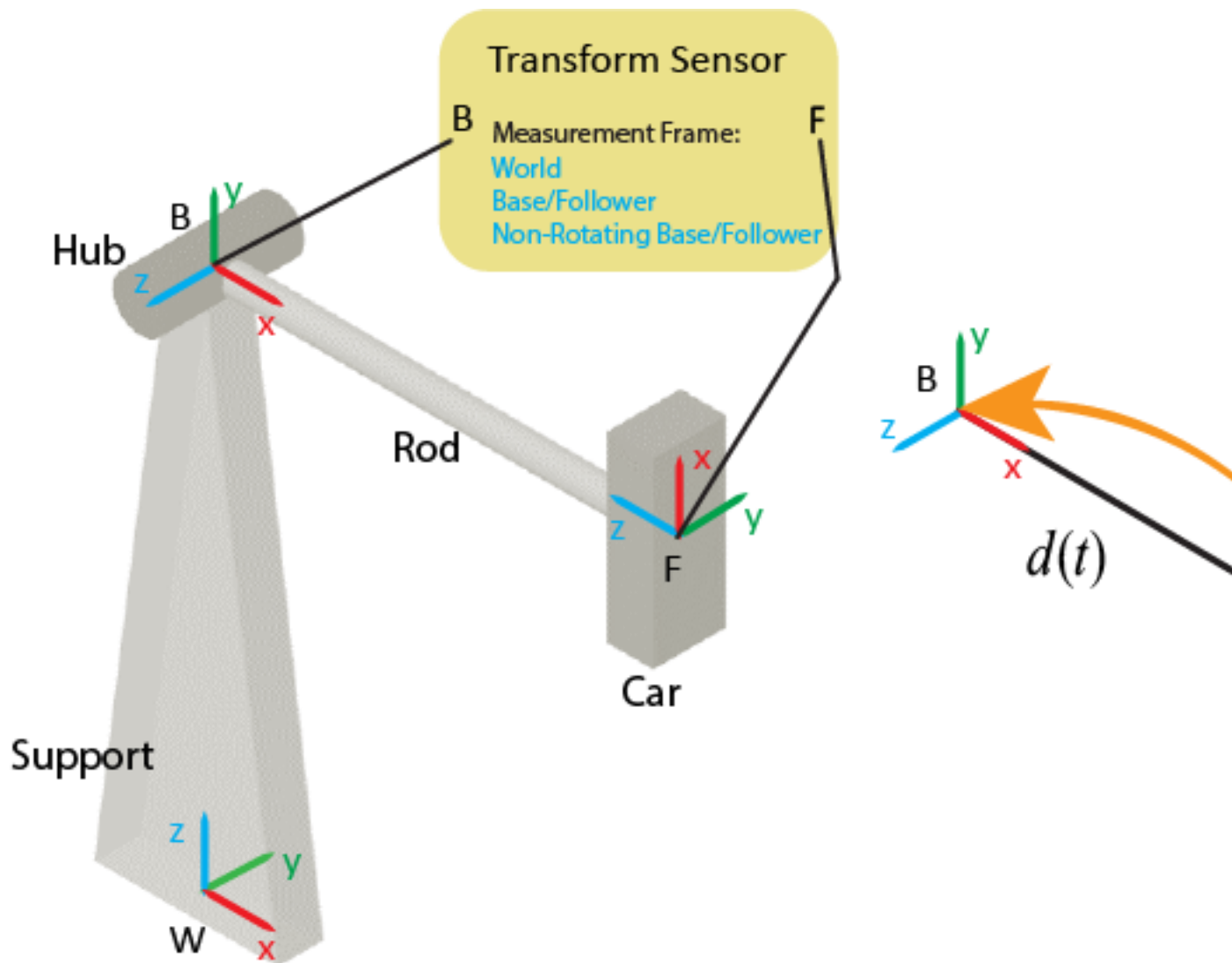
The table compares the properties of the measurements for the different **Measurement Frame** settings.

Measurement Frame	Standard Derivative Relationship
World	Yes
Base	Yes
Follower	Yes
Non-Rotating Base	No
Non-Rotating Follower	No

When a selected frame satisfies the standard derivative relationship, the measurements resolved in this frame are related to each other. For example, when you select **World**, the resolved linear acceleration vector is the time derivative of the resolved linear velocity vector, which is the time derivative of the resolved linear translation vector.

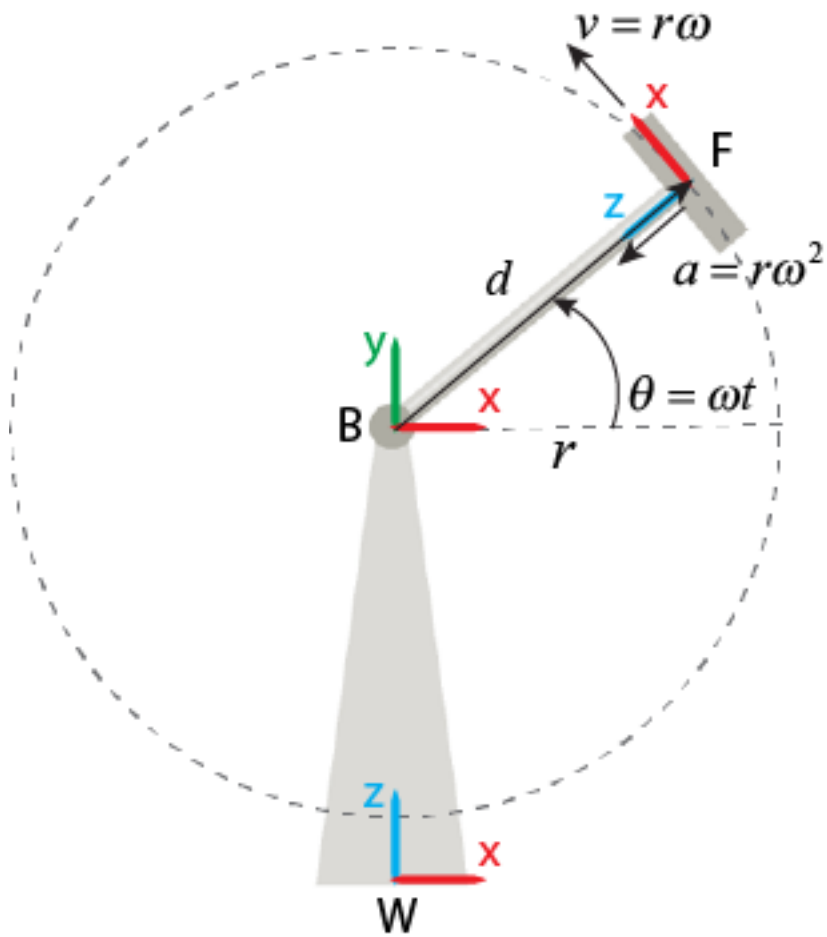
Example

This example shows the measurements of the Transform Sensor block with different settings of **Measurement Frame** parameter. The image illustrates a single degree-of-freedom system with four parts: a support, hub, rod, and car. The support is fixed on the ground, and the rod connects the hub and car. The base, follower, and world frames of the system are located at the center of the hub, car, and support's bottom, respectively.



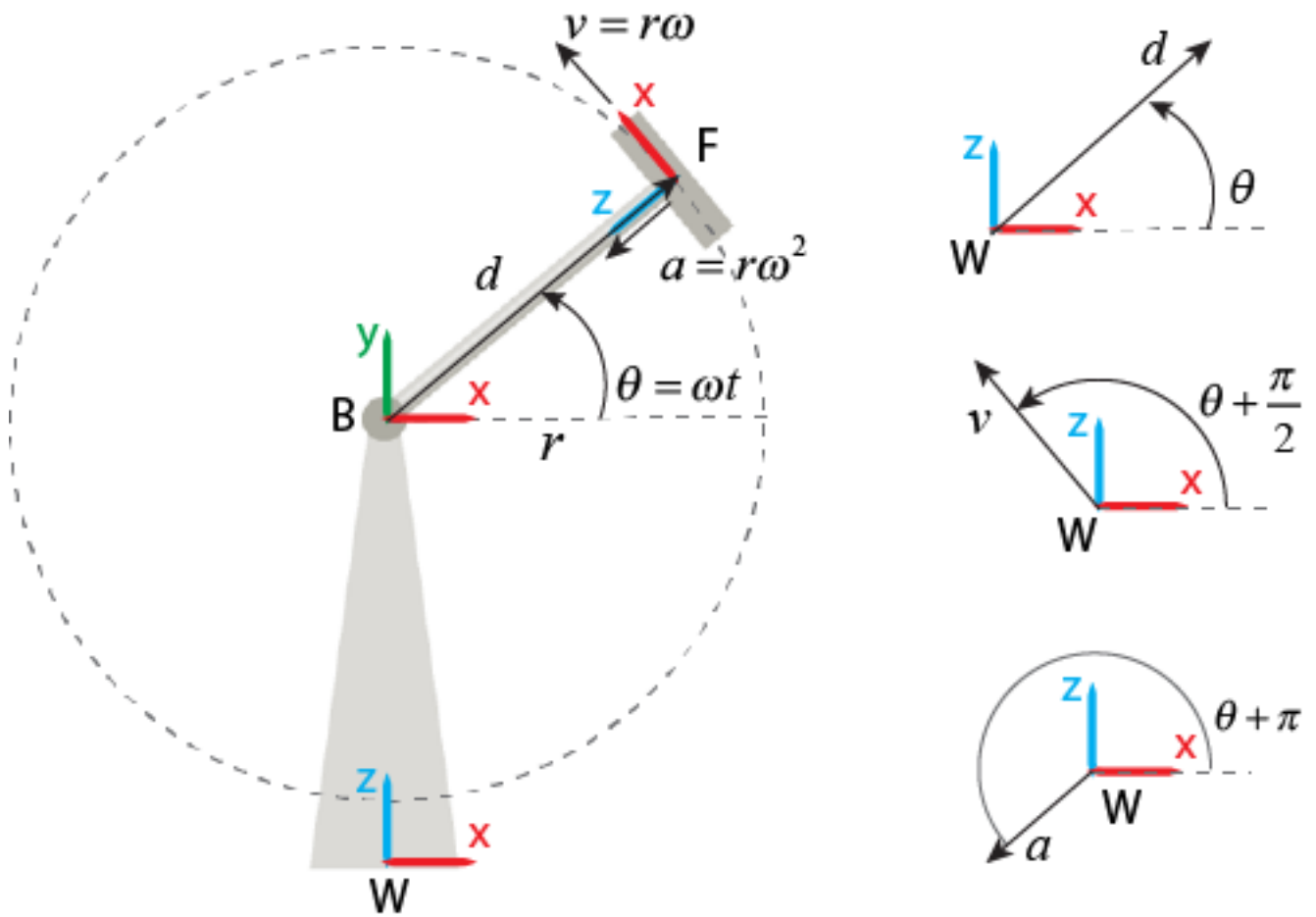
The rod has a length of r and rotates with a constant angular velocity, ω , around the Z-axis of the base frame. A Transform Sensor block is used to measure the relative motions between the car and hub. For example, the block measures the relative translation, $d(t)$ and rotation, $R_F^B(t)$ between the car and hub.

The image shows the front view of the system. For simplicity purposes, this example only shows how to resolve linear measurements, such as translation, velocity, and acceleration, in Cartesian coordinates.



World

When you set **Measurement Frame** to World, the block measures the motion of the follower frame with respect to the base frame then resolves the relative motion in the world frame.



The translation, velocity, and acceleration vectors have constant magnitudes because the length of the rod is constant. However, they rotate with a constant rotational velocity, ω , around the Y-axis of the world frame. Therefore, the translation, velocity, and acceleration vectors can be expressed as:

$$d_w(t) = r \begin{bmatrix} \cos\omega t \\ 0 \\ \sin\omega t \end{bmatrix}$$

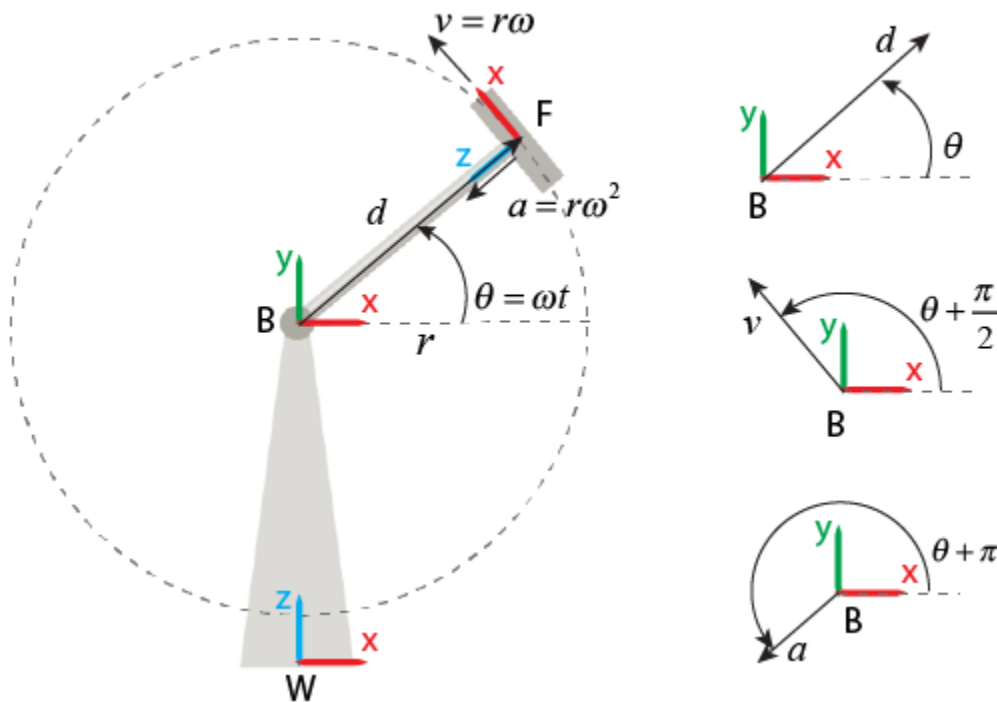
$$v_w(t) = r\omega \begin{bmatrix} -\sin\omega t \\ 0 \\ \cos\omega t \end{bmatrix}$$

$$a_w(t) = r\omega^2 \begin{bmatrix} -\cos\omega t \\ 0 \\ -\sin\omega t \end{bmatrix}$$

Note that the vectors resolved in the world frame always satisfy the standard derivative relationship. For example, a_w equals the time derivative of v_w .

Base or Follower

When you set **Measurement Frame** to Base, the block measures the relative motion of the follower frame with respect to the base frame, and resolves the measurements in the base frame's coordinates.



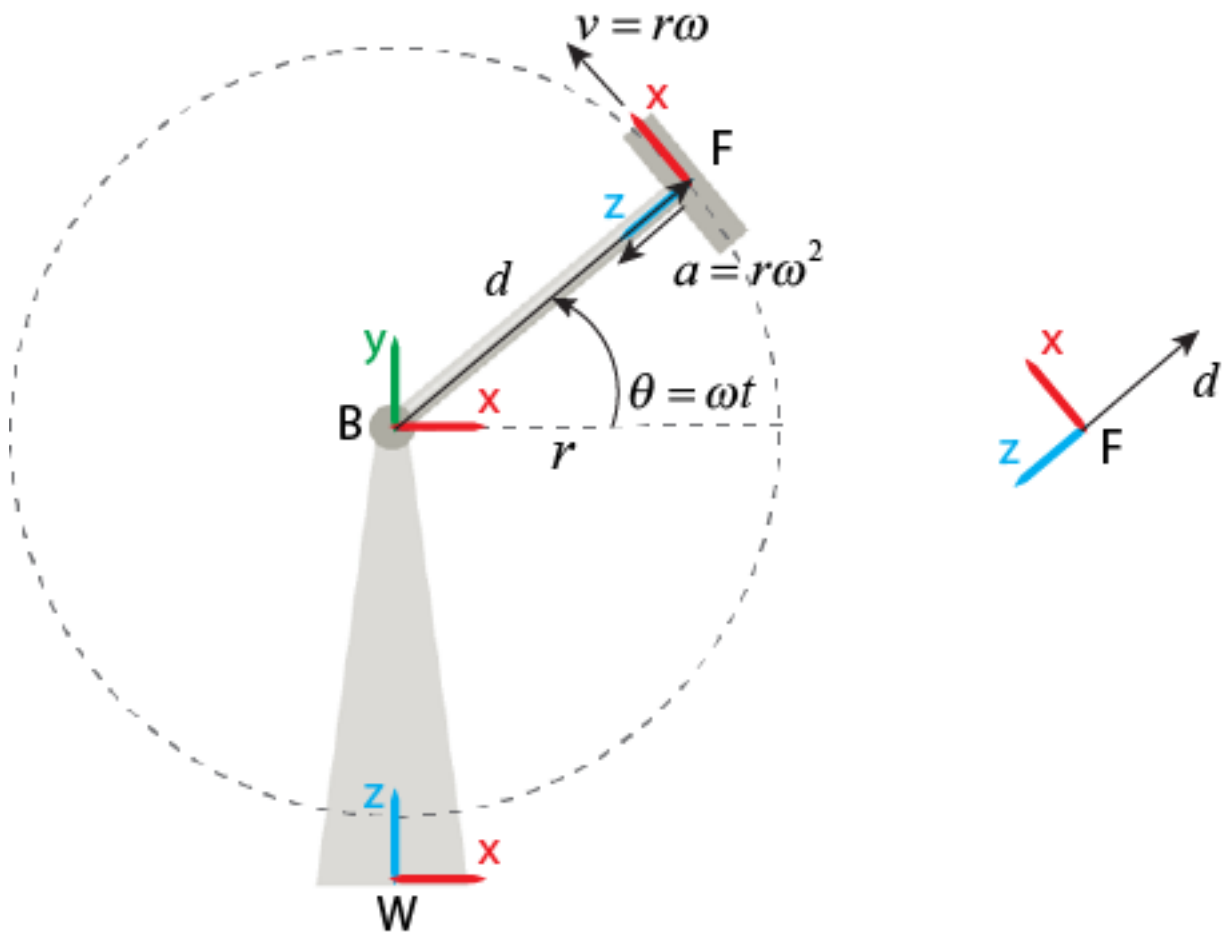
Because the base frame is fixed in this example, the measurements can be expressed as:

$$d_b(t) = r \begin{bmatrix} \cos\omega t \\ \sin\omega t \\ 0 \end{bmatrix}$$

$$v_b(t) = r\omega \begin{bmatrix} -\sin\omega t \\ \cos\omega t \\ 0 \end{bmatrix}$$

$$a_b(t) = r\omega^2 \begin{bmatrix} -\cos\omega t \\ -\sin\omega t \\ 0 \end{bmatrix}$$

When you set **Measurement Frame** to Follower, the block measures the relative motion of the follower frame to the base frame then resolves the measurements in the follower frame's coordinates. The resolved vectors include centripetal and Coriolis terms because the follower frame rotates over time. To an observer attached to the follower frame, the base frame's origin never moves. Therefore, linear velocity and linear acceleration are zero.



$$d_f(t) = r \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

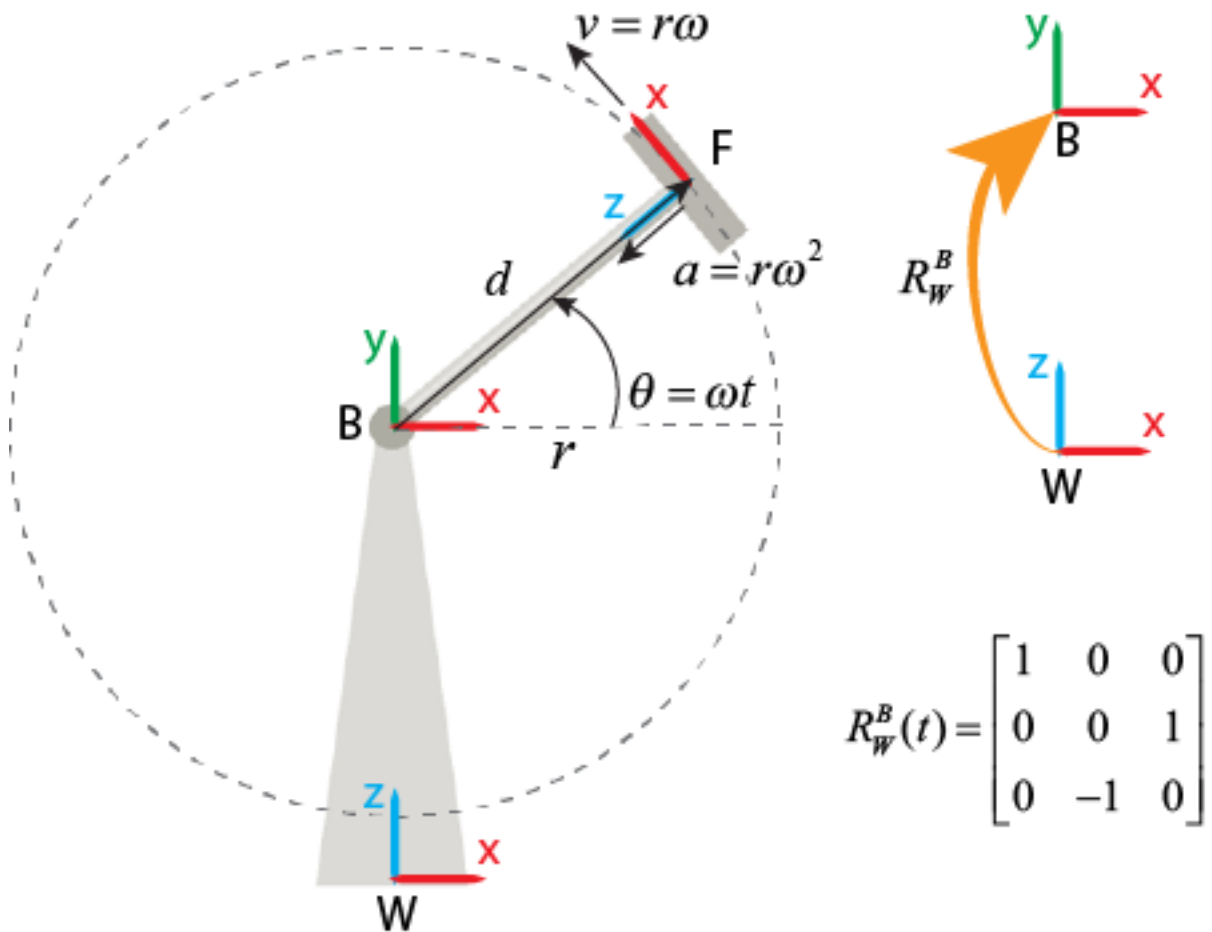
$$v_f(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$a_f(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Note that the vectors resolved in the base and follower frames always satisfy the standard derivative relationship. For example, v_b equals the time derivative of d_b .

Non-Rotating Base or Non-Rotating Follower

When you set **Measurement Frame** to Non-Rotating Base, the block maps the vectors resolved in the world frame to an instantaneous frame that is coincident and aligned with the base frame at the current moment.



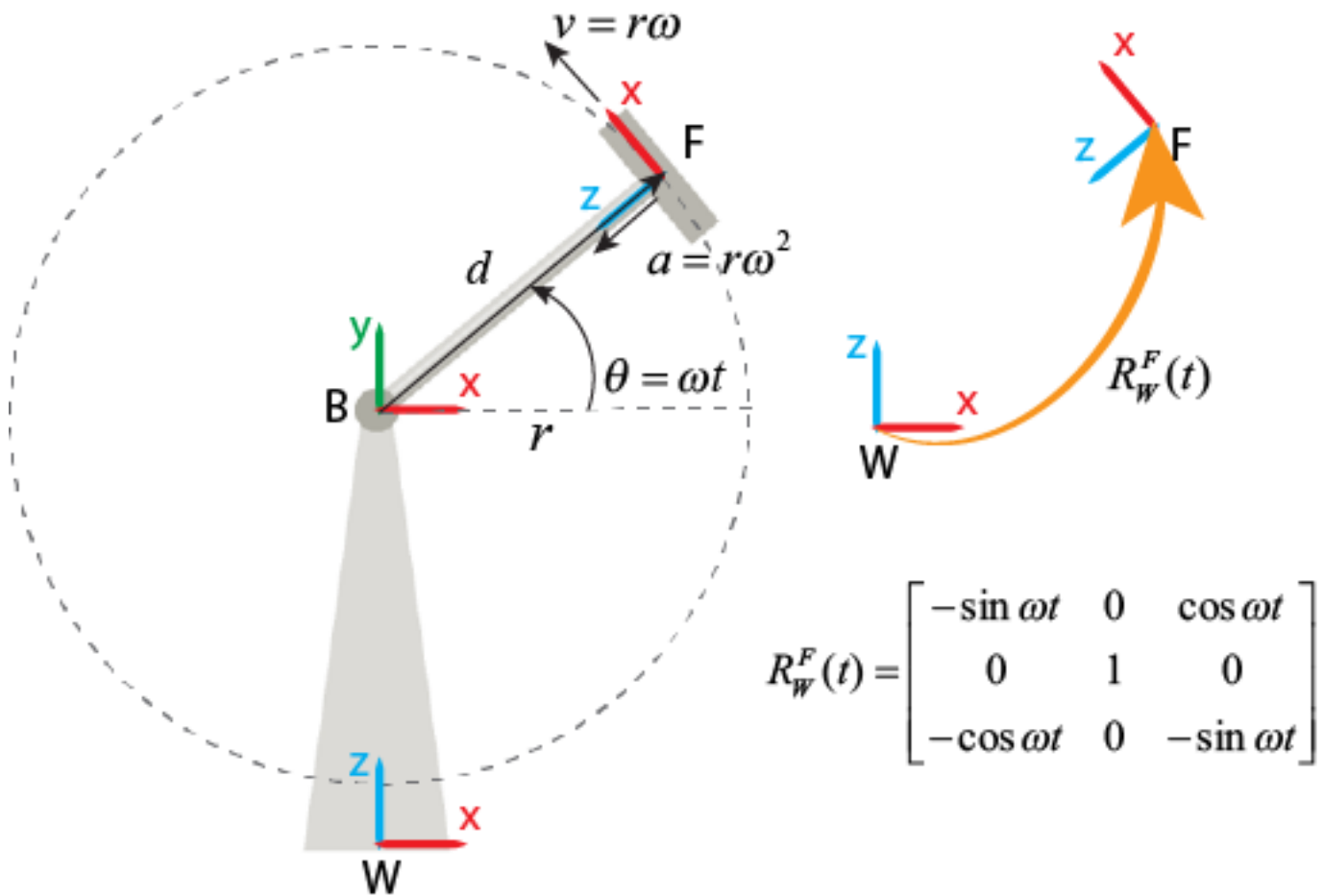
$$R_W^B(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$d_{nb}(t) = R_W^B * d_w(t) = r \begin{bmatrix} \cos\omega t \\ \sin\omega t \\ 0 \end{bmatrix}$$

$$v_{nb}(t) = R_W^B * v_w(t) = r\omega \begin{bmatrix} -\sin\omega t \\ \cos\omega t \\ 0 \end{bmatrix}$$

$$a_{nb}(t) = R_W^B * a_w(t) = r\omega^2 \begin{bmatrix} -\cos\omega t \\ -\sin\omega t \\ 0 \end{bmatrix}$$

When you set **Measurement Frame** to Non-Rotating Follower, the block maps the vectors resolved in the world frame to an instantaneous frame that is coincident and aligned with the follower frame at the current moment.



$$d_{nf}(t) = R_W^F * d_w(t) = r \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

$$v_{nf}(t) = R_W^F * v_w(t) = r\omega \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$a_{nf}(t) = R_W^F * a_w(t) = r\omega^2 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Note that if a base or follower frame is not fixed, the measurements in its corresponding non-rotating frame do not satisfy the standard derivative relationship. For example, because the follower frame rotates, if you set **Measurement Frame** to Non-Rotating Follower, the resolved velocity vector is not the time derivative of the resolved translation vector.

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specify Joint Actuation Torque” on page 3-84

More About

- “Motion Sensing” on page 3-56
- “Rotational Measurements” on page 3-60
- “Translational Measurements” on page 3-65

Sense Motion Using a Transform Sensor Block

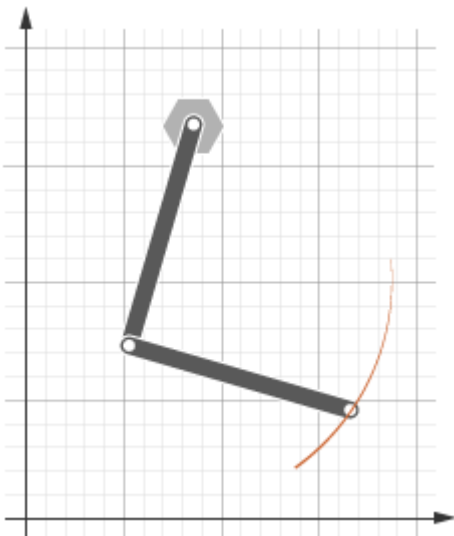
In this section...

“Model Overview” on page 3-79
 “Modeling Approach” on page 3-79
 “Build Model” on page 3-80
 “Guide Model Assembly” on page 3-81
 “Simulate Model” on page 3-81
 “Save Model” on page 3-83

Model Overview

The Transform Sensor block provides the broadest motion-sensing capability in Simscape Multibody models. Using this block, you can sense motion variables between any two frames in a model. These variables can include translational and rotational position, velocity, and acceleration.

In this example, you use a Transform Sensor block to sense the lower link translational position with respect to the World frame. You output the position coordinates directly to the model workspace, and then plot these coordinates using MATLAB commands. By varying the joint state targets, you can analyze the lower-link motion under quasi-periodic and chaotic conditions.



Modeling Approach

In this example, you rely on gravity to cause the double pendulum to move. You displace the links from equilibrium and then let gravity act on them. To displace the links at time zero, you use the **State Targets** section of the Revolute Joint block dialog box. You can specify position or velocity. When you are ready, you simulate the model to analyze its motion.

To sense motion, you use the Transform Sensor block. First, you connect the base and follower frame ports to the World Frame and lower link subsystem blocks. By connecting the ports to these blocks,

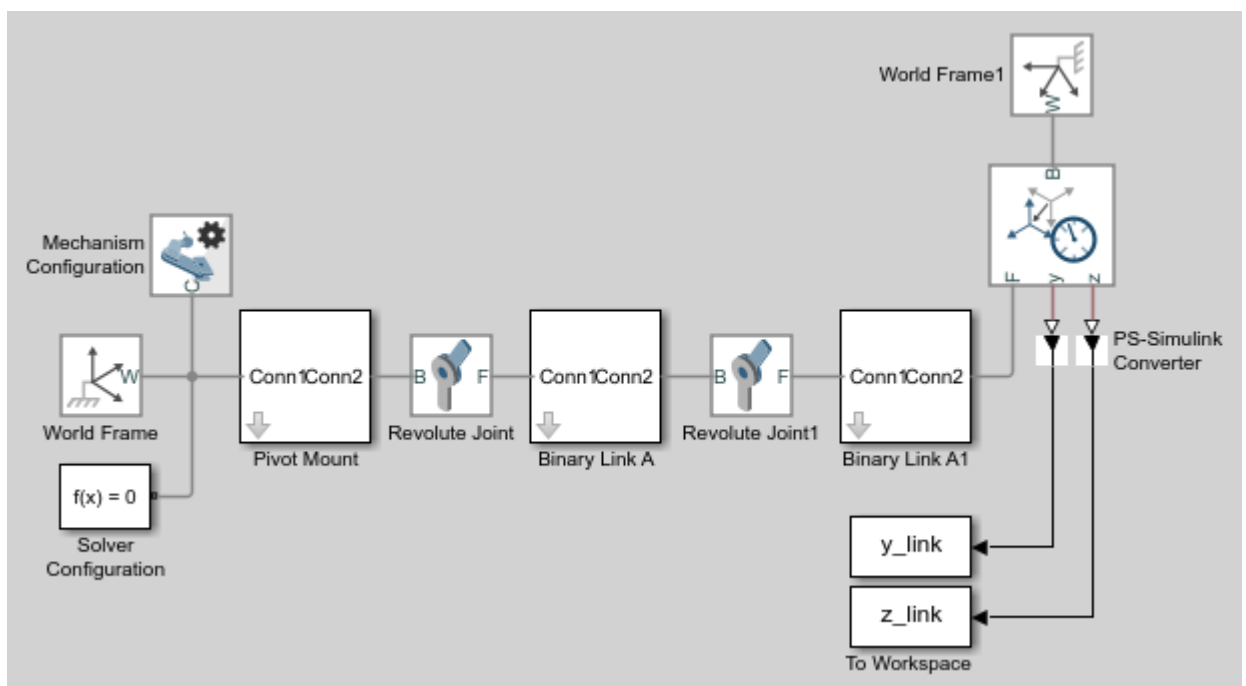
you can sense motion in the lower link with respect to the World frame. Then, you select the translation parameters to sense. By selecting **Y** and **Z**, you can sense translation along the Y and Z axes, respectively. You can plot these coordinates with respect to each other and analyze the motion that they reveal.

Build Model

- 1 At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double pendulum model opens up. For instructions on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-13.
- 2 Drag these blocks into the model to sense motion.

Library	Block	Quantity
Simscape > Multibody > Frames and Transforms	Transform Sensor	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Utilities	PS-Simulink Converter	2
Simulink > Sinks	To Workspace	2

- 3 In the Transform Sensor block dialog box, select **Translation > Y** and **Translation > Z**. The block exposes two physical signal output ports, labeled y and z.
- 4 In the PS-Simulink Converter blocks, specify units of cm.
- 5 In the To Workspace blocks, enter the variable names `y_link` and `z_link`.
- 6 Connect the blocks to the model as shown in the figure. You must connect the base frame port of the Transform Sensor block to the World Frame block. The new blocks are shaded gray.



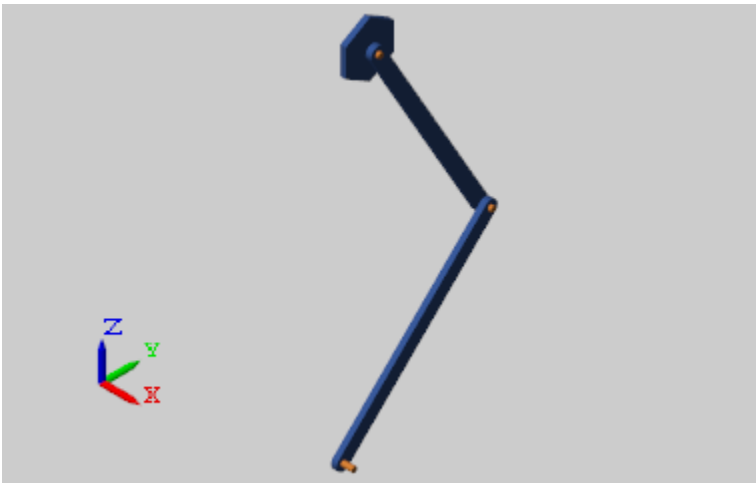
Guide Model Assembly

Specify the initial state of each joint. Later, you can modify this state to explore different motion types. For the first iteration, rotate only the top link by a small angle.

- 1 In the Revolute Joint block dialog box, select **State Targets > Specify Position Target**.
- 2 Set **Value** to 10 deg.
- 3 In the Revolute Joint1 block dialog box, check that **State Targets > Specify Position Target** is cleared.

Simulate Model

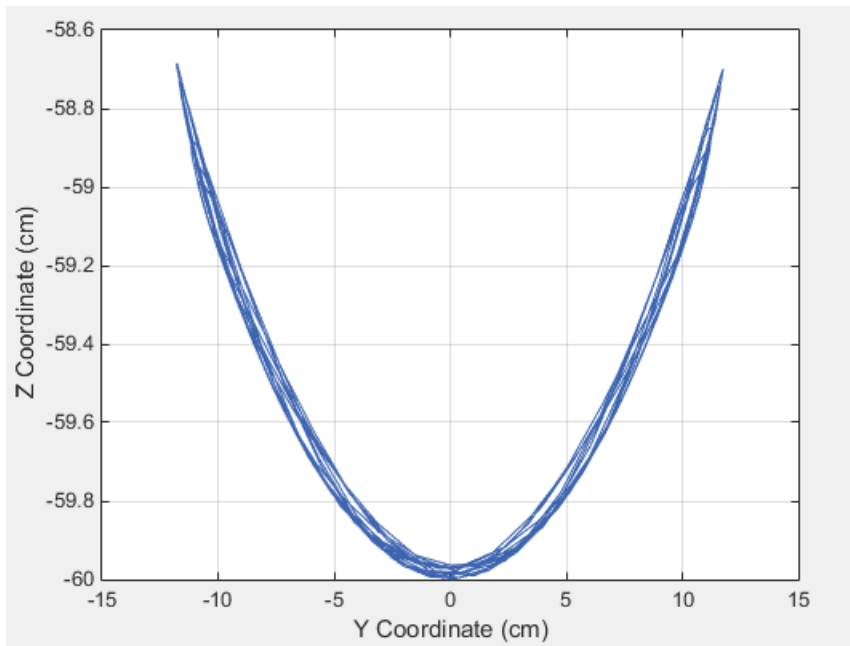
Run the simulation. Mechanics Explorer plays a physics-based animation of the double pendulum assembly.



You can now plot the position coordinates of the lower link. To do this, at the MATLAB command line, enter:

```
figure;
plot(y_link.data, z_link.data, 'color', [60 100 175]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
grid on;
```

The figure shows the plot that opens. This plot shows that the lower link path varies only slightly with each oscillation. This behavior is characteristic of quasi-periodic systems.



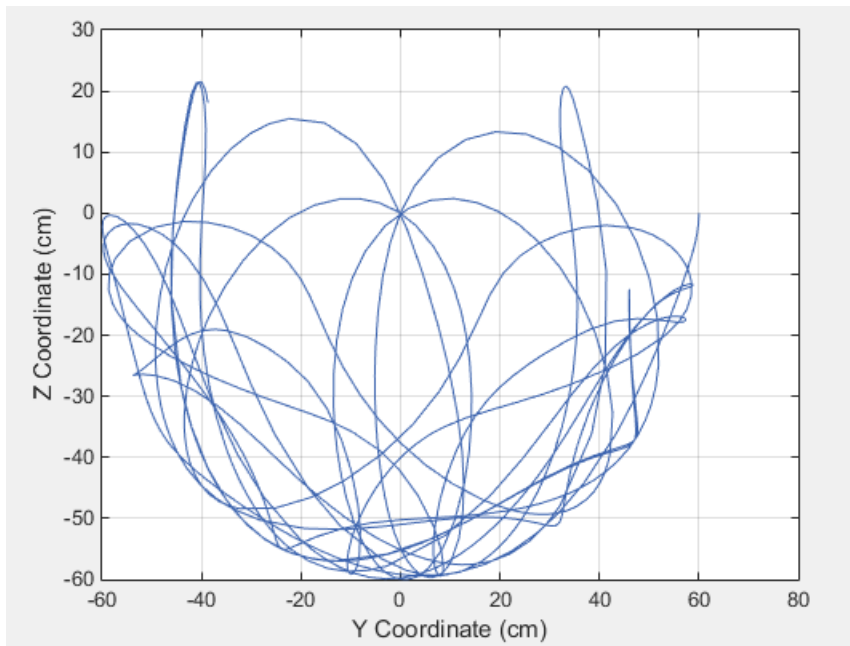
Simulate Chaotic Motion

By adjusting the revolute joint state targets, you can simulate the model under chaotic conditions. One way to obtain chaotic motion is to rotate the top revolute joint by a large angle. To do this, in the Revolute Joint dialog box, change **State Targets > Position > Value** to 90 and click **OK**.

Simulate the model with the new joint state target. To plot the position coordinates of the lower pendulum link with respect to the world frame, at the MATLAB command prompt, enter this code:

```
figure;  
plot(y_link.data, z_link.data, 'color', [60 100 175]/255);  
xlabel('Y Coordinate (cm)');  
ylabel('Z Coordinate (cm)');  
grid on;
```

The figure shows the plot that opens.



Save Model

Save the model in a convenient folder under the name `double_pendulum_sensing`. You reuse this model in a subsequent tutorial, "Specify Joint Motion in Planar Manipulator Model" on page 3-114.

Specify Joint Actuation Torque

In this section...

“Model Overview” on page 3-84

“Four-Bar Linkages” on page 3-84

“Modeling Approach” on page 3-86

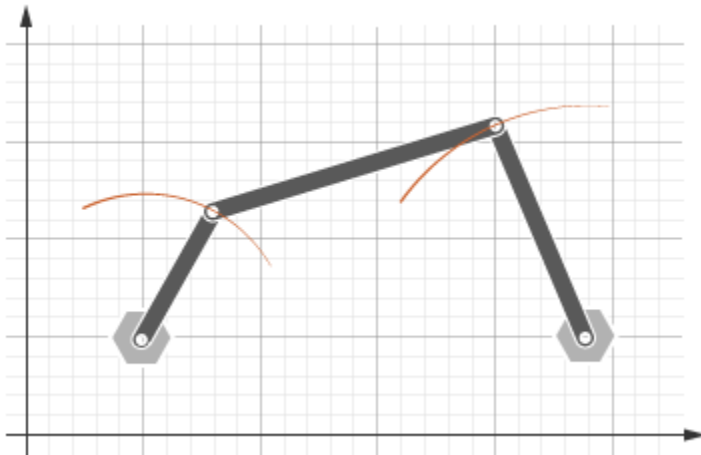
“Build Model” on page 3-87

“Simulate Model” on page 3-90

Model Overview

In Simscape Multibody, you actuate a joint directly using the joint block. Depending on the application, the joint actuation inputs can include force/torque or motion variables. In this example, you prescribe the actuation torque for a revolute joint in a four-bar linkage model.

Transform Sensor blocks add motion sensing to the model. You can plot the sensed variables and use the plots for kinematic analysis. In this example, you plot the coupler curves of three four-bar linkage types: crank-rocker, double-crank, and double-rocker.



Four-Bar Linkages

The four-bar linkage contains four links that interconnect with four revolute joints to form a planar closed loop. This linkage converts the motion of an input link into the motion of an output link. Depending on the relative lengths of the four links, a four-bar linkage can convert rotation into rotation, rotation into oscillation, or oscillation into oscillation.

Links

Links go by different names according to their functions in the four-bar linkage. For example, coupler links transmit motion between crank and rocker links. The table summarizes the different link types that you may find in a four-bar linkage.

Link	Motion
Crank	Revolves with respect to the ground link
Rocker	Oscillates with respect to the ground link
Coupler	Transmits motion between crank and rocker links
Ground	Rigidly connects the four-bar linkage to the world or another subsystem

It is common for links to have complex shapes. This is especially true of the ground link, which may be simply the fixture holding the two pivot mounts that connect to the crank or rocker links. You can identify links with complex shapes as the rigid span between two adjacent revolute joints. In example “Model a Closed-Loop Kinematic Chain” on page 2-16, the rigid span between the two pivot mounts represents the ground link.

Linkages

The type of motion conversion that a four-bar linkage provides depends on the types of links that it contains. For example, a four-bar linkage that contains two crank links converts rotation at the input link into rotation at the output link. This type of linkage is known as a double-crank linkage. Other link combinations provide different types of motion conversion. The table describes the different types of four-bar linkages that you can model.

Linkage	Input-Output Motion
Crank-rocker	Continuous rotation-oscillation (and vice-versa)
Double-Crank	Continuous rotation-continuous rotation
Double-rocker	Oscillation-oscillation

Grashof Condition

The Grashof theorem provides the basic condition that the four-bar linkage must satisfy so that at least one link completes a full revolution. According to this theorem, a four-bar linkage contains one or more crank links if the combined length of the shortest and longest links does not exceed the combined length of the two remaining links. Mathematically, the Grashof condition is:

$$s+l \leq p+q \quad (3-1)$$

where:

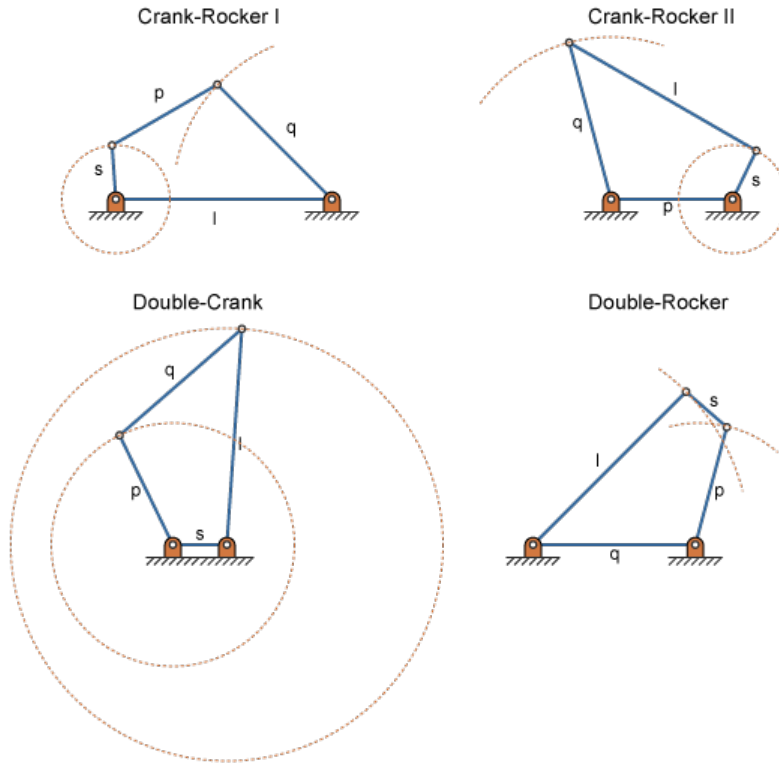
- s is the shortest link
- l is the longest link
- p and q are the two remaining links

Grashof Linkages

A Grashof linkage can be of three different types:

- Crank-rocker
- Double-crank
- Double-rocker

By changing the ground link, you can change the Grashof linkage type. For example, by assigning the crank link of a crank-rocker linkage as the ground link, you obtain a double-crank linkage. The figure shows the four linkages that you obtain by changing the ground link.



Modeling Approach

In this example, you perform two tasks. First you add a torque actuation input to the model. Then, you sense the motion of the crank and rocker links with respect to the World frame. The actuation input is a torque that you apply to the joint connecting the base to the crank link. Because you apply the torque at the joint, you can add this torque directly through the joint block. The block that you add the actuation input to is called Base-Crank Revolute Joint.

You add the actuation input to the joint block through a physical signal input port. This port is hidden by default. To display it, you must select **Provided by Input** from the **Actuation > Torque** drop-down list.

You can then specify the torque value using either Simscape or Simulink blocks. If you use Simulink blocks, you must use the Simulink-PS Converter block. This block converts the Simulink signal into a physical signal that Simscape Multibody can use. For more information, see “Actuating and Sensing with Physical Signals” on page 3-28.

To sense crank and rocker link motion, you use the Transform Sensor block. With this block, you can sense motion between any two frames in a model. In this example, you use it to sense the [Y Z] coordinates of the crank and rocker links with respect to the World frame.

The physical signal output ports of the Transform Sensor blocks are hidden by default. To display them, you must select the appropriate motion outputs. Using the PS-Simulink Converter, you can

convert the physical signal outputs into Simulink signals. You can then connect the resulting Simulink signals to other Simulink blocks.

In this example, you output the crank and rocker link coordinates to the workspace using Simulink To Workspace blocks. The output from these blocks provide the basis for phase plots showing the different link paths.

Build Model

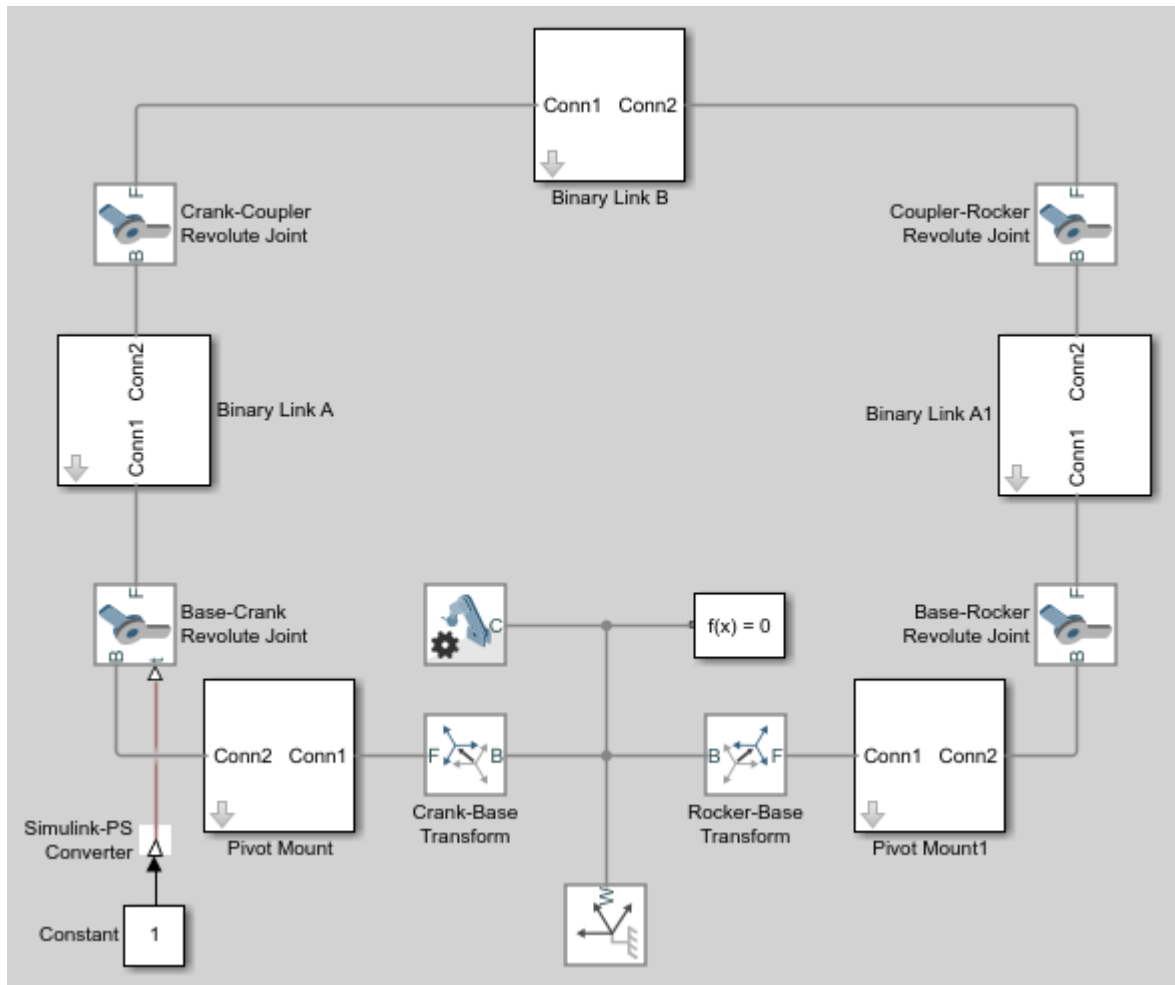
Provide the joint actuation input, specify the joint internal mechanics, and sense the position coordinates of the coupler link end frames.

Provide Joint Actuation Input

- 1 At the MATLAB command prompt, enter `smdoc_four_bar`. A four bar model opens up. For instructions on how to create this model, see “Model a Closed-Loop Kinematic Chain” on page 2-16.
- 2 In the Base-Crank Revolute Joint block dialog box, in the **Actuation > Torque** drop-down list, select **Provided by Input**. The block exposes a physical signal input port, labeled `t`.
- 3 Drag these blocks into the model. The blocks enable you to specify the actuation torque signal.

Library	Block
Simulink > Sources	Constant
Simscape > Utilities	Simulink-PS Converter

- 4 Connect the blocks as shown in the figure. The new blocks are shaded gray.



Specify Joint Internal Mechanics

Real joints dissipate energy due to damping. You can specify joint damping directly in the block dialog boxes. In each Revolute Joint block dialog box, under **Internal Mechanics > Damping Coefficient**, enter $5e-4$ and press **OK**.

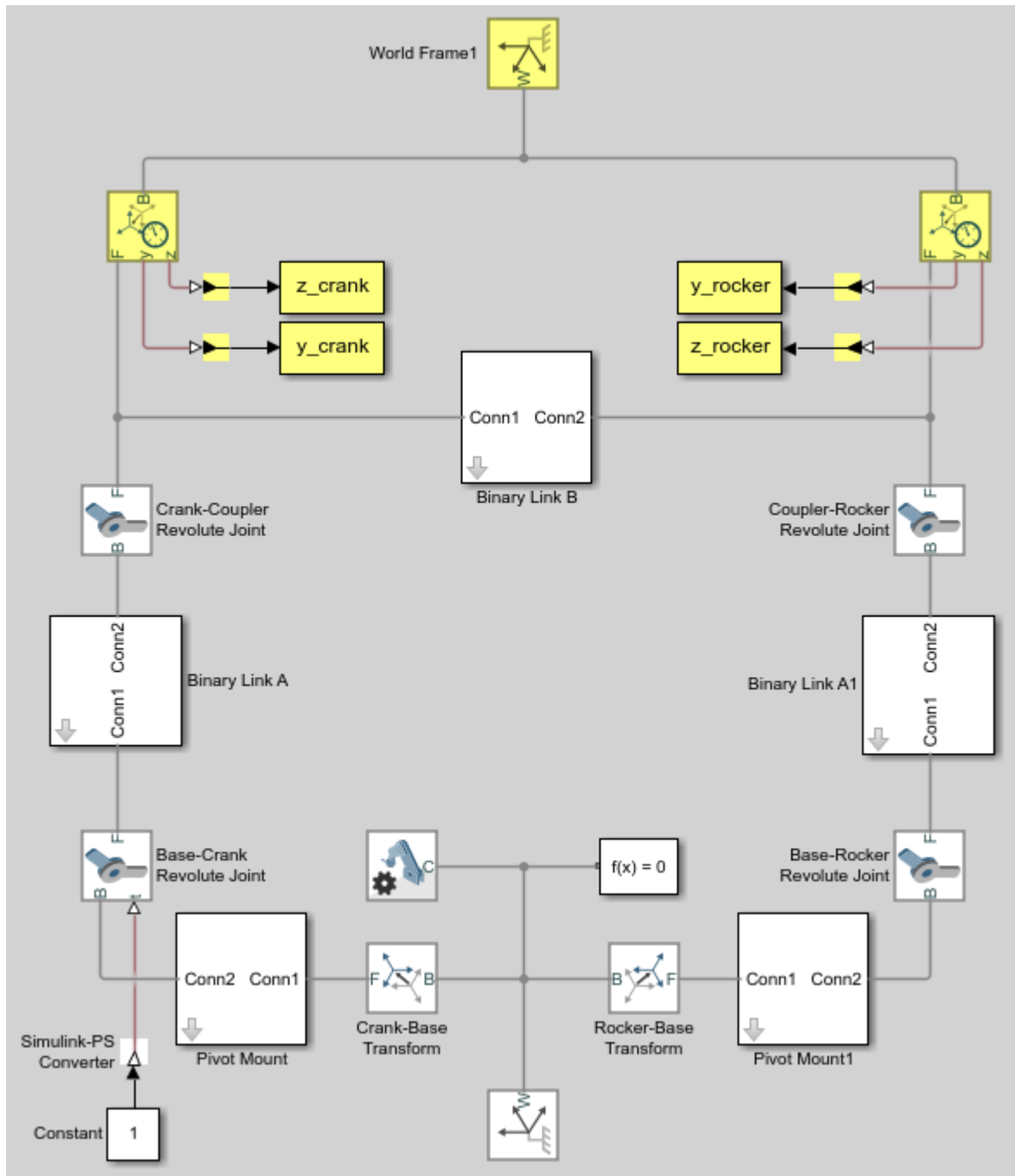
Sense Link Position Coordinates

- 1 Add these blocks to the model. The blocks enable you to sense frame position during simulation.

Library	Block	Quantity
Simscape > Multibody > Frames and Transforms	Transform Sensor	2
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Utilities	PS-Simulink Converter	4
Simulink > Sinks	To Workspace	4

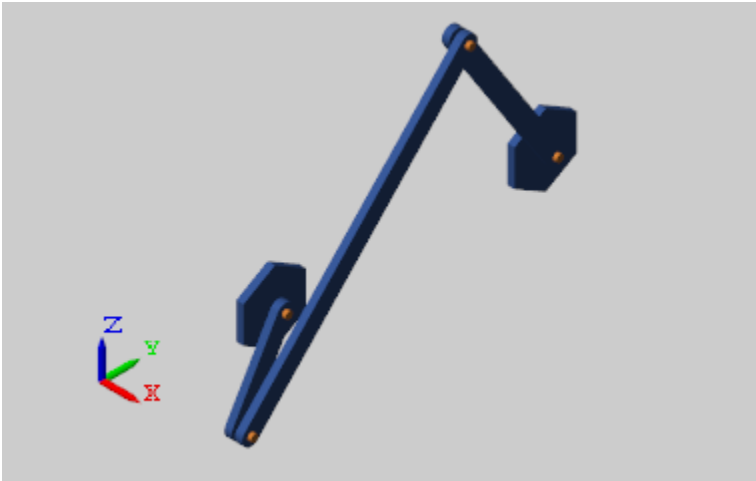
- 2 In the Transform Sensor block dialog boxes, select **Translation > Y** and **Translation > Z**. Resize the block as needed.

- 3** In the **Output signal unit** parameters of the PS-Simulink Converter block dialog boxes, enter cm.
- 4** In the **Variable Name** parameters of the To Workspace block dialog boxes, enter the variable names:
 - y_crank
 - z_crank
 - y_rocker
 - z_rocker
- 5** Connect and name the blocks as shown in the figure, rotating them as needed. Ensure that the To Workspace blocks with the z_crank and z_rocker variable names connect to the z frame ports of the Transform Sensor blocks. The new blocks are shaded yellow.



Simulate Model

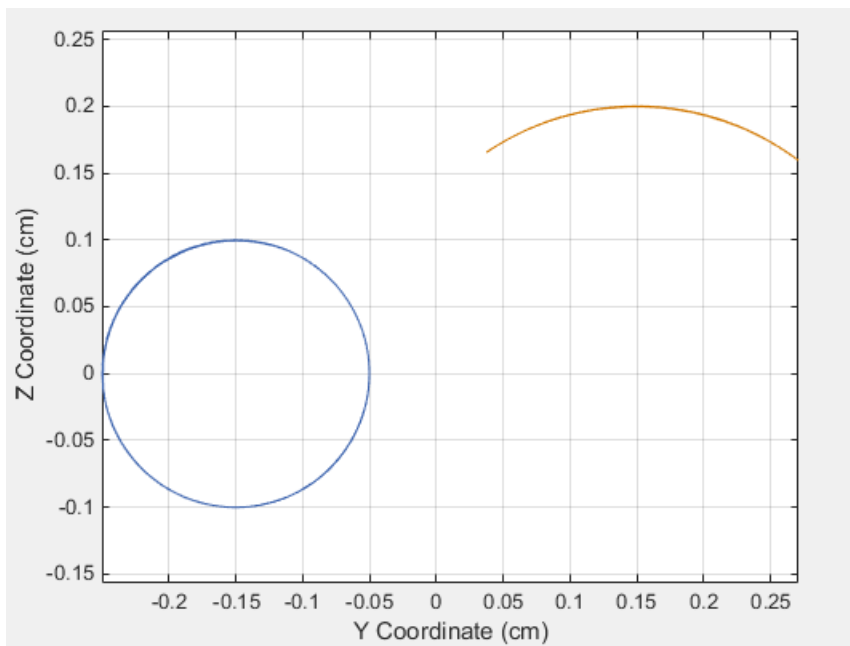
Run the simulation. You can do this in the Simulink tool bar by clicking the run button. Mechanics Explorer plays a physics-based animation of the four bar assembly.



Once the simulation ends, you can plot the position coordinates of the coupler link end frames, e.g., by entering the following code at the MATLAB command line:

```
figure;
plot(y_crank.data, z_crank.data, 'color', [60 100 175]/255);
hold;
plot(y_rocker.data, z_rocker.data, 'color', [210 120 0]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
axis equal; grid on;
```

The figure shows the plot that opens. This plot shows that the crank completes a full revolution, while the rocker completes a partial revolution, e.g., it oscillates. This behavior is characteristic of crank-rocker systems.

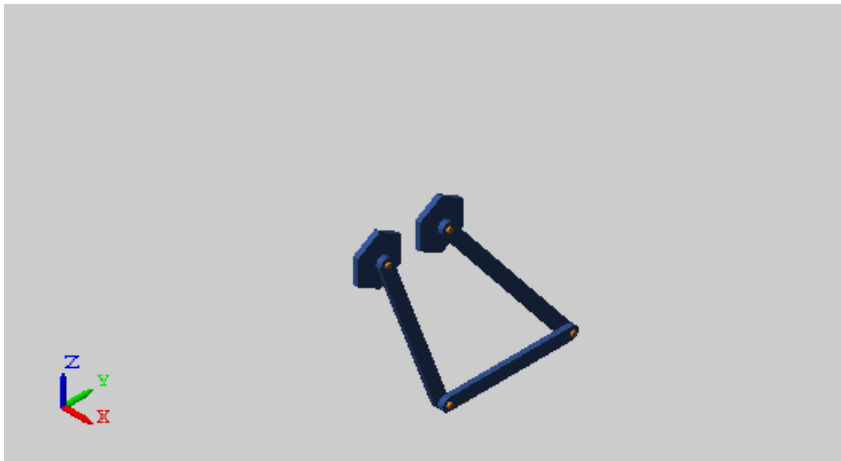


Simulate Model in Double-Crank Mode

Try simulating the model in double-crank mode. You can change the four-bar linkage into a double-crank linkage by changing the binary link lengths according to the table.

Block	Parameter	Value
Binary Link A	Length	25
Binary Link B	Length	20
Binary Link A1	Length	30
Crank-Base Transform	Translation > Offset	5
Rocker-Base Transform	Translation > Offset	5

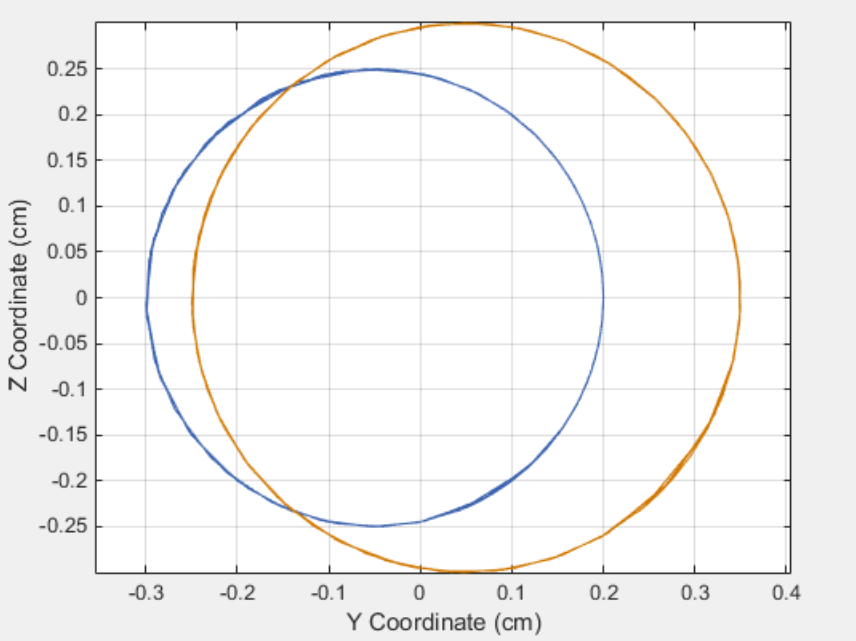
Update and simulate the model. The figure shows the updated visualization display in Mechanics Explorer.



Plot the position coordinates of the coupler link end frames. At the MATLAB command line, enter:

```
figure;
plot(y_crank.data, z_crank.data, 'color', [60 100 175]/255);
hold;
plot(y_rocker.data, z_rocker.data, 'color', [210 120 0]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
axis equal; grid on;
```

The figure shows the plot that opens. This plot shows that both links complete a full revolution. This behavior is characteristic of double-crank linkages.



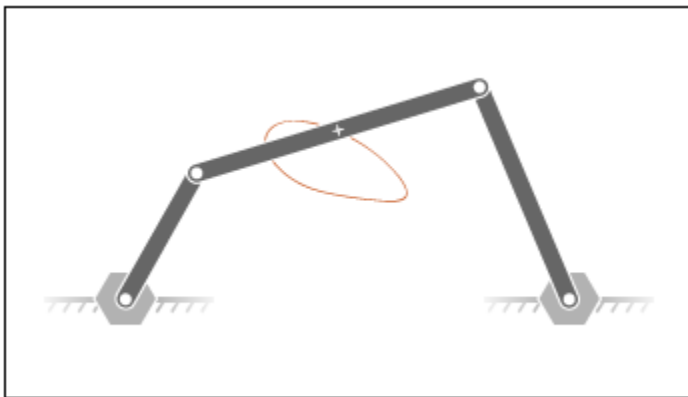
Analyze Motion at Various Parameter Values

In this section...

- “Model Overview” on page 3-94
- “Build Model” on page 3-94
- “Specify Block Parameters” on page 3-96
- “Create Simulation Script” on page 3-97
- “Run Simulation Script” on page 3-97

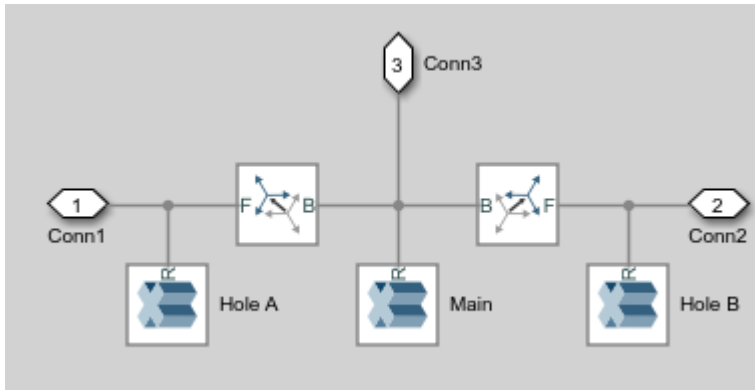
Model Overview

In this tutorial, you create a simple MATLAB script to simulate a four-bar model at various coupler lengths. The script uses the coupler motion coordinates, obtained using a Transform Sensor block, to plot the resulting coupler curve at each value of the coupler length. For information on how to create the four-bar model used in this tutorial, see “Model a Closed-Loop Kinematic Chain” on page 2-16.



Build Model

- 1 At the MATLAB command prompt, enter `smdoc_four_bar`. A four-bar model opens up. For instructions on how to create this model, see “Model a Closed-Loop Kinematic Chain” on page 2-16.
- 2 Under the mask of the Binary Link B block, connect a third Outport block as shown in the figure. You can add an Outport block by copying and pasting Conn1 or Conn2. The new block identifies the frame whose trajectory you plot in this tutorial.



- 3 Add the following blocks to the model. During simulation, the Transform Sensor block computes and outputs the coupler trajectory with respect to the world frame.

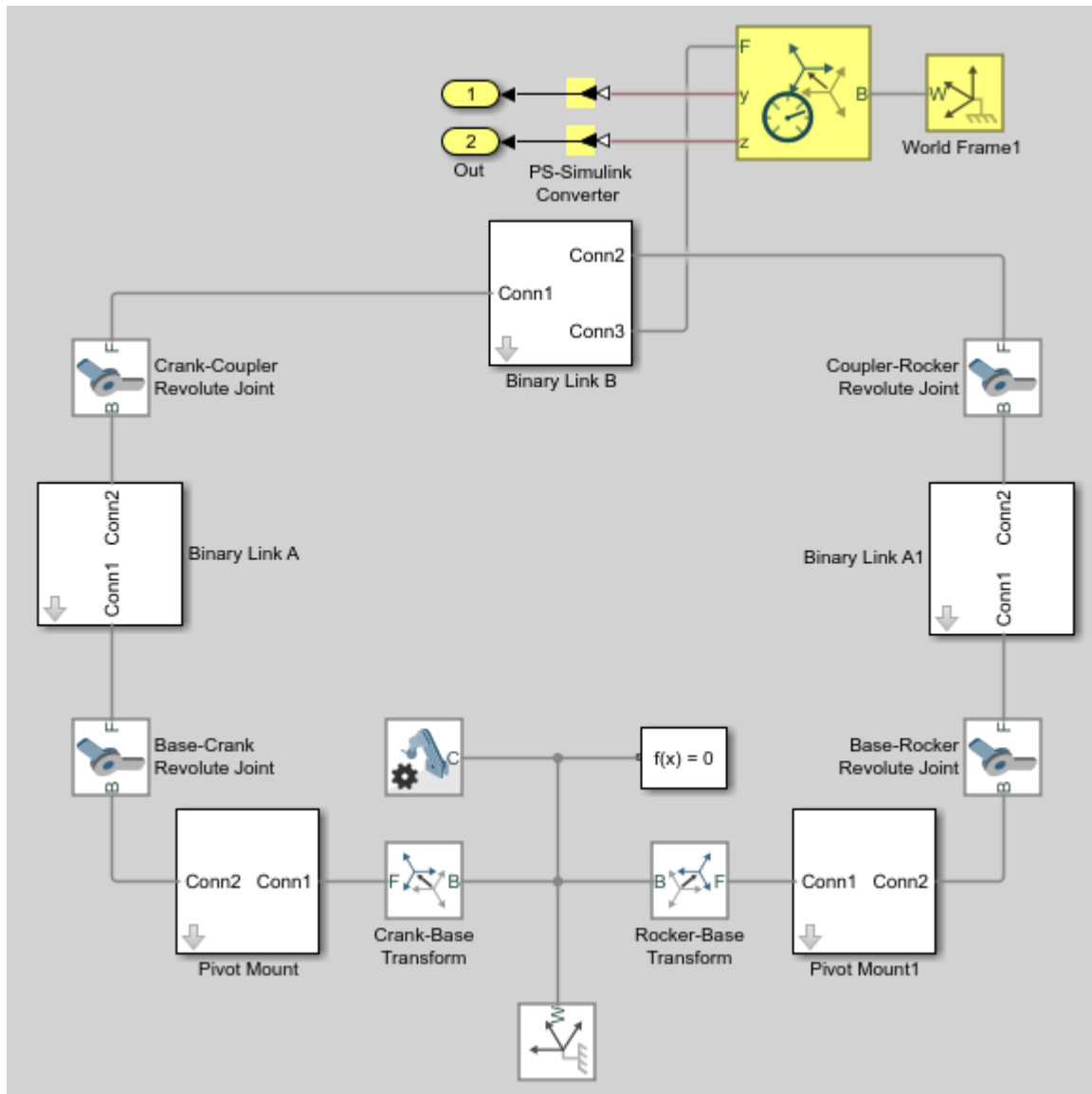
Library	Block	Quantity
Frames and Transforms	World Frame	1
Frames and Transforms	Transform Sensor	1
Simscape Utilities	PS-Simulink Converter	2
Simulink Sinks	Outport	2

- 4 In the Transform Sensor block dialog box, select these variables:

- **Translation > Y**
- **Translation > Z**

The block exposes frame ports y and z, through which it outputs the coupler trajectory coordinates.

- 5 Connect the blocks as shown in the figure. Be sure to flip the Transform Sensor block so that its base frame port, labeled B, connects to the World Frame block.



Specify Block Parameters

- 1 In the Mechanism Configuration block, change **Uniform Gravity** to None.
- 2 In the Base-Crank Revolute Joint block, specify the following velocity state targets. The targets provide an adequate source of motion for the purposes of this tutorial.
 - Select **State Targets > Specify Velocity**.
 - In **State Targets > Specify Velocity > Value**, enter 2 rev/s.
 - Deselect **State Target > Specify Position**.
- 3 Specify the following link lengths. The coupler link length is parameterized in terms of a MATLAB variable, `LCoupler`, enabling you change its value iteratively using a simple MATLAB script.

Block	Parameter	Value
Binary Link B	Length	LCoupler
Binary Link A1	Length	25

- 4 Save the model in a convenient folder, naming it `smdoc_four_bar_msensing`.

Create Simulation Script

Create a MATLAB script to iteratively run simulation at various coupler link lengths:

- 1 On the MATLAB toolstrip, click **New Script**.
- 2 In the script, enter the following code:

```
% Run simulation nine times, each time
% increasing coupler length by 1 cm.
% The original coupler length is 20 cm.
for i = (0:8);
    LCoupler = 20+i;

    % Simulate model at the current coupler link length (LCoupler),
    % saving the Output block data into variables y and z.
    [~, ~, y, z] = sim('smdoc_four_bar_msensing');

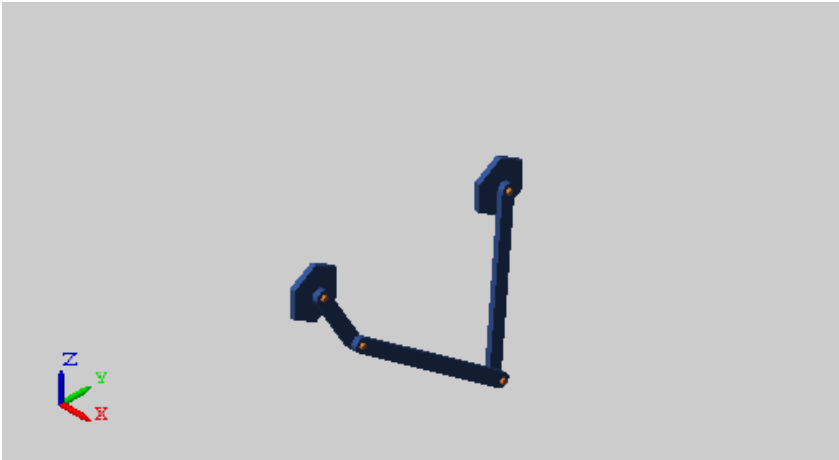
    % Plot the [y, z] coordinates of each coupler curve
    % on the x = i plane. i corresponds to the simulation run number.
    x = zeros(size(y)) + i;
    plot3(x, y, z, 'Color', [1 0.8-0.1*i 0.8-0.1*i]);
    view(30, 60); hold on;
end
```

The code runs simulation at nine different coupler link lengths. It then plots the trajectory coordinates of the coupler link center frame with respect to the world frame. Coupler link lengths range from 20 cm to 28 cm.

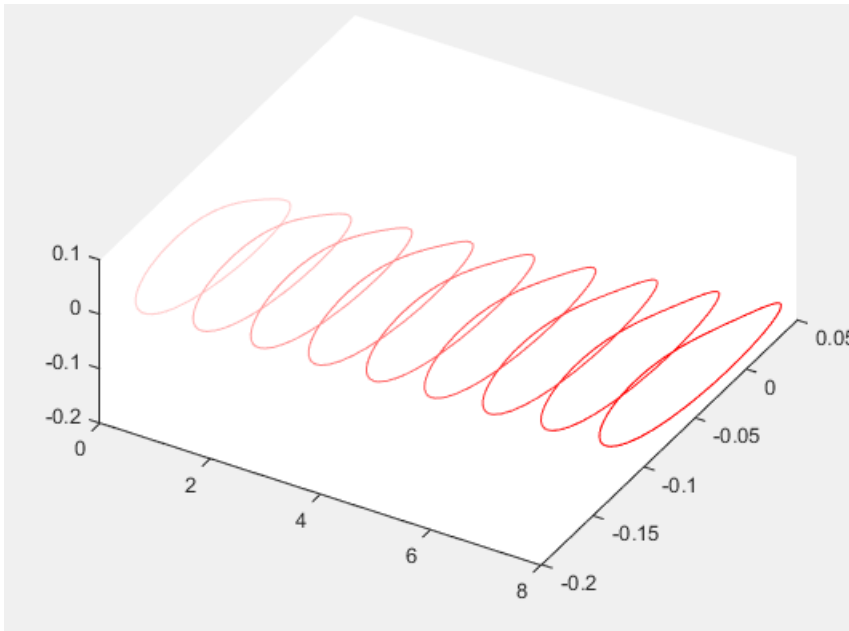
- 3 Save the script as `sim_four_bar` in the folder containing the four-bar model.

Run Simulation Script

Run the `sim_four_bar` script. In the MATLAB Editor toolstrip, click the **Run** button or, with the editor active, press **F5**. Mechanics Explorer opens with a dynamic 3-D view of the four-bar model.



Simscape Multibody iteratively runs each simulation, adding the resulting coupler link curve to the active plot. The figure shows the final plot.



You can use the simple approach shown in this tutorial to analyze model dynamics at various parameter values. For example, you can create a MATLAB script to simulate a crank-slider model at different coupler link lengths, plotting for each simulation run the constraint force acting on the piston.

Measure Forces and Torques Acting at Joints

In this section...

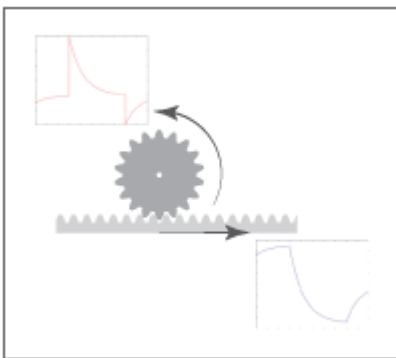
“Open Model” on page 3-100

“Sense Actuation Torque” on page 3-100

“Sense Constraint Forces” on page 3-102

“Sense Total Torques” on page 3-103

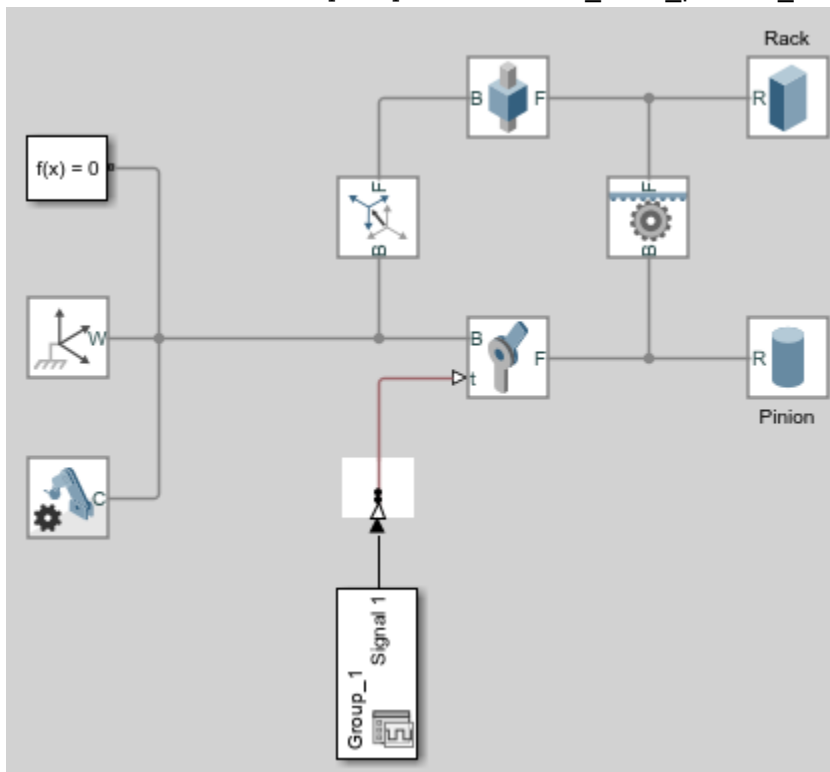
This example shows how to measure the actuator torque, constraint force, and total torque acting on a Revolute Joint block. The example uses a rack and pinion model.



The joint blocks in Simscape Multibody have ports that measure force and torque. You can use these ports to compute and output various types of forces and torques acting directly at joints. For more information about forces and torques, see “Force and Torque Sensing” on page 3-33.

Open Model

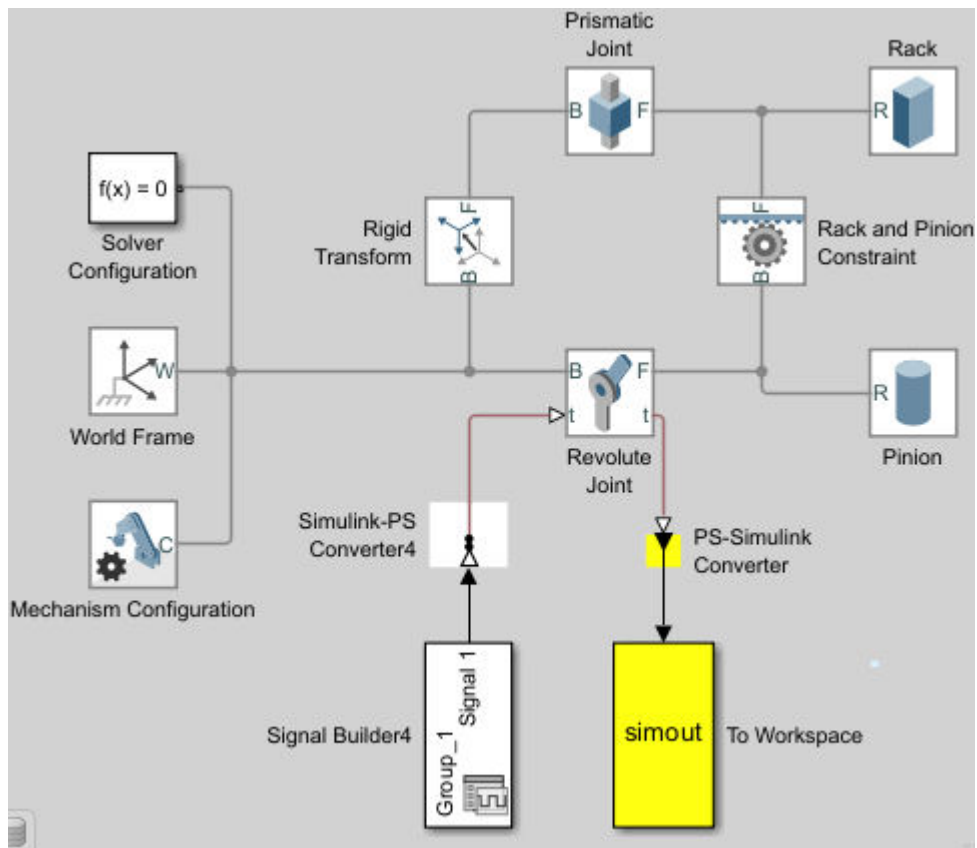
At the MATLAB command prompt, enter `smdoc_rack_pinion_c` to open the rack and pinion model.



Sense Actuation Torque

The rack and pinion model uses a Signal Editor block to specify the actuator torque that drives the pinion. Because the **Input filtering order** property of the Simulink-PS Converter block is set to **Second-order filtering**, the block processes the input torque by smoothing any abrupt changes or discontinuities. To measure the actuator torque at the Revolute Joint block:

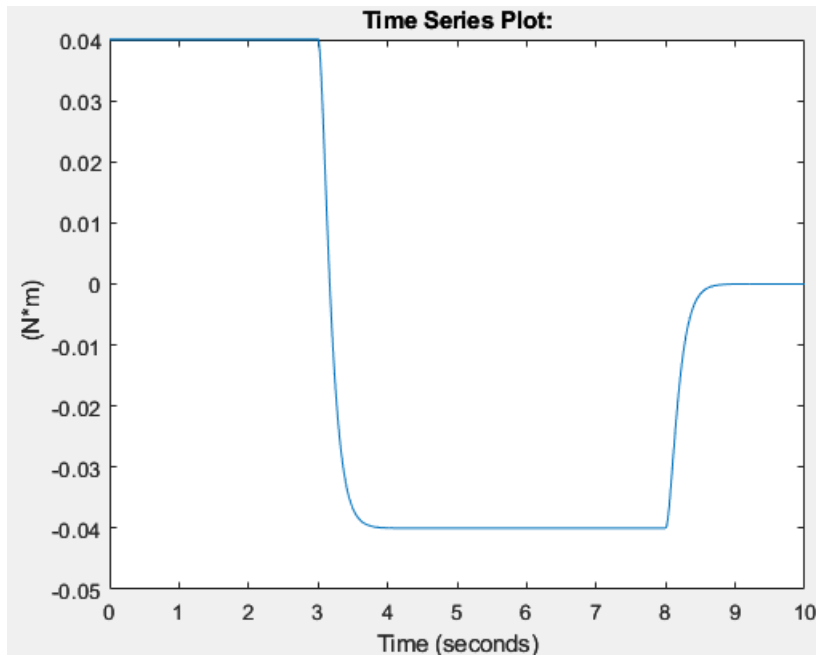
- 1 Double-click the Revolute Joint block to open the block dialog. Under **Z Revolute Primitive (Rz)** > **Sensing**, select **Actuator Torque**. The block exposes the port **t** that outputs the actuator torque, which is a 3-D vector physical signal acting at the joint primitive.
- 2 Add a PS-Simulink Converter block and To Workspace block to the model, then connect the blocks as shown in the figure.



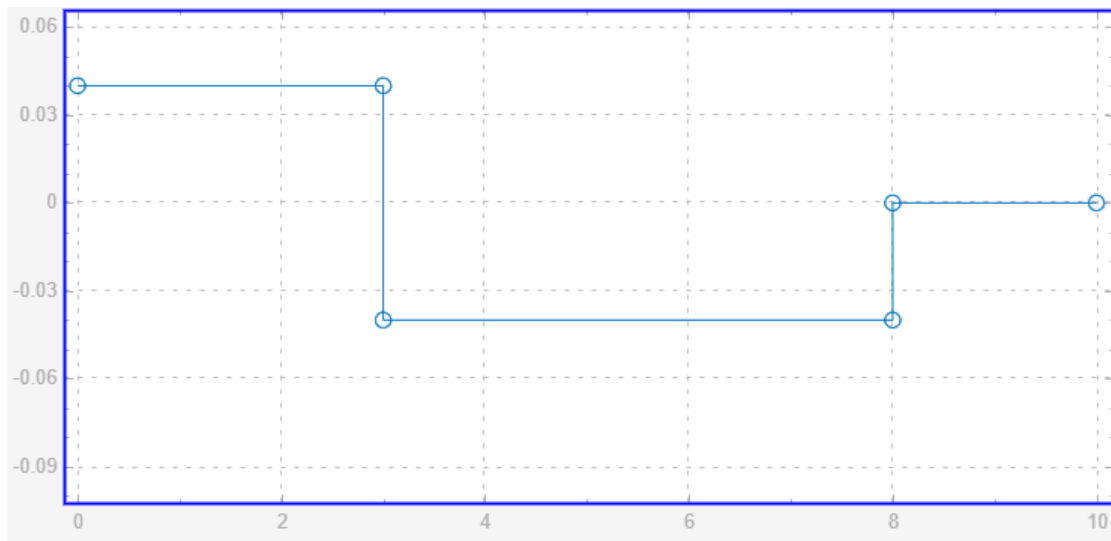
- 3 Double-click the PS-Simulink Converter block. Set the **Output signal unit** parameter to N*m.
- 4 Simulate the model. The To Workspace block outputs the actuator torque to the simout variable in the MATLAB base workspace.
- 5 Plot the torque. At the MATLAB command prompt, enter:

```
figure;
plot(simout);
```

MATLAB plots the joint actuator torque. All but the z-component are zero throughout the simulation.



Compare the actuator torque plot to the input torque specified in the Signal Editor block. Neglecting any signal smoothing due to the second-order filtering, the two signals are identical. This figure shows the input torque.



Sense Constraint Forces

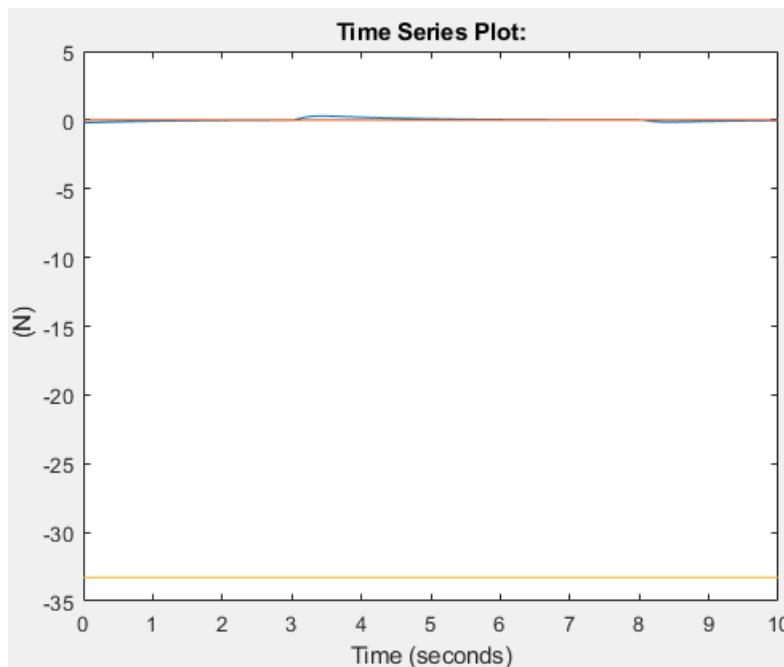
Joint constraint forces, which act normal to the joint primitive axes, restrict motion to the allotted degrees of freedom. In this model, the constraint forces acting at the Revolute Joint block resist the pull of gravity and keep the position of the pinion with respect to the world frame. To sense the constraint forces:

- 1 In the Mechanism Configuration block, set **Uniform Gravity** to Constant. Ensure that the **Gravity** parameter is $[0 \ 0 \ -9.80665]$.

- 2 In the Revolute Joint block, select **Composite Force/Torque Sensing > Constraint Force**. The block exposes the port **fc**. The port outputs constraint forces that act at the joint to maintain the translational constraints.
- 3 Under **Z Revolute Primitive (Rz) > Sensing**, clear **Actuator Torque**.
- 4 Double-click the PS-Simulink Converter block, set the **Output signal unit** parameter to N. Ensure that the PS-Simulink Converter block connects to the port **fc**.
- 5 Simulate the model, then plot the constraint forces. At the MATLAB command prompt, enter:

```
figure;
plot(simout);
```

MATLAB plots the constraint force with respect to time. the x and y -components are zero throughout the simulation. The z component, which opposes the gravity vector, is the only component needed to hold the joint frames in place.



Tip In a Weld Joint block, the constraint forces ensure the base and follower frames remain fixed with respect to each other. You can place a Weld Joint block inside a body subsystem to measure the internal forces or torques acting in that body during a simulation. For an example of how you can do this in a double pendulum model, see “Sense Constraint Forces” on page 3-105.

Sense Total Torques

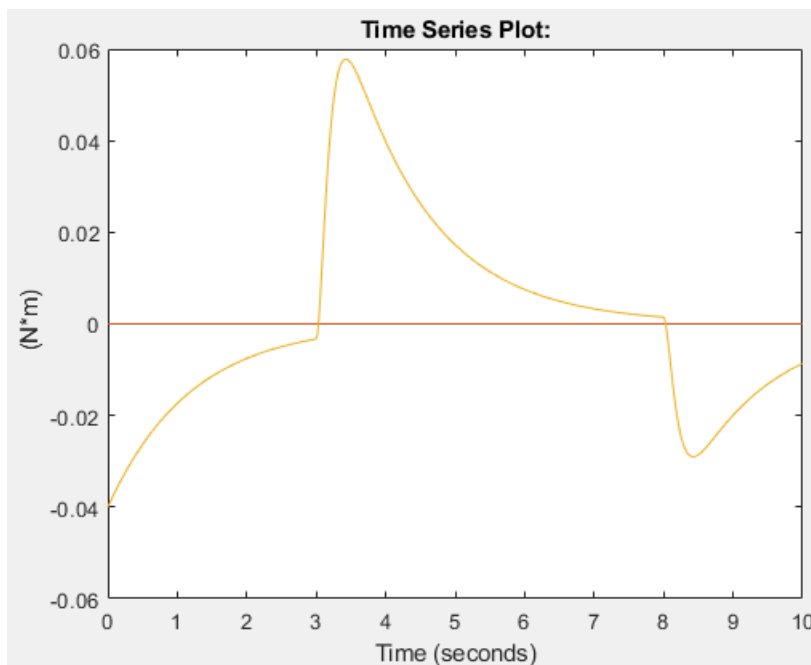
The total force or torque is a sum of all forces or torques that act between the joint base and follower frames and includes contributions from the actuator, constraint, and internal forces or torques. To measure the total torque that acts at the revolute joint:

- 1 Double-click the Revolute Joint block to open the block diagram. Under **Composite Force/Torque Sensing**, select **Total Torque**. The block exposes the port **tt**. This port outputs the total torque that acts at the joint.

- 2 Under **Composite Force/Torque Sensing**, clear **Constraint Force**.
- 3 Double-click the PS-Simulink Converter block, set the **Output signal unit** parameter to N*m. Ensure that the PS-Simulink Converter block connects to the port **tt**.
- 4 Simulate the model.
- 5 Plot the total torque. At the MATLAB command prompt, enter:

```
figure;  
plot(simout);
```

MATLAB plots the total torque, which is a 3-D vector, as a function of time. The x and y -components are zero throughout simulation. The z -component contains torque contributions from the actuation and internal torques.



In the plot, the torque peaks are due to the actuator torque. These peaks decay with time due to the internal damping torques specified in the Revolute Joint block. The damping torques cause the energy dissipation evident in the transient portions of the total torque plot.

See Also

Revolute Joint

More About

- "Force and Torque Sensing" on page 3-33
- "Sense Constraint Forces" on page 3-105

Sense Constraint Forces

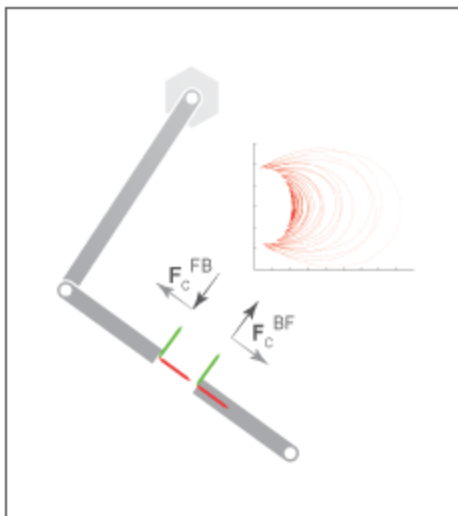
In this section...

“Model Overview” on page 3-105
 “Add Weld Joint Block to Model” on page 3-106
 “Add Constraint Force Sensing” on page 3-106
 “Add Damping to Joints” on page 3-107
 “Simulate Model” on page 3-107
 “Plot Constraint Forces” on page 3-108

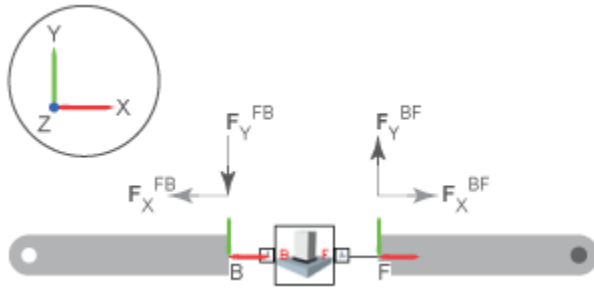
Model Overview

Simscape Multibody provides various types of force and torque sensing. Using joint blocks, you can sense the actuation forces and torques driving individual joint primitives. You can also sense the total and constraint forces acting on an entire joint.

In this tutorial, you use a Weld Joint block to sense the time-varying internal forces that hold a body together. A double-pendulum model, `smdoc_double_pendulum`, provides the starting point for the tutorial. For information on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-13.



By connecting the Weld Joint block between solid elements in a binary link subsystem, you can sense the constraint forces acting between them. The following figure shows the constraint forces that you sense in this tutorial. The longitudinal constraint force aligns with the X axis of the weld joint frames. The transverse constraint force aligns with the Y axis. The constraint force along the Z axis is negligible and therefore ignored.

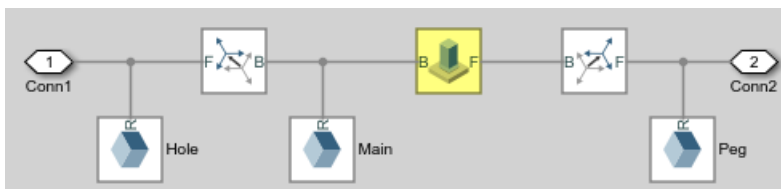


The Weld Joint block enables you to sense the constraint force that the follower frame exerts on the base frame or, alternatively, the constraint force that the base frame exerts on the follower frame. The two forces have the same magnitude but, as shown in the binary link schematic, opposite directions. In this tutorial, you sense the constraint force that the follower frame exerts on the base frame.

You can also select the frame to resolve the constraint force measurement in. The resolution frame can be either the base frame or the follower frame. Certain joint blocks allow their port frames to have different orientations, causing the same measurement to differ depending on your choice of resolution frame. However, because the Weld Joint block provides zero degrees of freedom, both resolution frames yield the same constraint force vector components.

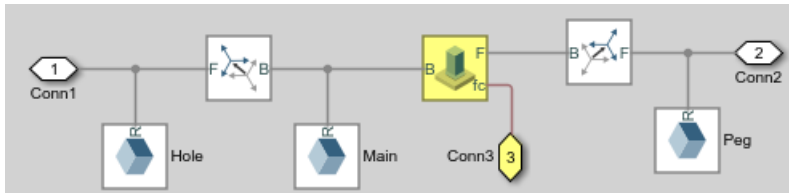
Add Weld Joint Block to Model

- 1 At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double-pendulum model opens up.
- 2 Click the **Look Inside Mask** arrow in the Binary Link A1 subsystem block.
- 3 From the **Simscape > Multibody > Joints** library, drag a Weld Joint block.
- 4 Connect the Weld Joint block as shown in the figure. This block enables you to sense the constraint forces that hold the body together during motion. Because it provides zero degrees of freedom between its port frames, it has no effect on model dynamics.



Add Constraint Force Sensing

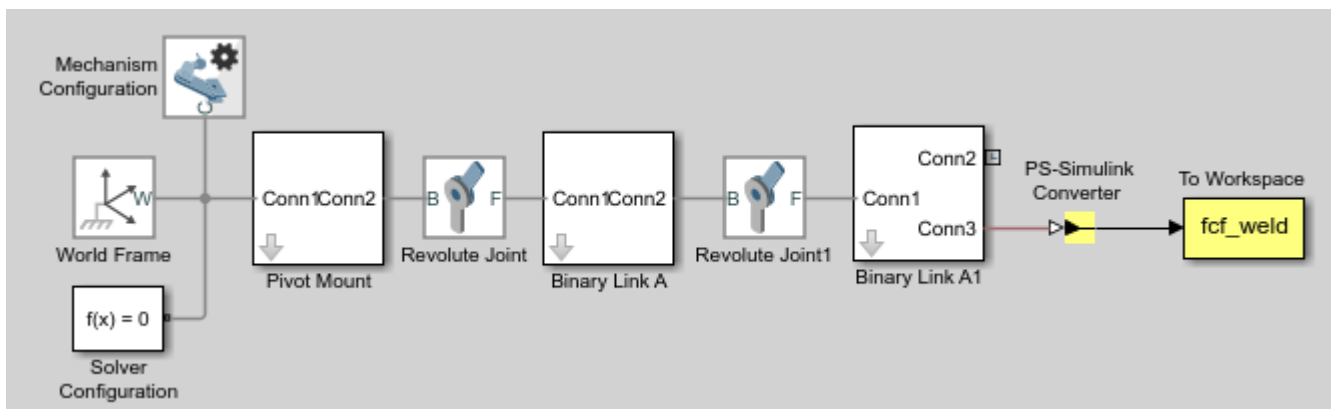
- 1 In the Weld Joint block dialog box, select **Constraint Force**. The block exposes a physical signal output port labeled `fc`.
- 2 Add a Simscape Output port to the subsystem block diagram. Connect the block as shown in the figure and exit the subsystem view.



- 3 Drag the following blocks into the main window of the model. These blocks enable you to output the constraint force signal into the MATLAB workspace.

Library	Block
Simscape > Utilities	PS-Simulink Converter
Simulink > Sinks	To Workspace

- 4 Connect the blocks as shown in the figure. Check that the PS-Simulink Converter block connects to the newly added Simscape port.



- 5 Specify these block parameters.

Block	Dialog Box Parameter	Value
PS-Simulink Converter	Output signal unit	mN
To Workspace	Variable name	fcf_weld

Units of mN are appropriate for this model, which contains Aluminum links roughly 30 cm × 2 cm × 0.8 cm.

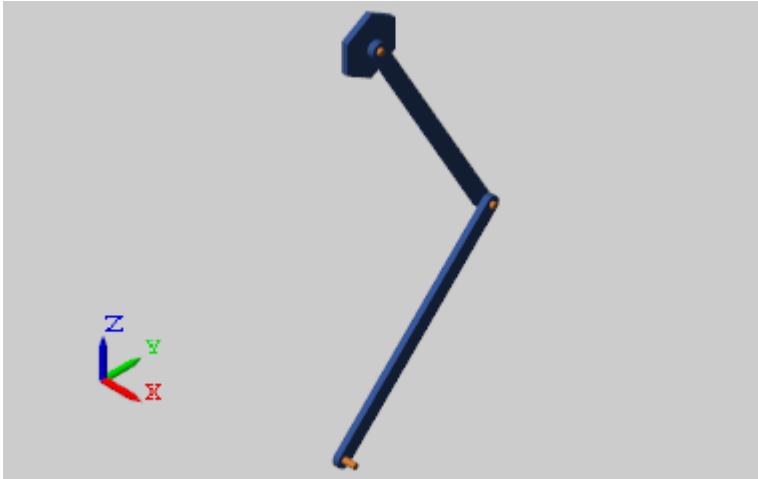
Add Damping to Joints

In each Revolute Joint block dialog box, select **Internal Mechanics > Damping Coefficient** and enter $1e-5$. The damping coefficient enables you to model energy dissipation during motion, so that the double-pendulum model eventually comes to rest.

Simulate Model

- 1 In the **Modeling** tab, click **Model Settings**.
- 2 In the Solver tab of the Configuration Parameters window, set the **Solver** parameter to ode15s. This is the recommended solver for physical models.

- 3 In the same tab, set the **Max step size** parameter to 0.001 s.
- 4 Run the simulation. Mechanics Explorer opens with a dynamic view of the model. In the Mechanics Explorer menu bar, select the Isometric View button to view the double pendulum from an isometric perspective.

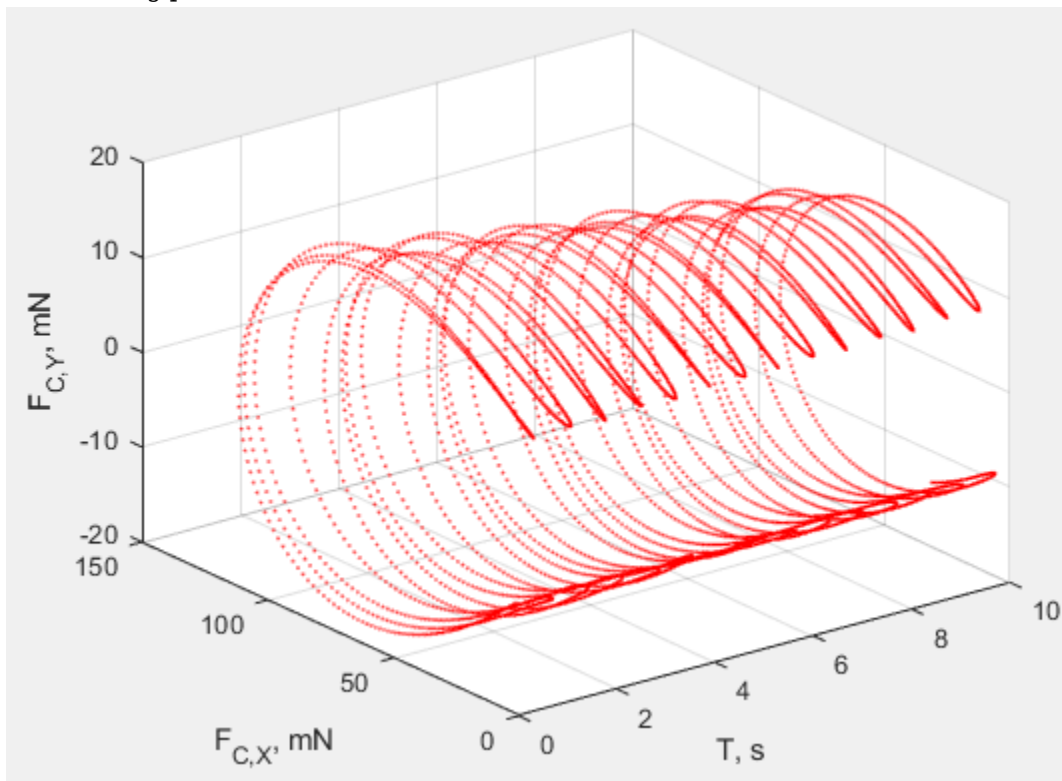


Plot Constraint Forces

At the MATLAB command prompt, enter the following plot commands:

```
figure;  
plot3(fcf_weld.time, fcf_weld.data(:,1), fcf_weld.data(:,2),...  
'.', 'MarkerSize', 1, 'Color', 'r');  
grid on;  
xlabel('T, s');  
ylabel('F_{C,X}, mN');  
zlabel('F_{C,Y}, mN');
```

MATLAB plots the axial and transverse constraint forces with respect to time in 3-D. The figure shows the resulting plot.



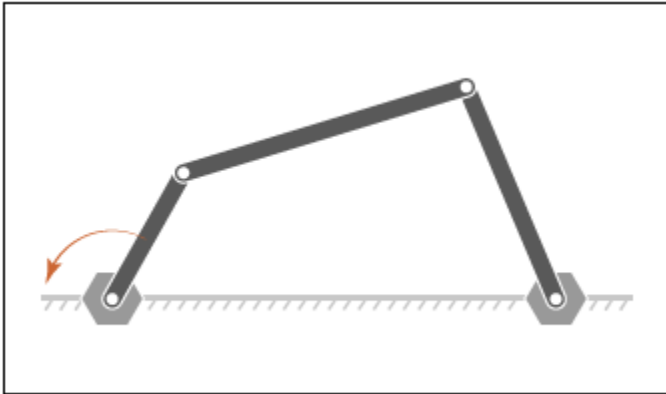
Specify Joint Motion Profile

In this section...

“Build Model” on page 3-110

“Simulate Model” on page 3-112

This example shows how to specify the motion of a Revolute Joint block and sense the corresponding actuation torque at the joint. The example uses a four-bar linkage mechanism.



To drive the linkage, you prescribe a time-varying position signal for the crank angle. Then you use the Revolute Joint block to sense the actuation torque at the joint that corresponds to the prescribed motion.

Build Model

- 1 At the MATLAB command prompt, enter `smdoc_four_bar` to open the linkage model. To learn how to build the model, see “Model a Closed-Loop Kinematic Chain” on page 2-16.
- 2 Double-click the Base-Crank Revolute Joint block and specify these parameters:

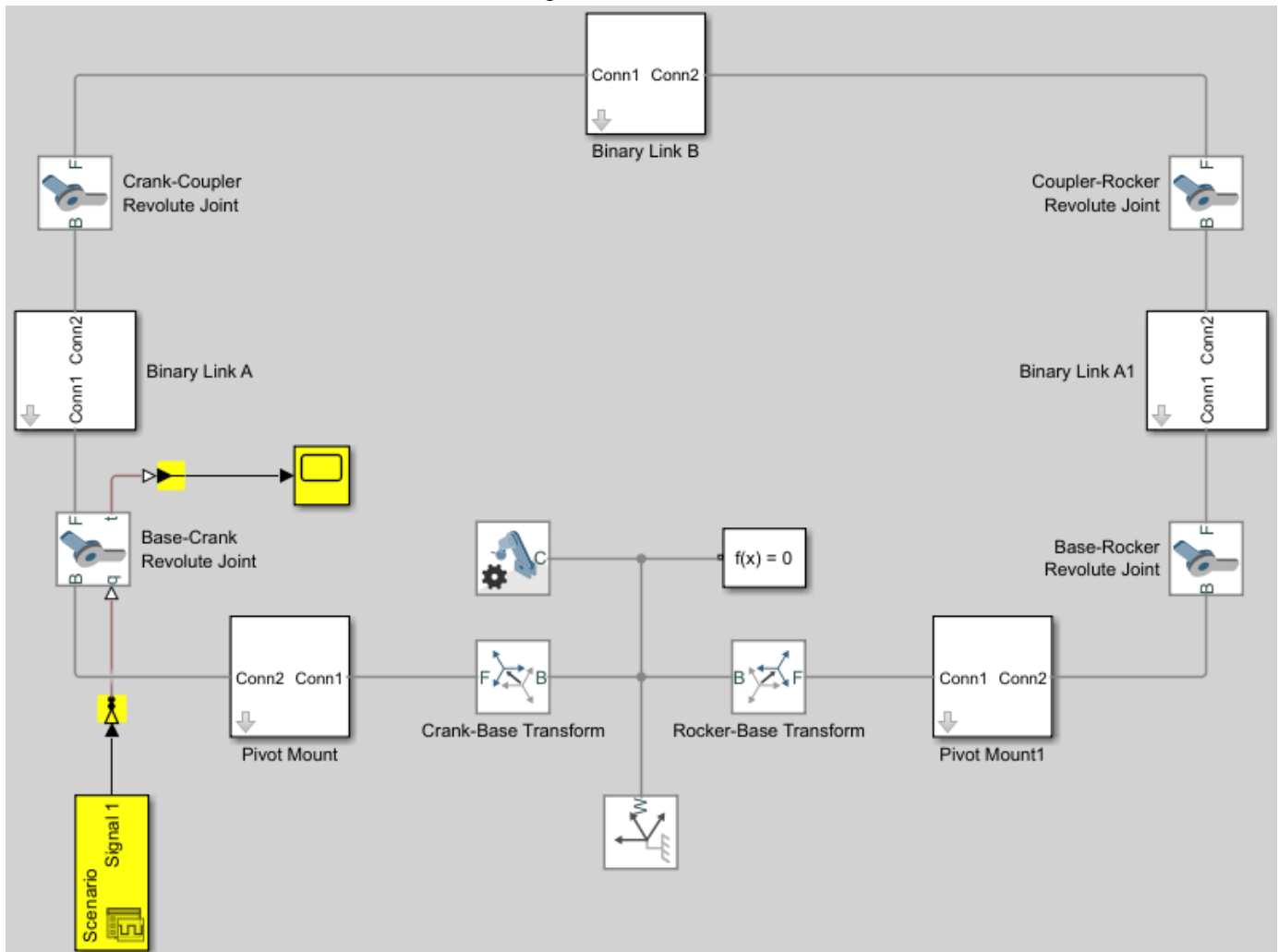
Parameter	Setting
Actuation > Torque	Automatically Computed
Actuation > Motion	Provided by Input
Sensing > Actuator Torque	Selected

The block displays two physical signal ports. Input port **q** accepts the joint angular position. Output port **t** provides the joint actuation torque required to achieve the prescribed angular positions.

- 3 For each of the four Revolute Joint blocks, in the **Internal Mechanics** section, set the **Damping Coefficient** to $5e-4 \text{ N}\cdot\text{m}/(\text{deg}/\text{s})$. During the simulation, damping forces between the joint frames account for dissipative losses at the joints.
- 4 Add these blocks to the model. These blocks enable you to specify an actuation position signal and plot the actuation torque for the joint.

Block	Library
Simulink-PS Converter	Simscape > Utilities
PS-Simulink Converter	Simscape > Utilities
Scope	Simulink > Sinks
Signal Editor	Simulink > Sources

5 Connect the blocks as shown in the figure.




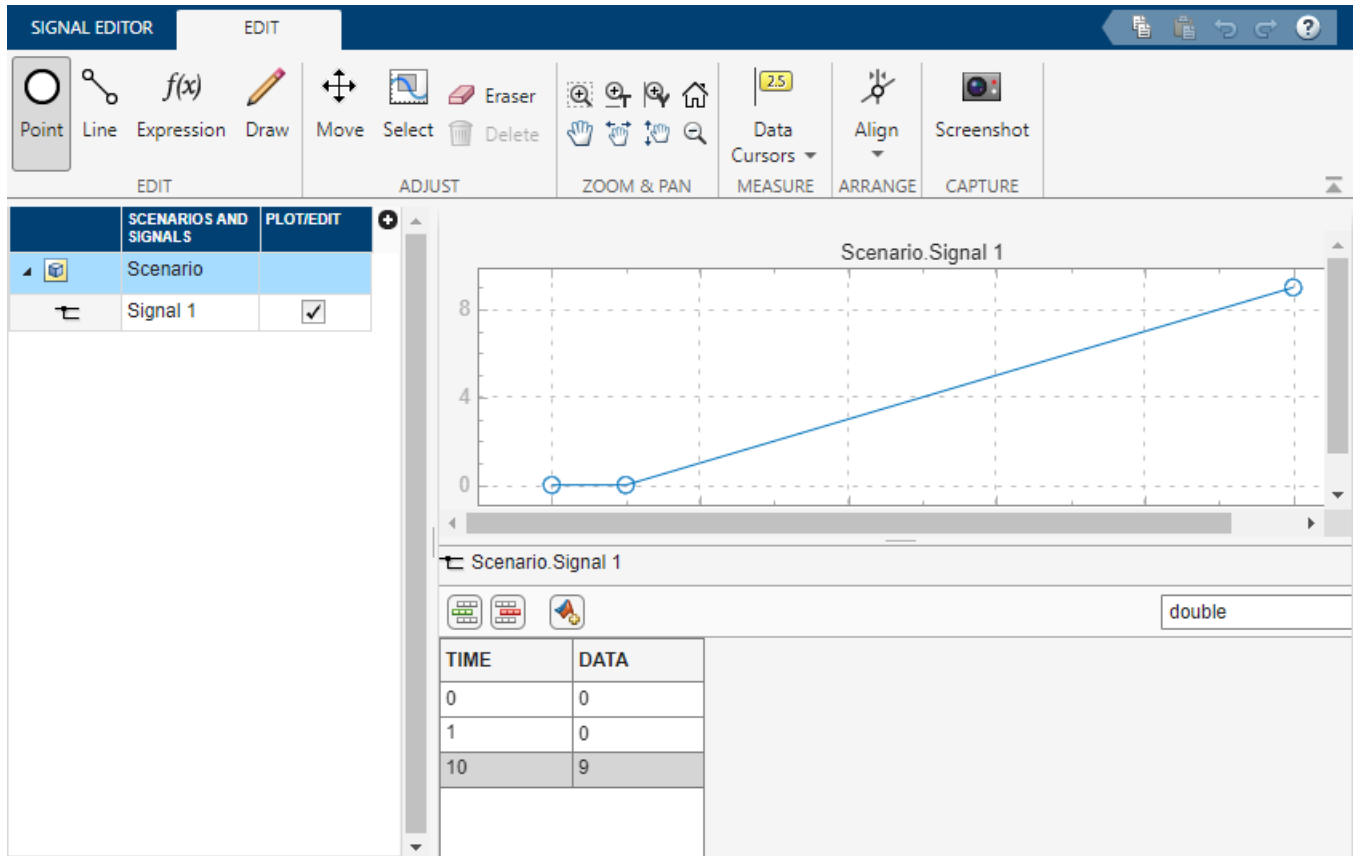
6 Double-click the Simulink-PS Converter block and specify these parameters:

Parameter	Value
Input Heading > Filtering and derivatives	Filter input, derivatives calculated
Input Heading > Input filtering order	Second-order filtering

7 Use the Signal Editor block to specify the position signal.

- Double-click the Signal Editor block to open the dialog box.

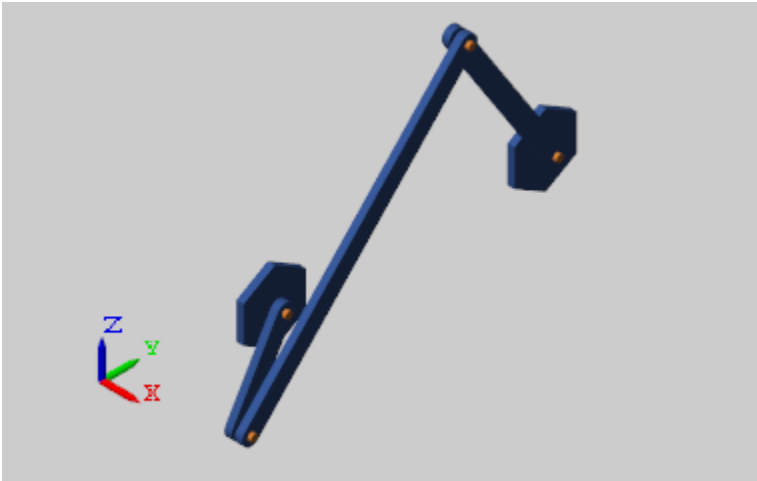
- Under **Signal properties**, click the Launch Signal Editor button  to open the Signal Editor window.
- In the Signal Editor window, in the left pane, expand **Scenario**, select the **Plot/Edit** check box, and edit data as shown in the image. This signal corresponds to a constant angular speed of 1 rad/s from $t = 1$ s onwards.



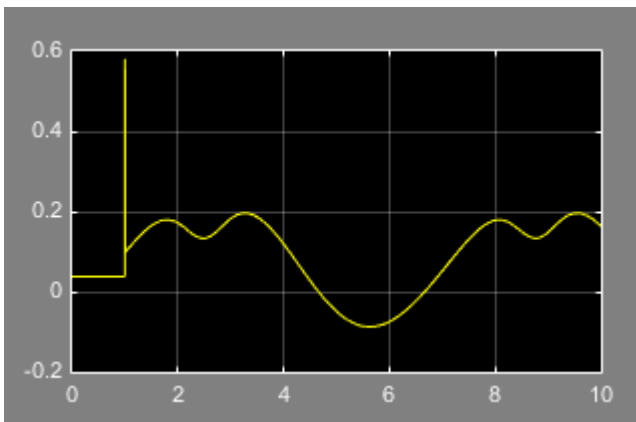
- On the Signal Editor tab, click **Save** to save the data to a MAT-file. Close the Signal Editor window.
- In the dialog box, under **Signal properties**, select **Interpolation data**, click **Apply** and **OK**.

Simulate Model

Run the simulation. Mechanics Explorer opens with a dynamic display of the four-bar model.



Open the Scope window. The plot shows the joint actuation torque with which you can achieve the motion you prescribed.



See Also

Related Examples

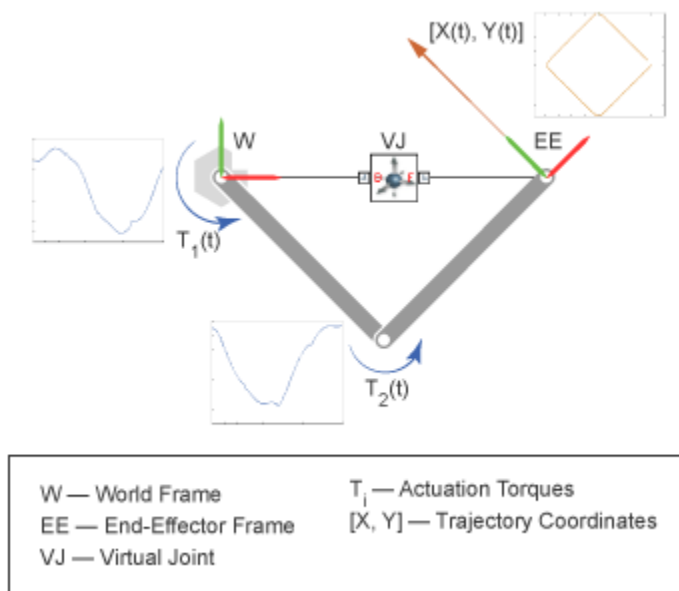
- “Create and Edit Signal Data”
- “Specify Joint Motion in Planar Manipulator Model” on page 3-114
- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specifying Motion Input Derivatives” on page 3-24

Specify Joint Motion in Planar Manipulator Model

In this section...

“Add Virtual Joint” on page 3-114
 “Prescribe Motion Inputs” on page 3-115
 “Sense Joint Actuation Torques” on page 3-118
 “Simulate Model” on page 3-119

This example shows how to specify the trajectory of the end effector of a planar manipulator and compute the required actuator torques for the manipulator joints.

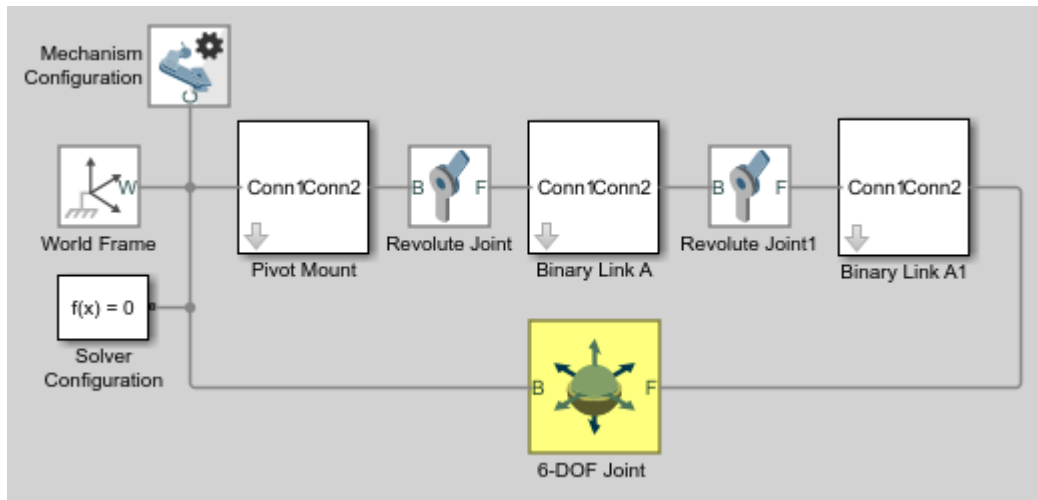


In this example, you specify the time-varying trajectory coordinates of the end-effector frame with respect to the world frame by using a 6-DOF Joint block. This block provides the requisite degrees of freedom between the two frames. In this model, the 6-DOF Joint block does not act as a physical connection.

During the simulation, the end effector traces the specified square pattern and the revolute joints compute the required actuator torques.

Add Virtual Joint

- 1 At the MATLAB command prompt, enter `smdoc_double_pendulum` to open the model. For instructions on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-13.
- 2 Add a 6-DOF Joint block to the model and connect the blocks as shown in the figure.




In this model, the 6-DOF Joint block works as a virtual connection between the world frame and the end-effector frame. The block uses the position inputs to specify the trajectory of the end-effector frame with respect to world frame. Ensure that the base frame of the 6-DOF Joint block connects to the world frame and the follower frame connects to the end-effector frame.

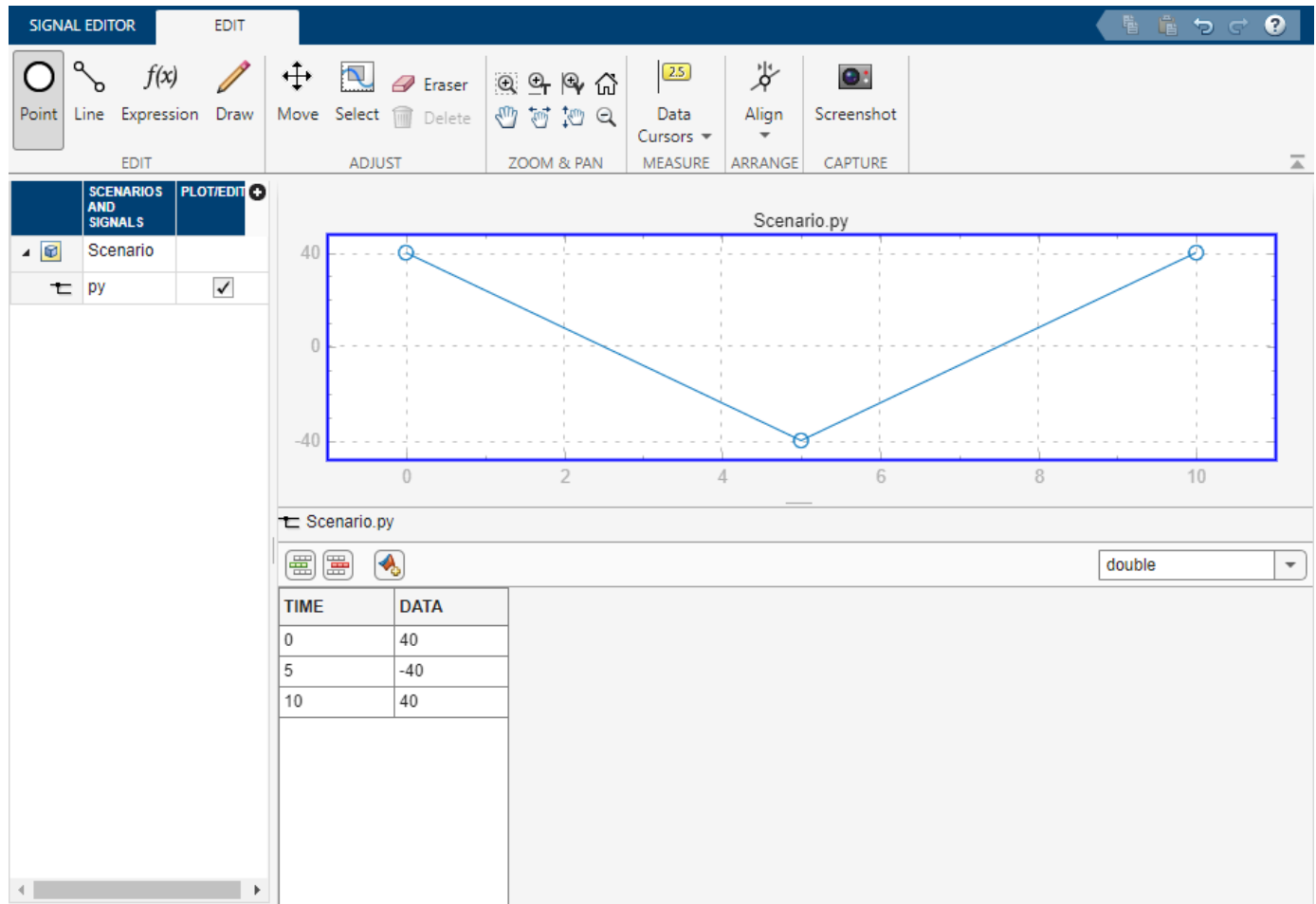
Prescribe Motion Inputs

- 1 Double-click the 6-DOF Joint block to open the block dialog and specify these parameters.

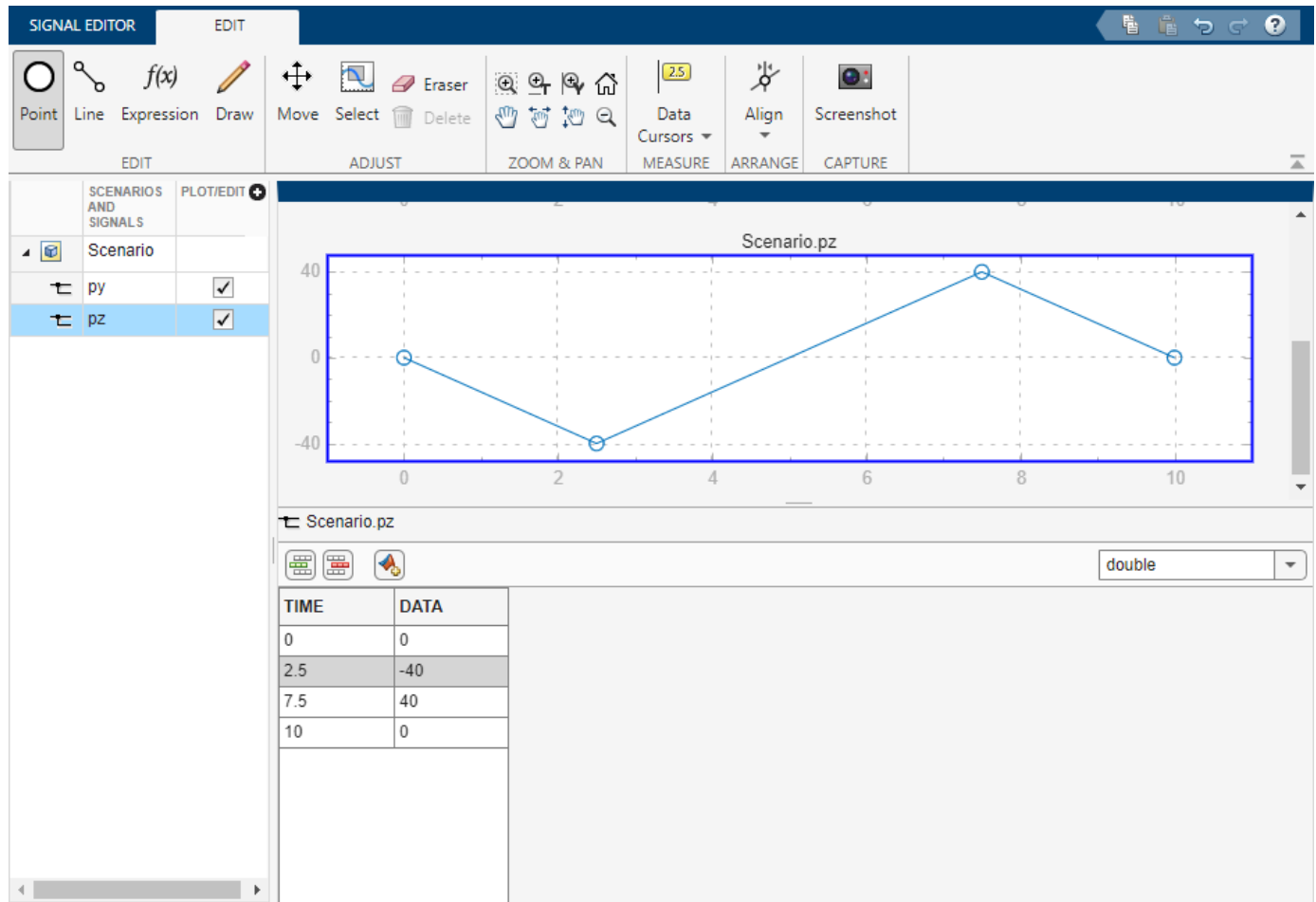
Parameter	Select
Y Prismatic Primitive (Py) > Actuation > Motion	Provided by Input
Z Prismatic Primitive (Pz) > Actuation > Motion	Provided by Input

The block exposes ports **py** and **pz**.

- 2 Add two Simulink-PS Converter blocks and one Signal Editor block to the model.
- 3 Use the Signal Editor block to specify the position inputs.
 - Double-click the Signal Editor block to open the dialog box.
 - Under **Signal properties**, click the Launch Signal Editor button  to open the Signal Editor window.
 - In the left pane, expand **Scenario**, rename the signal 1 to **py**, select the **Plot/Edit** check box, and edit the data as shown in the image.



- In the left pane, right-click **Scenario**, select **Insert > Signal**, rename the new signal to pz, select the **Plot/Edit** check box, and edit data as shown in the image.



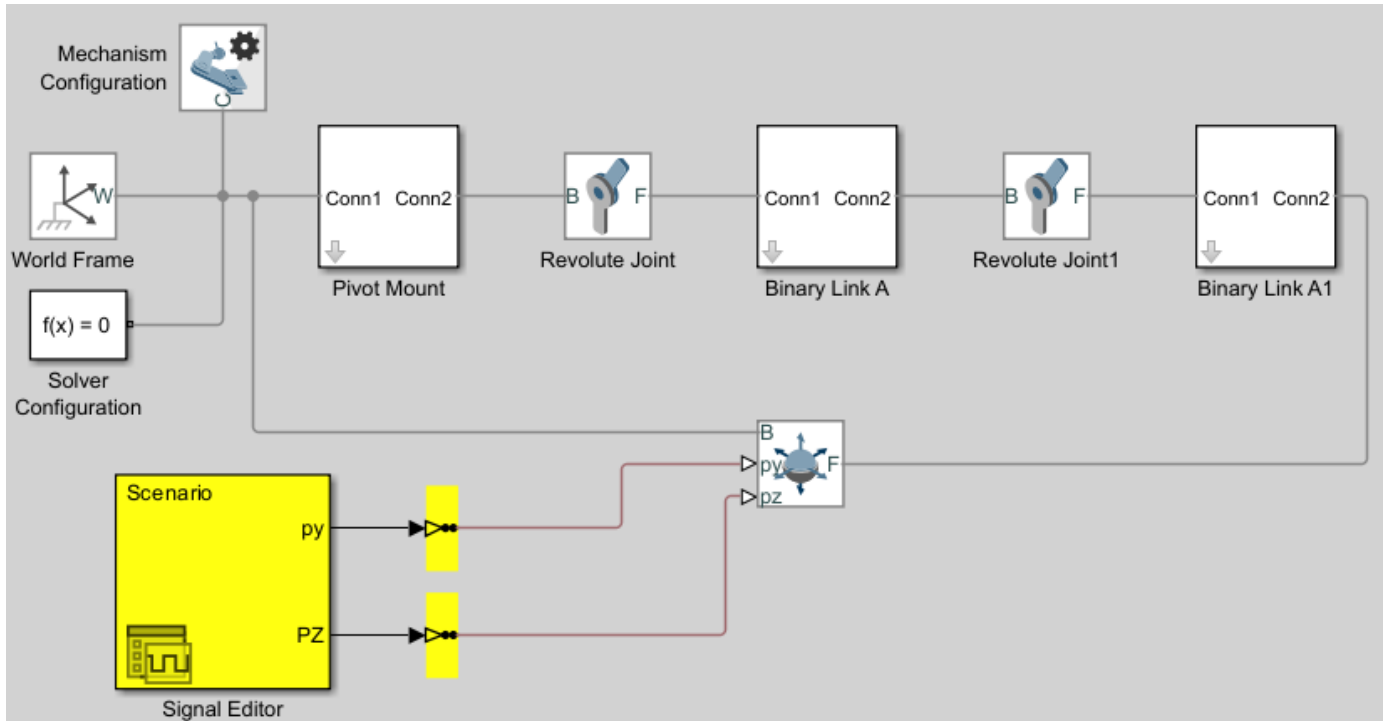
- In the Signal Editor tab, click **Save** to save the data to a MAT-file. Close the Signal Editor window.
 - In the dialog box, under **Signal properties**, set **Active signal** to **py** and select **Interpolate data**. Then, set **Active signal** to **pz** and select **Interpolate data**. Click **Apply** and **OK**.
- 4 Use the Simulink-PS Converter blocks to convert Simulink signals into physical signals.

In the two Simulink-PS Converter blocks, specify these parameters.

Parameter	Value
Units > Input signal unit	cm
Input Handling > Filtering and derivatives	Filter input, derivatives calculated
Input Handling > Input filtering order	Second-order filtering
Input Handling > Input filtering time constant (in seconds)	0.1

Note that a small time filtering constant can slow a simulation significantly. For most Simscape Multibody models, a value of 0.1 seconds often provides a sufficient balance between simulation time and accuracy.

5 Connect the blocks as shown in the figure.



Sense Joint Actuation Torques

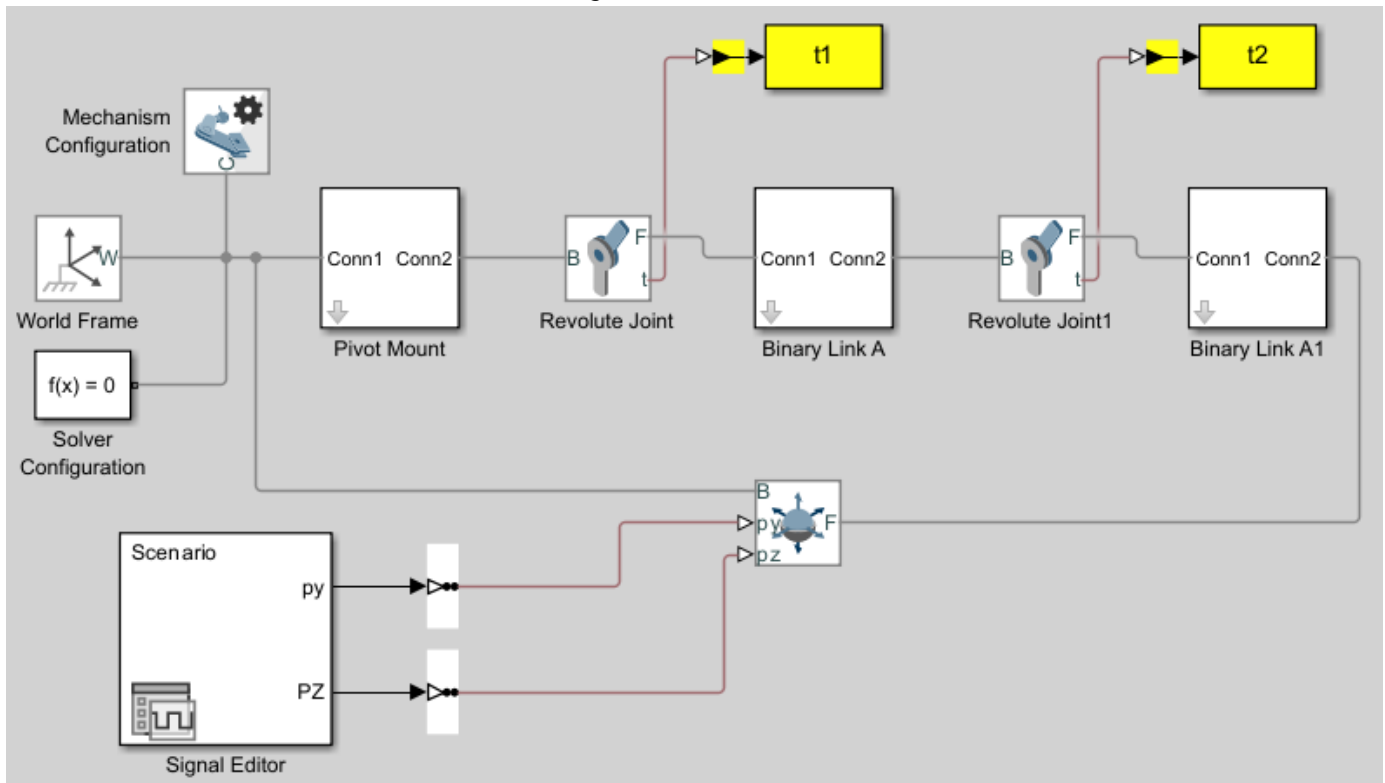
1 In the two Revolute Joint blocks, set these parameters:

Parameter	Setting
Actuation > Torque	Automatically Computed
Sensing > Actuator Torque	Selected

Note Simscape Multibody requires the number of joint primitive degrees of freedom with motion inputs to be equal to the number of joint primitive degrees of freedom with automatically computed joint actuator forces and torques. If the model does not meet this condition, the simulation fails with an error.

- 2 Add two PS-Simulink Converter blocks and two To Workspace blocks to the model.
- 3 In the PS-Simulink Converter blocks, set **Vector format** to 1-D array. The PS-Simulink Converter blocks convert the physical signal outputs into Simulink signals.
- 4 In the two To Workspace blocks, specify the **Variable name** parameter as t1 and t2, respectively.

5 Connect the blocks as shown in the figure.

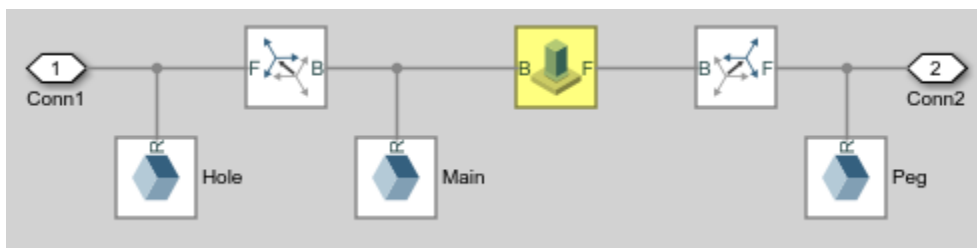


Simulate Model

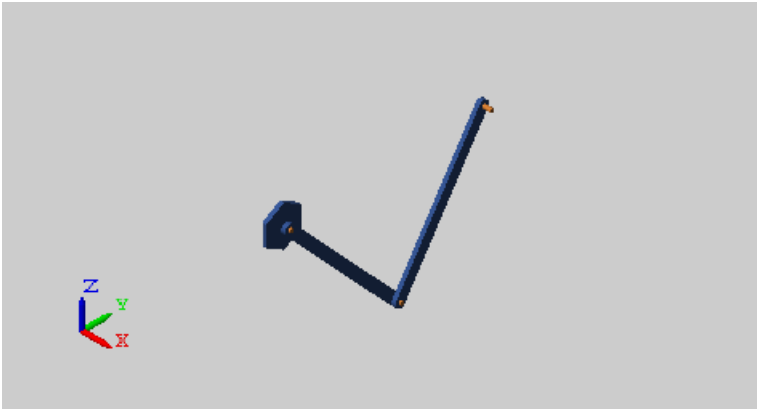
If you run the simulation, the simulink fails with an error.

Note Simscape Multibody requires any closed kinematic loop to contain at least one joint block without motion inputs or automatically computed actuator forces or torques.

To solve the error, add a Weld Joint block to the Binary Link A subsystem and connect the blocks as shown in the figure.



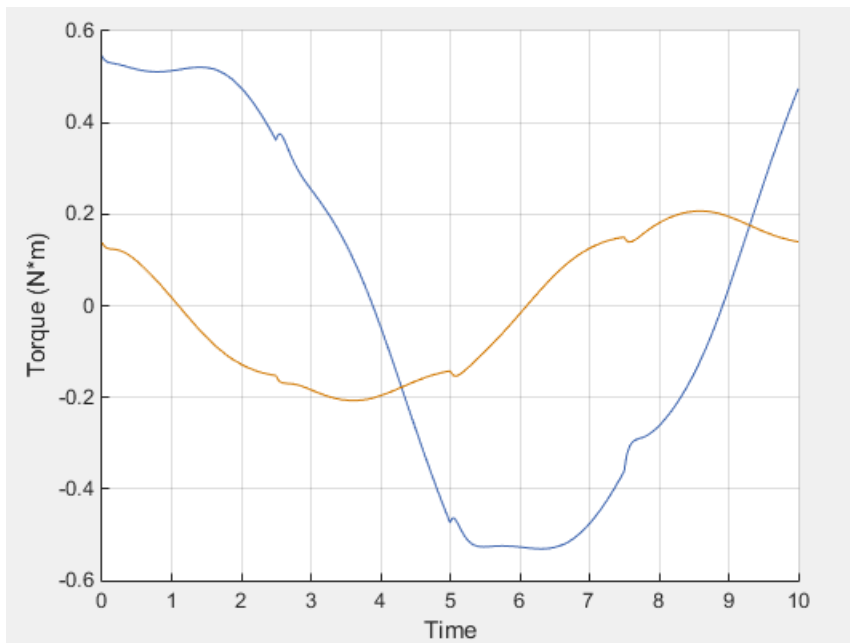
Run the simulation again. Mechanics Explorer opens with a dynamic 3-D display of the planar manipulator model.



Plot the computed actuation torques that act at the two revolute joints in the linkage. At the MATLAB command line, enter:

```
figure;
hold on;
plot(t1.time, t1.data, 'color', [60 100 175]/255);
plot(t2.time, t2.data, 'color', [210 120 0]/255);
xlabel('Time');
ylabel('Torque (N*m)');
grid on;
```

The plot shows the time-varying actuation torques. These torques enable the manipulator end frame to trace the prescribed square trajectory.



See Also

Related Examples

- “Model an Open-Loop Kinematic Chain” on page 2-13
- “Sense Motion Using a Transform Sensor Block” on page 3-79
- “Specify Joint Motion Profile” on page 3-110
- “Specifying Motion Input Derivatives” on page 3-24

More About

- “Actuating and Sensing with Physical Signals” on page 3-28
- “Force and Torque Sensing” on page 3-33
- “Specifying Joint Actuation Inputs” on page 3-19

Simulation and Analysis

Simulation

- “Update and Simulate a Model” on page 4-2
- “Multibody Simulation Issues” on page 4-4

Update and Simulate a Model

In this section...

“Create or Open a Model” on page 4-2

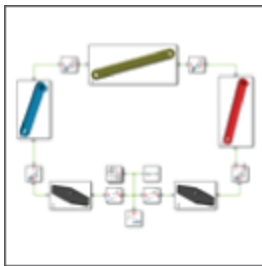
“Update the Block Diagram” on page 4-2

“Examine the Model Assembly” on page 4-3

“Configure the Solver Settings” on page 4-3

“Run Simulation and Analyze Results” on page 4-3

Create or Open a Model



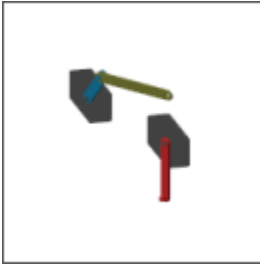
You can create a model manually or import it from a supported CAD application. For an example showing how to create a model manually, see “Model an Open-Loop Kinematic Chain” on page 2-13. For an example showing how to import a model, see “Import a Robotic Arm CAD Model” on page 6-14.

Update the Block Diagram



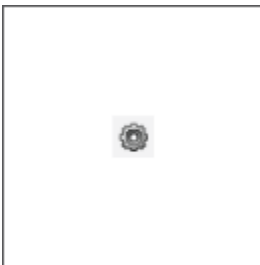
In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a static visualization of the model in its initial state.

Examine the Model Assembly



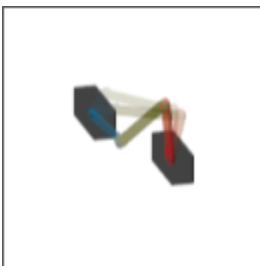
Check the model assembly in the visualization pane of Mechanics Explorer. Look for bodies placed and oriented in unexpected ways. Use the Simscape Variable Viewer or the Simscape Multibody Model Report to identify any assembly issues. For an example showing the use of Model Report, see “Model a Closed-Loop Kinematic Chain” on page 2-16.

Configure the Solver Settings



Open the Configuration Parameters. In the **Modeling** tab, click **Model Settings**. Pick a solver and specify the desired step sizes. Ensure that the time steps are small enough to accurately capture the fastest meaningful changes in your model. Use care, though, as small time steps slow down simulation.

Run Simulation and Analyze Results



Click **Run**. Mechanics Explorer plays a physics-based animation of your model. Examine any data generated during simulation, for example, through Simulink Scope plots. For an example showing how to work with sensing data from a model, see “Sense Motion Using a Transform Sensor Block” on page 3-79.

Multibody Simulation Issues

Under certain conditions, a model that you simulate can behave in unexpected ways. Some issues that you can encounter while simulating a Simscape Multibody model include:

Limited visualization in models with For Each Subsystem blocks. Models with one or more For Each Subsystem blocks simulate with limited visualization. The Mechanics Explorer visualization utility displays the model in only one of the instances provided by the For Each Subsystem blocks. Model simulation is not affected.

No visualization in models with Model blocks. Models with Model blocks (often referred to as referenced models) simulate without visualization. During simulation, Simscape Multibody software issues a warning at the MATLAB command line. If previously closed, The Mechanics Explorer visualization utility does not open.

No Simscape local solver support. Simscape Multibody software does not support Simscape local solvers. If you select a local solver in the Simscape Solver Configuration block, the solver does not apply to the Simscape Multibody portion of a model—which relies exclusively on the global Simulink solver selected in the **Model Configuration Parameters** window.

Visualization and Animation

- “Enable Mechanics Explorer” on page 5-2
- “Working with Animation” on page 5-3
- “Manipulate the Visualization Viewpoint” on page 5-4
- “Visualization Cameras” on page 5-8
- “Create a Dynamic Camera” on page 5-12
- “Selective Model Visualization” on page 5-15
- “Selectively Show and Hide Model Components” on page 5-20
- “Visualize Simscape Multibody Frames” on page 5-24
- “Go to a Block from Mechanics Explorer” on page 5-27
- “Create a Model Animation Video” on page 5-28

Enable Mechanics Explorer

The Mechanics Explorer visualization utility opens by default whenever you update or simulate a Simscape Multibody model. Each model that you update or simulate adds a new tab to Mechanics Explorer. If Mechanics Explorer does not open, ensure that model visualization is turned on:

- 1** In the **Modeling** tab, click **Model Settings**.
- 2** Expand the **Simscape Multibody** node and select **Explorer**.
- 3** Ensure the **Open Mechanics Explorer on model update or simulation** check box is selected.

Working with Animation

In this section...

“Animation Playback” on page 5-3

“Looping Playback” on page 5-3

“Changing Playback Speed” on page 5-3

“Jumping to Playback Time” on page 5-3

Animation Playback

Animation is cached during model simulation. What you see when you run a simulation is the animation playback, unless the simulation is slower than the animation caching. In that case, the animation goes no faster than the simulation can produce the cache.

Once a partial or complete animation is cached from simulation, starting the animation again plays back the cache, without running the simulation a second time. You can move backward and forward to any time in the cached animation.

The animation cache is stored until you close Mechanics Explorer. When you simulate the model, the cache is updated with new animation data. To create a permanent record of a model animation, you must create an animation video. See “Create a Model Animation Video” on page 5-28

Looping Playback

Use the **Toggle Loop** button in the Mechanics Explorer playback toolstrip to automatically replay an animation from the start once it reaches the end. The cached animation replays indefinitely until you click the **Pause** button. Enable looping by clicking the **Toggle Loop** button. Disable looping by clicking the button again.

Changing Playback Speed

Use the playback speed slider in the Mechanics Explorer toolstrip to set the animation playback speed ratio relative to real time. Set the slider to a number greater than 1 for faster playback. Set it to a number smaller than 1 for slower playback.

You can set the slider to multiples of 2 from 1/256 to 256. For slower or faster animations, adjust the base playback speed for the model. To change this parameter, from the Mechanics Explorer menu bar, select **Tools > Animation Settings**.

Jumping to Playback Time

Use the playback slider in the Mechanics Explorer toolstrip to move the playback time to an arbitrary point in the animation timeline. The playback time counter shows the current playback time. Alternatively, enter the desired playback time directly in the playback time counter.

Manipulate the Visualization Viewpoint

In this section...

- “Model Visualization” on page 5-4
- “Select a Standard View” on page 5-4
- “Set View Convention” on page 5-5
- “Rotate, Roll, Pan, and Zoom” on page 5-6
- “Split Model View” on page 5-7

Model Visualization

Multibody models lend themselves to 3-D visualization, a qualitative means of analysis that you can use to examine body geometries, mechanical connections, and trajectories in three-dimensional space. In Simscape Multibody, you can visualize a model using Mechanics Explorer, adjusting the view point and detail level as needed. You can modify the model view by:

- Selecting a view convention.
- Selecting a standard view.
- Rotating, panning, and zooming.

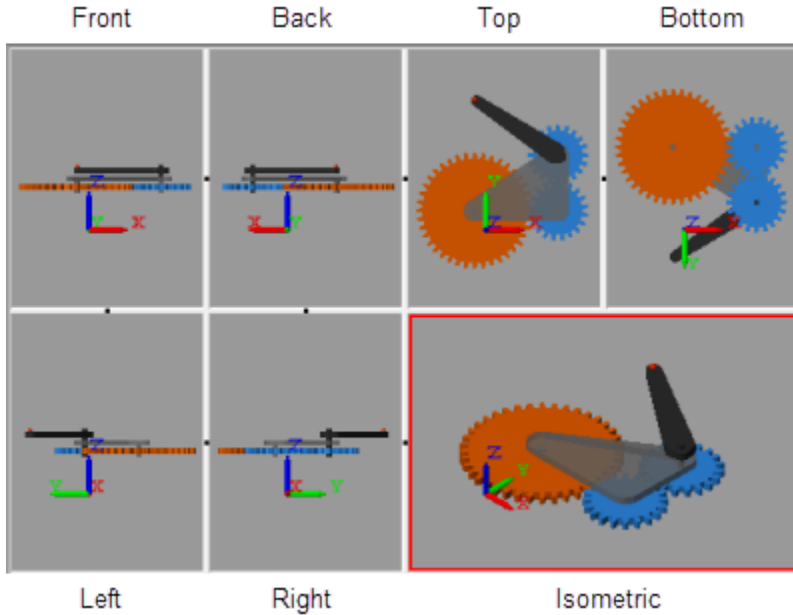
Select a Standard View

Some view points are so widely used that they are called standard. The isometric view point, corresponding to equal 120° angles between any two world frame axes, is one example. In Mechanics Explorer, you can select such view points by clicking the standard view buttons.



Standard View Buttons

The figure shows a Cardan gear model from the different view points using a Z up (XY Top) view convention.

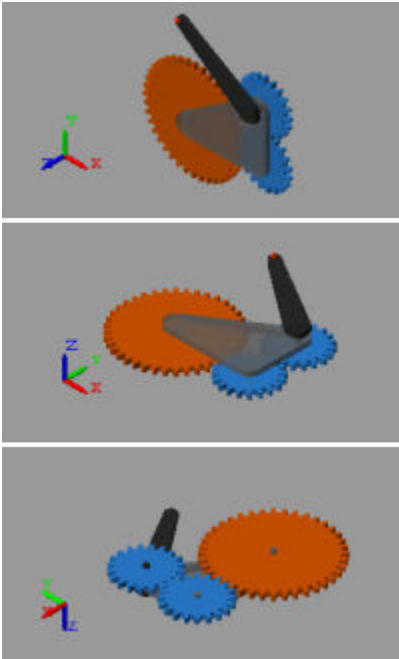


Set View Convention

The view convention helps to determine the perspective from which you view your model. You can align three world frame axes with the vertical direction on your screen, each corresponding to a different view convention:

- Y up (XY Front)
- Z up (XY Top)
- Z down (YZ Front)

The figure shows a Cardan gear model from an isometric perspective using the three view conventions: Y up, Z up, and Z down.








To change the view convention:

- 1 In the Mechanics Explorer tool strip, set **View convention** to one of the three options.
- 2 Select a standard view button.

The new view convention takes effect the moment you select a standard view.

Rotate, Roll, Pan, and Zoom

To view your model from an arbitrary point of view or at varying zoom levels, use the Rotate, Roll, Pan, and Zoom buttons. You can find these buttons in the Mechanics Explorer tool strip:

-  — Rotate the camera about a general 3-D axis.
-  — Roll the camera about its current aim axis.
-  — Pan the camera in the current visualization plane.
-  — Increase or decrease the camera zoom level.
-  — Change the camera zoom to show only the selected region.

You can also use keyboard-and-mouse shortcuts. The table summarizes the available shortcuts.

Button	Shortcut
Rotate	<ol style="list-style-type: none"> 1 Click and hold the mouse scroll wheel. 2 Move the mouse in the direction you want to rotate the model.

Button	Shortcut
Pan	<ol style="list-style-type: none"> 1 Press and hold Shift. 2 Click and hold the mouse scroll wheel. 3 Move the mouse in the direction you want to pan the model.
Zoom	<ol style="list-style-type: none"> 1 Press and hold Ctrl. 2 Click and hold the mouse scroll wheel. 3 Move the mouse up to zoom in, down to zoom out.

Split Model View

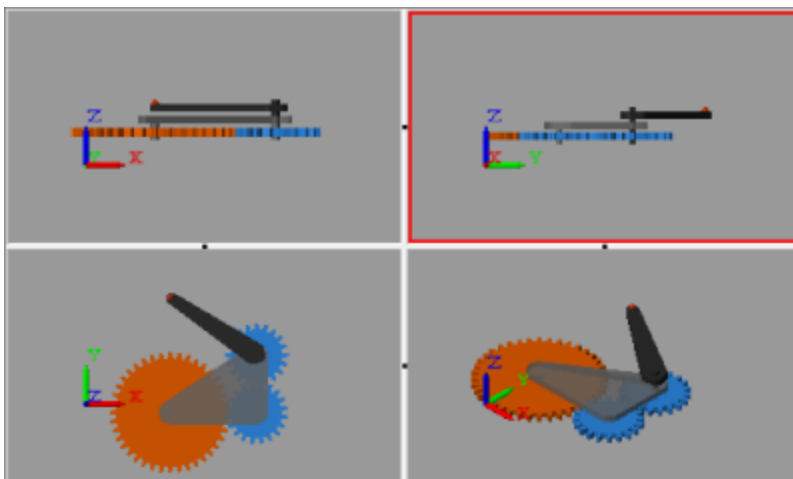
You can view your model from different perspectives, for example, to examine its motion in different planes. So that you can compare different model views, Mechanics Explorer enables you to split the visualization pane into tiles, each with its own view. To split the screen, you use the Mechanics Explorer toolbar buttons shown in the figure.



Use the buttons to:

- Split the model view into four equally sized tiles, each with a different view point (front, right, top, and isometric views).
- Merge all tiles into a single pane with the view point of the last highlighted tile.
- Split a visualization tile vertically or horizontally into two equally sized tiles.

The figure shows the Cardan gear model with a four-way visualization split.



You can merge two tiles by clicking the black dot between the tiles. To ensure that the resulting tile uses the view point of one or the other tile, select that tile first before clicking the black dot between the tiles.

Visualization Cameras

In this section...

“Camera Types” on page 5-8

“Global Camera” on page 5-9

“Dynamic Cameras” on page 5-9

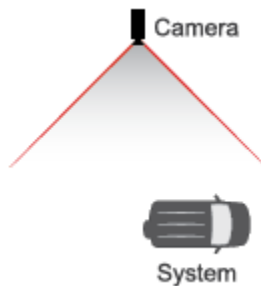
“Camera Trajectory Modes” on page 5-9

“Dynamic Camera Selection” on page 5-10

“Dynamic Camera Reuse” on page 5-11

Camera Types

Cameras define the model viewpoints used during animation playback. Mechanics Explorer supports two camera types—global and dynamic. The global camera provides a static viewpoint that you can manipulate interactively during animation playback. Dynamic cameras provide moving viewpoints that you predefine using **Camera Manager**.



Camera in a Model

The moving viewpoint of a dynamic camera enables you to more easily track the motion of a system. You can use a dynamic camera to keep a moving vehicle such as an automobile or aircraft in view during animation playback. You must define and select a dynamic camera in order to use it in a model. See “Create a Dynamic Camera” on page 5-12.

The figure shows a model visualization captured from the view point of a dynamic camera. This model is part of the “Configuring Dynamic Cameras - Vehicle Slalom” on page 8-64 featured example. You can open the model from the MATLAB command prompt by entering `sm_vehicle_slalom`.



Example of a Visualization Captured with a Dynamic Camera

Global Camera

The global camera is a static camera that:

- Has no planned trajectory.

You must manipulate the camera manually to change the camera viewpoint, for example, by using the Pan, Rotate, Roll, and Zoom buttons.

- Is external to the model.

You cannot position the global camera between bodies, for example, to prevent one body from obstructing another during animation playback.

- Uses an orthographic projection.

Apparent body sizes remain constant regardless of object distance to the camera. This effect, shown in the figure, is consistent with a camera located relatively far from the model.



The global camera is the default camera for all model visualization tiles—each a subdivision of the model visualization pane, when split. In the absence of custom dynamic cameras, the global camera is the only camera available in a model.

Dynamic Cameras

Dynamic cameras are custom cameras that:

- Have planned trajectories.

Every dynamic camera follows a trajectory that you prespecify through Camera Manager. You cannot use the Pan, Rotate, Roll, or Zoom buttons during animation playback.

- Can be internal to a model.

Dynamic cameras can be inside or outside the perimeter of a model. Position a camera between bodies for a viewpoint internal to the model.

- Use a perspective projection.

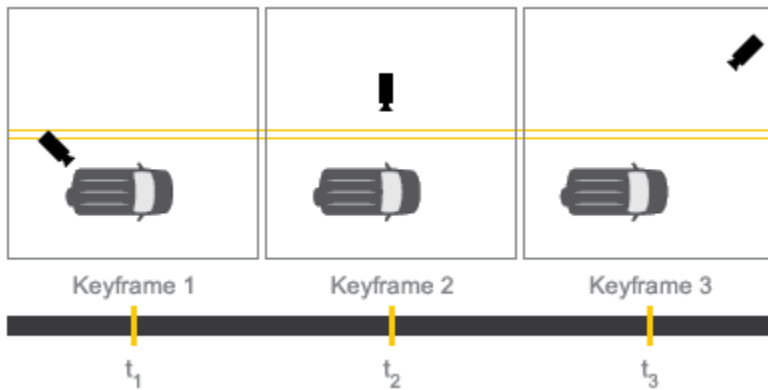
Apparent body sizes vary noticeably with object distance to the camera, creating a more realistic 3-D effect. This effect, shown in the figure, is consistent with a camera located relatively close to the model.



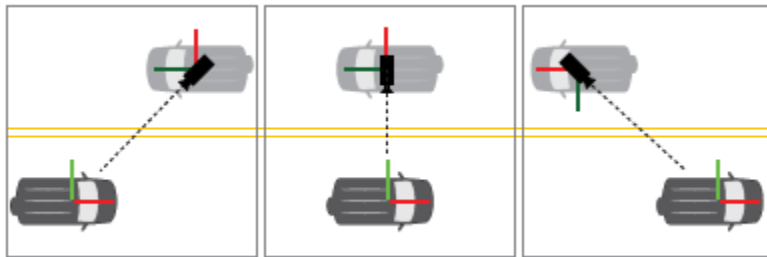
Camera Trajectory Modes

Camera Manager provides two dynamic camera modes that you use to define the camera trajectories:

- **Keyframes** — Set the camera viewpoints at various playback times. Each viewpoint constitutes a keyframe. During playback, the camera transitions between the keyframes using the smooth interpolation method of the `pchip` MATLAB function. Use this camera mode to obtain camera trajectories independent of any components in your model.

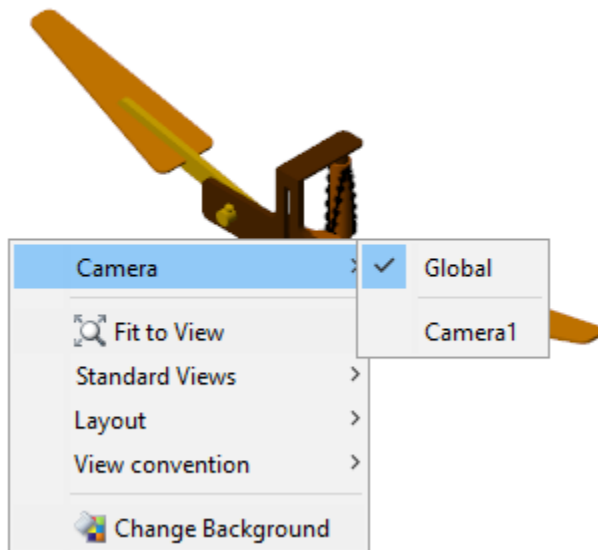


- **Tracking** — Constrain the camera position, aim, and up vector to coordinate frames in your model. During playback, the camera moves with the frames it is constrained to, translating and rotating as needed to satisfy the specified constraints. Use this camera mode to track frames and bodies during playback.



Dynamic Camera Selection

To visualize a model using a dynamic camera, you must first select that camera. To do this, Mechanics Explorer provides the list of available cameras in the visualization pane context-sensitive menu. Right-click the visualization pane to open the menu and select **Camera** to select from the list.



Dynamic Camera Reuse

Dynamic cameras exist only in the models they are defined in. The camera trajectories are based on model-specific frames or viewpoints and are not transferable to other models. You cannot move, copy, or reference a dynamic camera outside of its model. To use a camera in a different model, recreate the camera in that model.

See Also

Related Examples

- “Create a Dynamic Camera” on page 5-12

Create a Dynamic Camera

In this section...

“Start a New Camera Definition” on page 5-12
 “Define a Keyframes Camera” on page 5-12
 “Define a Tracking Camera” on page 5-13
 “Select a Dynamic Camera” on page 5-14

Start a New Camera Definition

If you are new to dynamic cameras, see “Visualization Cameras” on page 5-8. To start a new dynamic camera:

- 1 Simulate the model that you want to add the camera to.

Dynamic cameras exist only in the models that you define them in.

- 2 In the Mechanics Explorer menu bar, select **Tools > Camera Manager**.

Camera Manager opens with a list of previously created dynamic cameras. The list is by empty until you create your first camera.

- 3 In Camera Manager, click the  button.

Camera Manager switches to a camera definition view that lets you select the camera mode and specify the camera motion.

- 4 In the **Camera Name** field, enter a name for your camera.

Make the camera name descriptive so that you can later identify it when selecting an active camera from the Mechanics Explorer visualization context-sensitive menu.

Complete the camera definition by selecting the camera mode and specifying the camera motion. See:

- “Define a Keyframes Camera” on page 5-12 to define the camera motion in **Keyframes** mode. Keyframes are viewpoints that you specify at various playback times and that Simscape Multibody software interpolates to obtain smooth camera trajectories.
- “Define a Tracking Camera” on page 5-13 to complete the camera definition in **Tracking** mode. Tracking constraints include position, aim, and up vector constraints that you specify relative to coordinate frames in a model.

Define a Keyframes Camera

- 1 In Camera Manager, set the **Mode** parameter to **Keyframes**.

Camera Manager switches to a **Keyframes** view that lets you define the camera keyframes.

- 2 In the Mechanics Explorer toolbar, set the playback time for the current keyframe.

Drag the playback slider to the desired point in the animation timeline. Alternatively, enter the time directly in the playback time counter.

- 3 In the visualization pane or tile, manipulate the model viewpoint for your keyframe.

Use the Rotate, Roll, Pan, and Zoom buttons to manipulate the model viewpoint. Use the preset view buttons to obtain standard views such as front, side, or isometric.

- 4 In the Camera Manager **Keyframes** window, click the **Set** button.

Playback must be paused or stopped. Camera Manager commits the keyframe to the camera. The playback slider identifies the keyframe with a colored line marker located at the specified playback time.



- 5 Set new keyframes as in steps 2-4 until you are satisfied with the camera motion.

Simscape Multibody software transitions between keyframes using the smooth interpolation method of the `pchip` MATLAB function to yield the final camera motion.

- 6 Click the **Save** button in the camera definition and main panes of Camera Manager.

Camera Manager saves the camera and its motion to the model. The visualization context-sensitive menu adds the camera to the list of available cameras.

To edit an existing keyframe, use the **Previous** and **Next** buttons to navigate to the keyframe you want to edit. Then, repeat the procedure for adding a keyframe. Use the colored markers in the playback slider to identify the existing keyframes in your dynamic camera.

Click the **Remove** button if you want to delete the current keyframe. Click the **Save** button in the main pane to commit your changes to the camera.

Define a Tracking Camera

- 1 In Camera Manager, set the **Mode** parameter to **Tracking**.

Camera Manager switches to a **Tracking** view that lets you define the camera constraints—position, aim, and up vector—relative to frames in your model.

- 2 In the Camera Manager tracking window, set the camera **Position**, **Aim**, and **Up Vector** constraints:

- a In the tree view or visualization pane, select a frame to constrain the camera to.

If using the visualization pane, click a frame icon. If using the tree view pane, click a frame node. It is not enough to click the body that the frame belongs to.

- b Click the **Use Selected Frame** button to constrain the camera motion to the frame.

If you accidentally select the wrong frame, pick a new frame and click the **Use Selected Frame** button again.

- c For the **Aim** and **Up Vector** dropdown lists, select how to constrain the camera:

- The **Position** constraint fixes the camera to the frame origin only and has no options dropdown list.
- The **Aim** constraint provides the option to aim the camera at the frame origin or along a selected frame axis.
- The **Up vector** constraint provides the option to align the up vector along a selected frame axis.

Select a Dynamic Camera

The dynamic cameras that you create through Camera Manager are by default inactive during animation playback. To set a particular camera as the active camera for a visualization pane, use the visualization pane context-sensitive menu. You can perform this task separately for each visualization pane that you have open in Mechanics Explorer:

- 1** Right-click the visualization pane or tile whose camera you want to switch.

The visualization context-sensitive menu opens up.

- 2** Select **Cameras** and, from the cameras list, select the desired camera.

The model viewpoint switches to that provided by the selected camera.

Selective Model Visualization

In this section...

“What Is Visualization Filtering?” on page 5-15

“Changing Component Visibility” on page 5-15

“Visualization Filtering Options” on page 5-16

“Components You Can Filter” on page 5-16

“Model Hierarchy and Tree Nodes” on page 5-17

“Filtering Hierarchical Subsystems” on page 5-17

“Updating Models with Hidden Nodes” on page 5-18

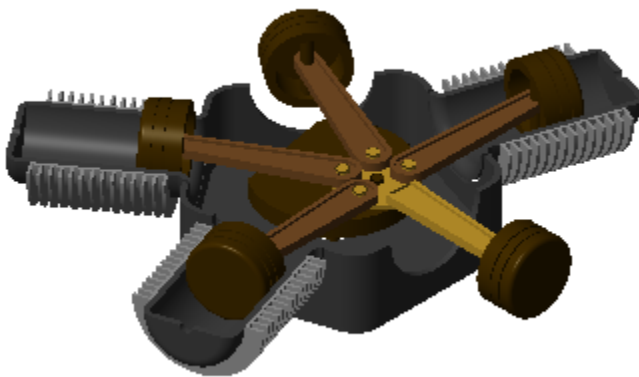
“Alternative Ways to Enhance Visibility” on page 5-18

What Is Visualization Filtering?

A multibody model can get so complex that you cannot easily tell its components apart. Solids, bodies, and multibody subsystems often hide behind each other, hindering your efforts to examine geometry, pose, and motion on model update or during simulation.

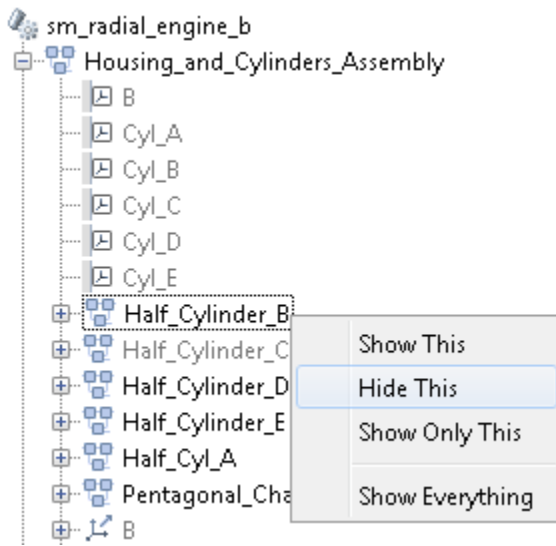
Visualization filtering is a Mechanics Explorer feature that lets you selectively show and hide parts of your model. By showing only those parts that you want to see, you can more easily discern any components placed within or behind other components—such as an engine piston traveling inside a cylinder casing.

The figure shows an example of visualization filtering. Two cylinders, one at the front and one at the rear, are hidden in the model visualization of the `sm_radial_engine` featured example. For a tutorial showing how to use visualization filtering, see “Selectively Show and Hide Model Components” on page 5-20.



Changing Component Visibility

You can show and hide components through a context-sensitive menu accessible in the tree-view pane of Mechanics Explorer. Right-click a model-tree node to open the menu and select the desired option. The figure shows the visualization filtering menu.



Visualization Filtering Options

The visualization filtering menu provides four options for you to select from:

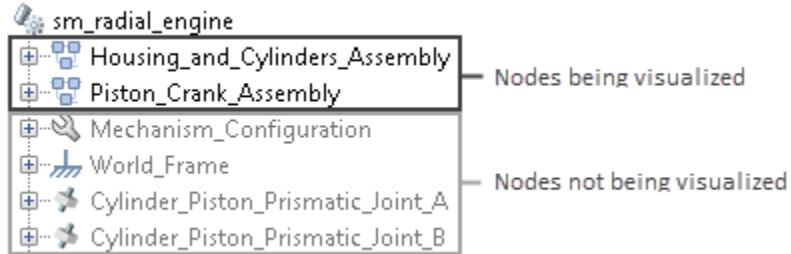
- **Show This** — Enable visualization for the selected component. This option has no effect if the component is already visible.
- **Hide This** — Disable visualization for the selected component. This option has no effect if the component is already hidden.
- **Show Only This** — Enable visualization for the selected component and disable visualization for the remainder of the model. This option has no effect if the selected component is already the only component visible.
- **Show Everything** — Enable visualization for every component in the model. This option has no effect if every component in the model is already visible.

Components You Can Filter

You can filter the visualization of any component with solid geometry. This includes individual solids, bodies, and multibody subsystems. In general, if a subsystem contains at least one Solid block, then you can switch its visualization on and off.

Frames, joints, constraints, forces, and torques have no solid geometry to visualize and therefore cannot be filtered in Mechanics Explorer. You can still open the visualization filtering context-sensitive menu by right-clicking these nodes, but only one option is active—**Show Everything**.

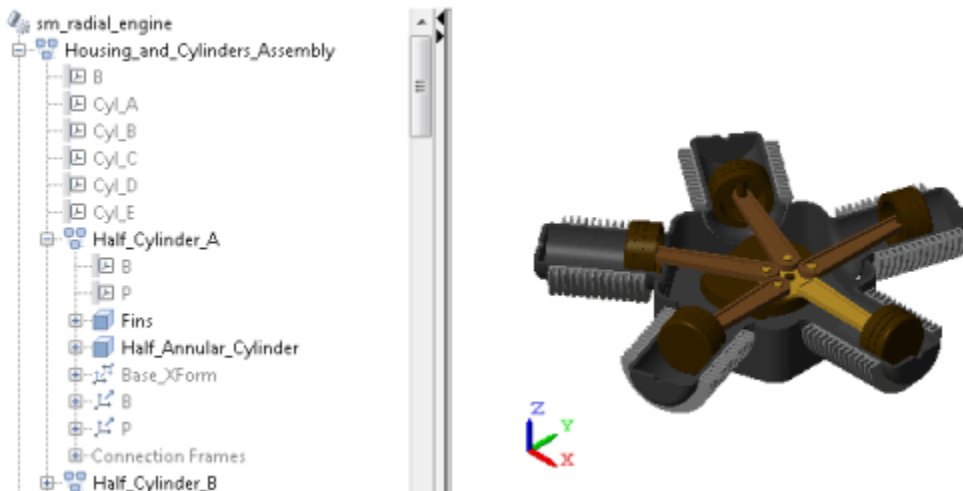
The tree-view pane identifies any node not being visualized by graying out its name. This includes nodes that can be visualized but are currently hidden and nodes that cannot be visualized at all. The figure shows an example with the grayed-out names of nodes not being visualized.



Model Hierarchy and Tree Nodes

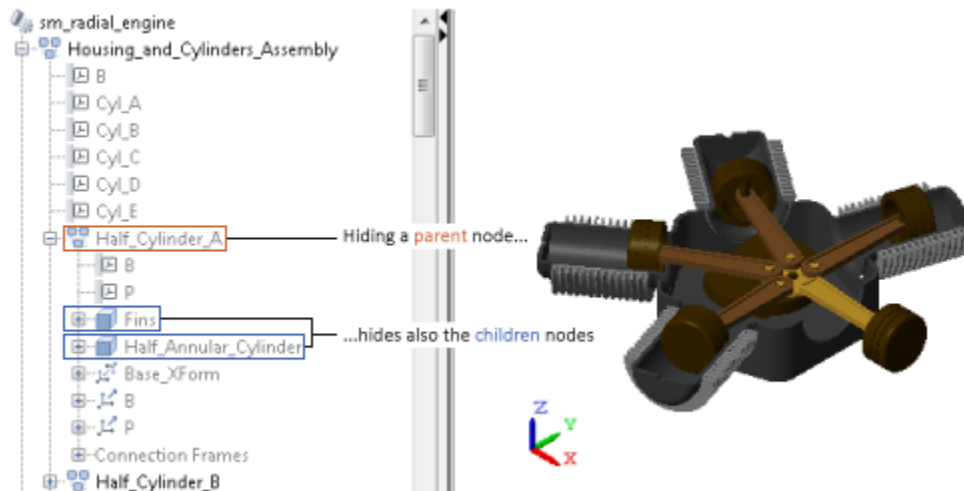
Multibody models are hierarchical in nature. They often contain multibody subsystems comprising body subsystems, each with one or more solids. The tree-view pane of Mechanics Explorer represents such a model structure through nodes arranged hierarchically. A node is a parent node if it contains other nodes, and a child node if it appears under another node. Nodes can simultaneously be children to some nodes and parents to others.

The figure shows portion of the tree-view pane of the `sm_radial_engine` featured example. The `Half_Cylinder_A` node is a child to the `Housing_and_Cylinders_Assembly` node and a parent to the `Fins` and `Half_Annular_Cylinder` nodes.



Filtering Hierarchical Subsystems


Any changes to the visualization settings of a tree node apply equally to all children of that node, if any. Nodes higher up in the model tree are not affected. As shown in the following figure, hiding the `Half_Cylinder_A` node in the `sm_radial_engine` model causes the `Fins` and `Half_Annular_Cylinder` nodes (children nodes) to hide, but not the `Housing_and_Cylinders_Assembly` node (parent node) or the `Half_Cylinder_B` node (sibling node).



If you want to show part of a subsystem you have previously hidden, you can change the visibility settings for the children nodes that you want to show. For example, if after hiding the `Half_Cylinder_A` node, you want to show the `Fins` child node, you need only right-click the `Fins` node and select `Show This`. Such changes have no effect on the remainder of the `Half_Cylinder_A` parent node.

Updating Models with Hidden Nodes

The following apply when you update or simulate a model with previously hidden nodes:

- If the model remains unchanged, the node visibility settings remain unchanged—that is, the hidden nodes remain hidden and the visible nodes remain visible. This happens even if you save the Mechanics Explorer configuration to the model by clicking the  icon.
- If you close Mechanics Explorer before updating the model, Mechanics Explorer reopens with all nodes visible, including any nodes you may have previously hidden.
- If you change the name of a block corresponding to a hidden node—e.g., a Solid block or a Subsystem block containing a Solid block—the hidden node and any children nodes it may have become visible.
- If you uncomment a block that corresponds to a hidden node and that you had previously commented out, the hidden node and any children nodes it may have become visible.
- If you add to a hidden Subsystem block a Solid block or another Subsystem block with a Solid block, the child node corresponding to the new block becomes visible upon model update but the visibility of the hidden parent node remains unchanged.
- If you change the parameters of a block corresponding to a hidden node, that node and its children nodes retain their original visibility settings—that is, hidden nodes remain hidden and visible nodes remain visible.

Alternative Ways to Enhance Visibility

Visualization filtering is not the only approach you can use to enhance component visibility in a model. However, it is often the simplest. It is also the only approach that doesn't require you to modify the model in any way. Alternative approaches you can use include:

- Lowering the opacity of obstructive components—those obscuring other parts of the model—for example, making the cylinder encasing an engine piston transparent.
- Modeling obstructive components only in part—for example, treating engine cylinders as half-cylinders to preserve piston visibility during simulation.
- Omitting obstructive components altogether if they serve a purely aesthetic purpose and have no impact on model dynamics—for example, removing the cylinder subsystems from the `sm_radial_engine` featured example.
- Commenting out or through obstructive components if they serve a purely aesthetic purpose and have no impact on model dynamics—for example, removing the cylinder subsystems from the `sm_radial_engine` featured example.

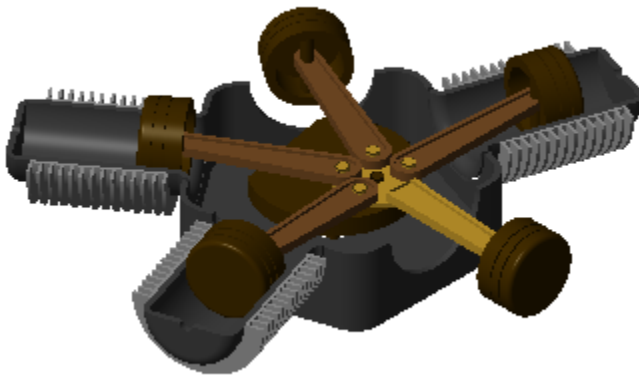
Selectively Show and Hide Model Components

In this section...

“Visualization Filtering” on page 5-20
“Open Example Model” on page 5-20
“Update Example Model” on page 5-21
“Hide Half-Cylinder Subsystem” on page 5-21
“Show Solid in Hidden Subsystem” on page 5-22
“Show Only Piston Subsystem” on page 5-22
“Show Everything” on page 5-23

Visualization Filtering

Visualization filtering is a Mechanics Explorer feature that enables you to selectively show and hide solids, bodies, and multibody subsystems. This tutorial shows you how to use this feature to control the visualization of a Simscape Multibody model, for example, to observe a model component that might otherwise remain obstructed during simulation. For more information, see “Selective Model Visualization” on page 5-15.

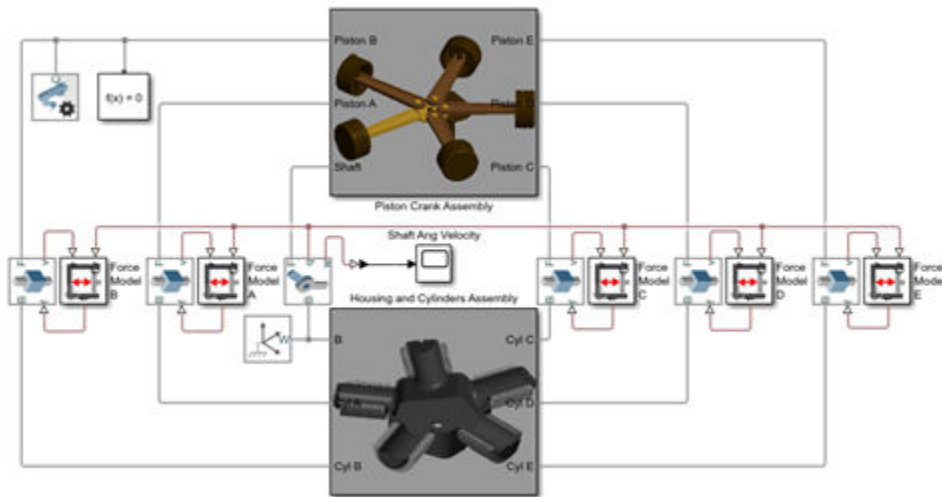


Radial Engine Visualization with Two Cylinders Hidden

Open Example Model

In this tutorial, you filter the visualization of the Simscape Multibody radial engine featured example. To open this model, at the MATLAB command prompt, enter `sm_radial_engine`.

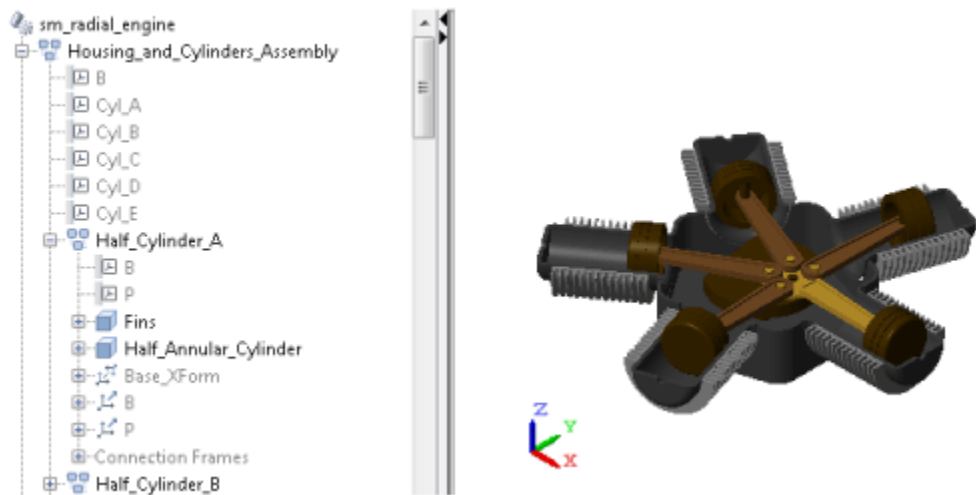
The model contains two top-level subsystems—the housing subsystem, named `Housing_and_Cylinders_Assembly`, and the piston subsystem, named `Piston_Crank_Assembly`. The housing subsystem contains five half cylinders. The piston subsystem contains five pistons that travel inside the half cylinders.



Radial Engine Block Diagram

Update Example Model

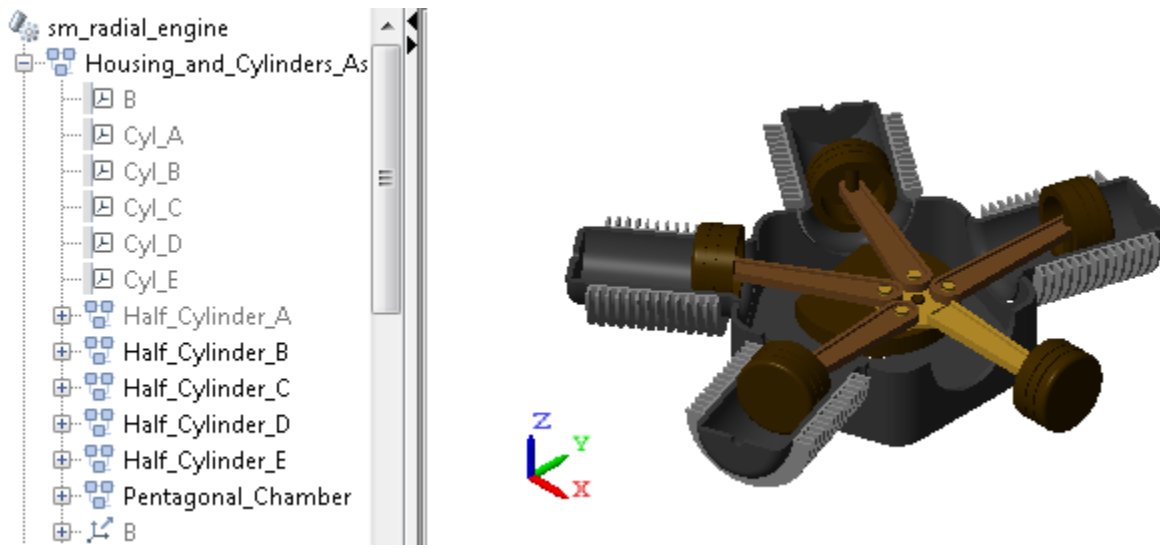
To open Mechanics Explorer, the Simscape Multibody visualization utility you must first update the example model. In the **Modeling** tab, click **Update Model (Ctrl + D)**. Note the tree-view pane on the left side of Mechanics Explorer. You access the visualization filtering menu by right-clicking a node on this pane.



Radial Engine Model Visualization

Hide Half-Cylinder Subsystem

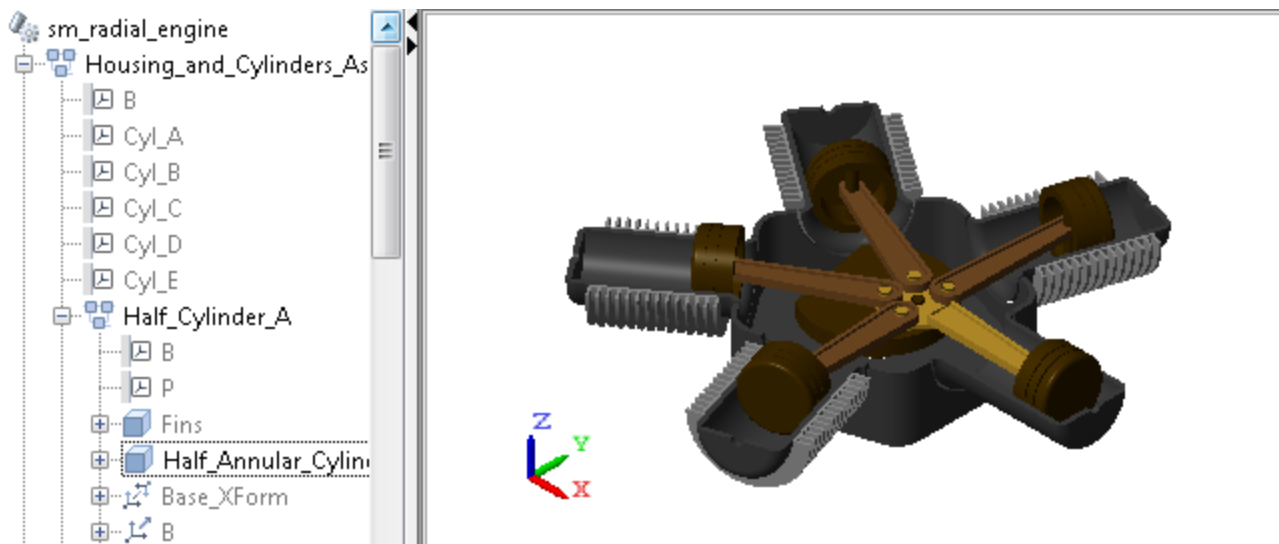
In the tree-view pane, expand the `Housing_and_Cylinders_Assembly` node. Right-click the `Half_Cylinder_A` node and select **Hide This**. Mechanics Explorer hides the half-cylinder subsystem and the solids it contains, corresponding to the nodes `Fins` and `Half_Annular_Cylinder`. The hidden-node names are grayed out in the tree-view pane. The figure shows the resulting model visualization.



Radial Engine with Hidden Half-Cylinder Subsystem

Show Solid in Hidden Subsystem

In the tree-view pane, expand the `Half_Cylinder_A` node. Then, right-click the `Half_Annular_Cylinder` node and select `Show This`. The half-cylinder solid is now visible, but the remainder of its parent of its parent subsystem—in this case, just the `Fins` solid—remains hidden. The newly visible half-cylinder node name is no longer grayed out in the tree-view pane. The figure shows the resulting model visualization.

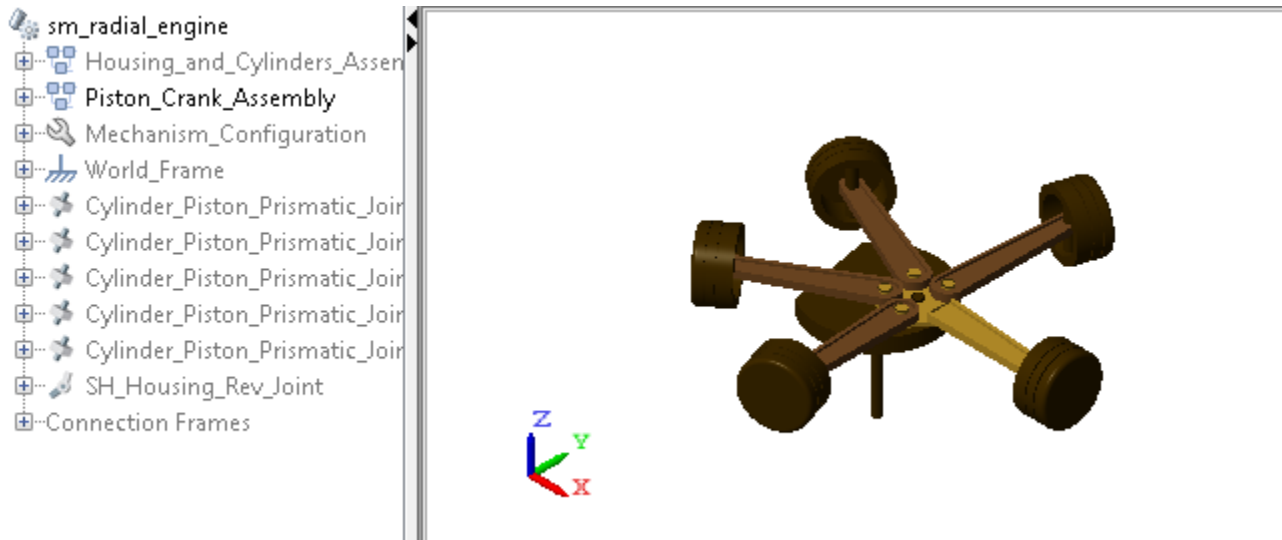


Radial Engine with Visible Solid in Hidden Half-Cylinder Subsystem

Show Only Piston Subsystem

In the tree-view pane, collapse the `Housing_and_Cylinders_Assembly` node. Then, right-click the `Piston_Crank_Assembly` node and select `Show Only This`. Mechanics Explorer shows the selected

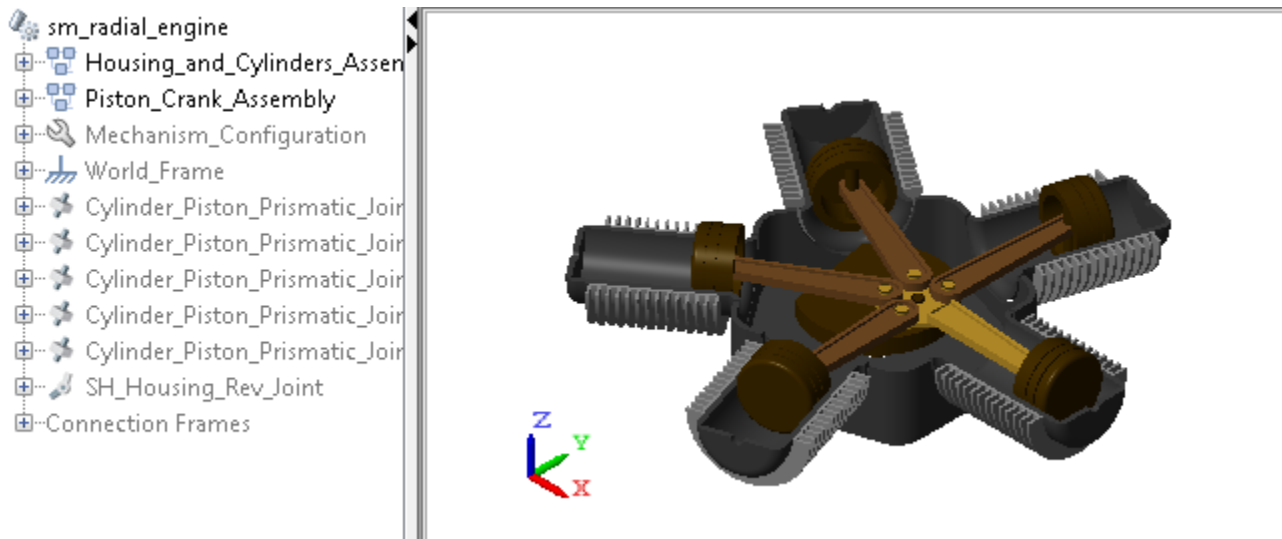
node and hides the remainder of the model. In the tree-view pane, the name of the selected node is the only that is not grayed out. The figure shows the resulting model visualization.



Radial Engine with Only Piston Subsystem Visible

Show Everything

In the tree-view pane, right-click any node and select Show Everything. All hidden components become visible. The corresponding nodes are no longer grayed out in the tree-view pane. The figure shows the resulting model visualization.



Visualize Simscape Multibody Frames

In this section...

“What Are Frames?” on page 5-24

“Show All Frames” on page 5-24

“Highlight Specific Frames” on page 5-25

“Visualize Frames via Graphical Markers” on page 5-26

What Are Frames?

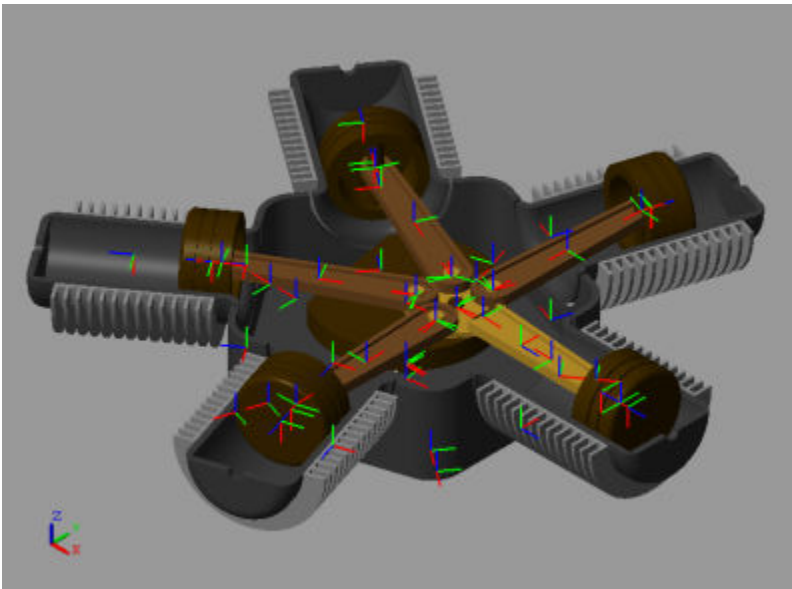
Simscape Multibody models are based on frames, abstract axis triads that contain all the position and orientation data in a model. These constructs enable you to connect solids into bodies, assemble bodies into mechanisms, and prescribe and sense forces, torques, and motion. Given their importance, then, it makes sense to visualize where and how you place different frames in a model.

Show All Frames

The easiest way to view the frames in your model is to toggle their visibility on. You can do this by clicking the **Toggle visibility of frames** icon in the Mechanics Explorer tool strip, shown in the following figure.



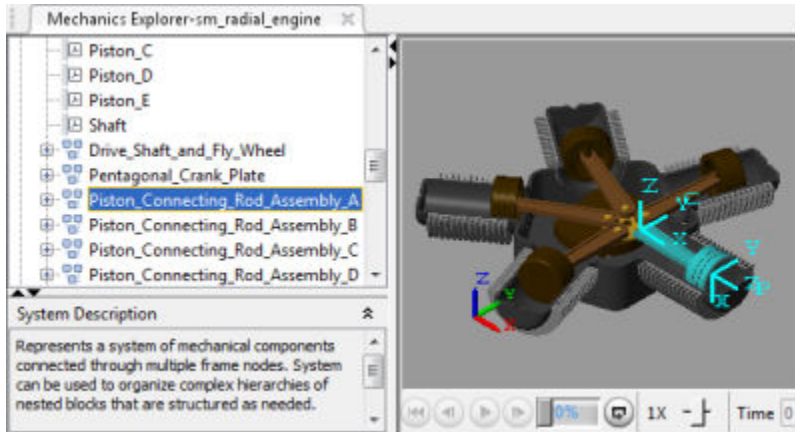
Alternatively, you can select **View > Show Frames** in the menu bar. Mechanics Explorer shows all the frames in your model, suiting this approach well for models with small numbers of frames. The figure shows a radial engine model with frame visibility toggled on.



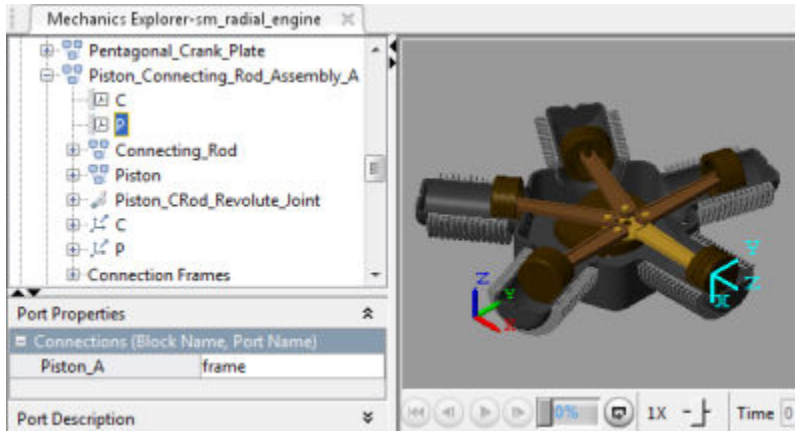
If your model has many frames, a different approach may be ideal, as toggling frame visibility may clutter the visualization pane with frames that you don't want to track.

Highlight Specific Frames

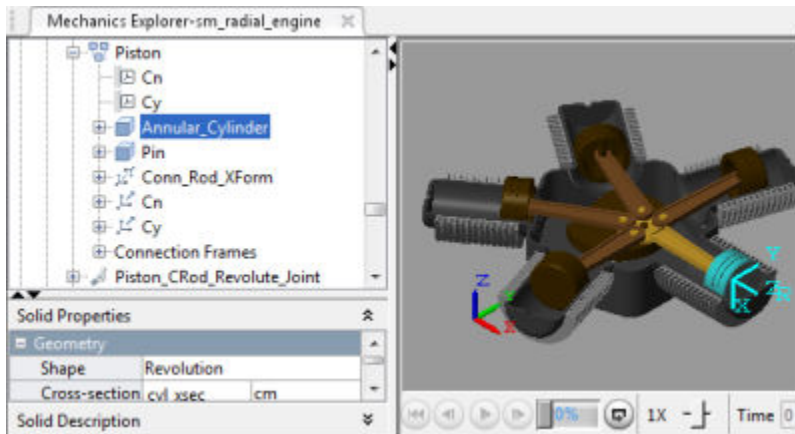
To view only the port frames of a block, including those of a subsystem block, you can select a node in the tree view pane. Mechanics Explorer highlights the port frames associated with the selected node using a turquoise color. The following figure shows an example in which one of the connecting rod assemblies in the radial engine model is highlighted in turquoise.



You can also select individual port frames, which you expose by expanding the tree nodes. For example, expanding the Piston_Connecting_Rod_Assembly_A node exposes the port frame P node, which you can then select in order to highlight that frame. The figure shows the result.

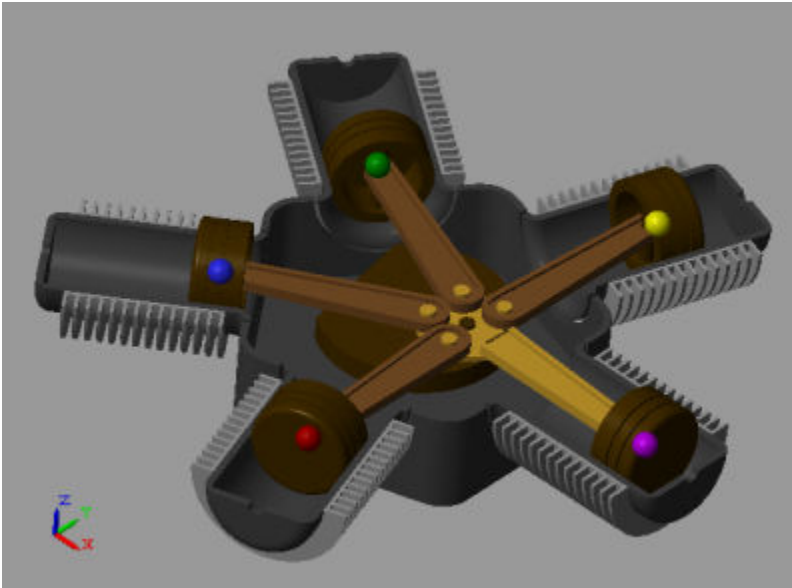


Finally, you can select individual solids directly in the visualization pane, highlighting their reference frames. The figure shows the result of selecting one of the piston solids directly. Mechanics Explorer highlights the solid and its reference frame, while the tree view pane reveals the associated Solid block name. This is the block that you need to change if you want to modify this particular solid.



Visualize Frames via Graphical Markers

If a frame in your model has special significance—e.g., if its origin is the point of application for an external force—you can connect to it a graphical marker. So that you can perform this task, the Body Elements library provides a Graphic block. Simply connect the block to the frame you want to visualize and select the marker type to use—sphere, cube, or frame. The figure shows the radial engine model with a sphere marker highlighting each of the piston connection frames.



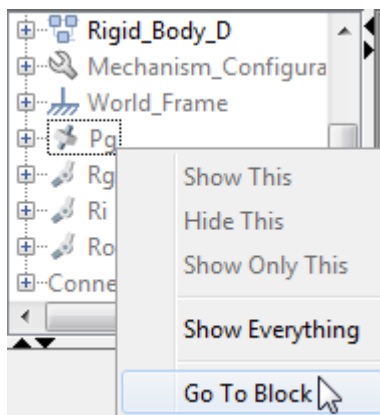
Go to a Block from Mechanics Explorer

The first indication that something is wrong in a model is often an unexpected result in the visualization pane. Unexpected results can include disparities in solid shape and size, incorrect translation and rotation transforms between solids, and even joints and constraints that fail to assemble.

To help you troubleshoot such modeling issues, Mechanics Explorer enables you to go directly to a block associated with a node in the tree view pane. This feature helps you also to iterate on a model that is working properly, for example, if you want to replace a body subsystem with an alternative version.

To highlight a block corresponding to a Mechanics Explorer tree node:

- 1 In the tree view pane of Mechanics Explorer, right-click the node whose block you want to examine.



- 2 From the context-sensitive menu, select **Go to Block**. Simscape Multibody brings the block diagram to the front and highlights the block corresponding to the selected node.



For an example showing how to troubleshoot a model using Mechanics Explorer block highlighting, see “Troubleshoot an Assembly Error” on page 2-21.

Create a Model Animation Video

In this section...

“UI and Command-Line Tools” on page 5-28

“Before Creating a Video” on page 5-28

“Create a Video Using Video Creator” on page 5-28

“Create a Video Using smwritevideo” on page 5-29

UI and Command-Line Tools

You can create a model animation video interactively, using the **Video Creator** tool, or programmatically, using the `smwritevideo` function. The tool and function provide equivalent ways to perform the same task. Use the tool to more intuitively configure and create a video. Use the function for your command-line workflows, e.g., to automate video capture following model simulation.

Before Creating a Video

- Mechanics Explorer must be set to open on model update.

You can view and change the current setting in the **Simscape Multibody > Visualization** tab of the Model Configuration Parameters window.

- Only the active visualization tile in Mechanics Explorer is recorded.

A visualization tile is a subdivision of the Mechanics Explorer visualization pane that shows a specific view of the model. The active tile is demarcated by a colored bounding box.

- The video viewpoint is always that of the active tile.

To change the model viewpoint in the recorded video, you must change the viewpoint of the active tile. Use a dynamic camera for a moving point or the global camera for a static viewpoint.

Create a Video Using Video Creator

- 1 Simulate the model to record.

Animations and the videos generated from them are based on your model simulation data.

- 2 In Mechanics Explorer, select **Tools > Video Creator**.

Video Creator relies on your model visualization and is accessible through Mechanics Explorer only.

- 3 In Video Creator, specify the desired video parameters.

Video parameters that you can modify include the video frame rate, frame size, playback speed ratio, and video file format.

- 4 Click the **Create** button.

Video Creator generates a model animation video and saves it with the specified name in the specified folder.

- 5 Save the model animation video in a folder for which you have write privileges.

Create a Video Using `smwritevideo`

- 1 Simulate the model to record. E.g.,

```
modelName = 'sm_dump_trailer';  
sim(modelName);
```

Simscape Multibody software simulates the dump trailer featured example.

- 2 Define any video parameters that deviate from your current Video Creator settings. E.g.,

```
fps = 60; speedRatio = 2;
```

All remaining video parameters are configured as specified in Video Creator.

- 3 Define the desired name and path of the animation video.

```
videoName = 'dump_trailer_animation'
```

In the absence of a path, the function saves the video in the current folder. You must have write privileges to the folder in order to save the video.

- 4 Call the `smwritevideo` function with the video name and parameters as function arguments.

```
smwritevideo(modelName,videoName,...  
'PlaybackSpeedRatio',speedRatio,'FrameRate',fps);
```

The `smwritevideo` function creates a video of the model animation and saves it with the name `dump_trailer_animation` in the current folder.

Multibody Model Import

CAD and URDF Model Import

- “CAD Translation” on page 6-2
- “Install the Simscape Multibody Link Plugin” on page 6-9
- “Import a CAD Assembly Model” on page 6-12
- “Import a Robotic Arm CAD Model” on page 6-14
- “Onshape Import” on page 6-17
- “Import an Onshape Humanoid Model” on page 6-22
- “URDF Primer” on page 6-25
- “Import URDF Models” on page 6-33
- “Import a URDF Humanoid Model” on page 6-40

CAD Translation

In this section...

“Translating a CAD Model” on page 6-2

“What’s in a Translated Model?” on page 6-2

“What’s in a Data File?” on page 6-5

“Exporting a CAD Model” on page 6-5

“Importing a CAD Model” on page 6-6

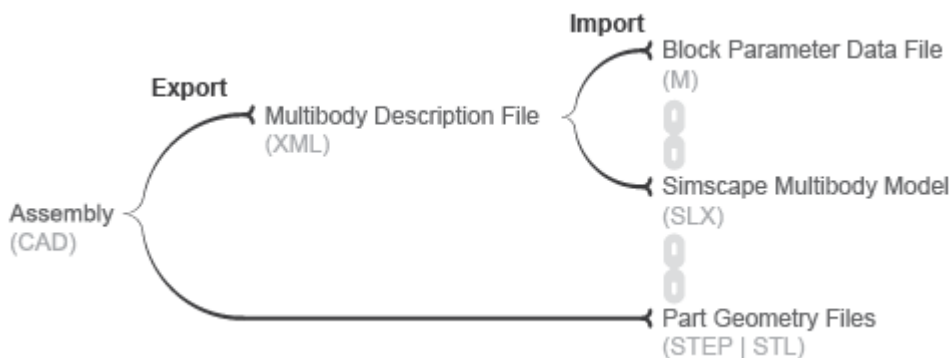
“Simplifying Model Topology” on page 6-7

“Updating an Existing Data File” on page 6-8

Translating a CAD Model

You can translate a CAD model into an equivalent Simscape Multibody block diagram. The conversion relies on the `smimport` function featuring an XML multibody description file name as its central argument. The XML file passes to Simscape Multibody software the data it needs to recreate the original model—or an approximation of it if unsupported constraints exist in the model.

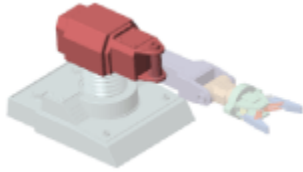
You translate a CAD model in two steps—export and import. The export step converts the CAD assembly model into an XML multibody description file and a set of STEP or STL part geometry files. The import step converts the multibody description and part geometry files into an SLX Simscape Multibody model and an M data file. The model obtains all block parameter inputs from the data file.



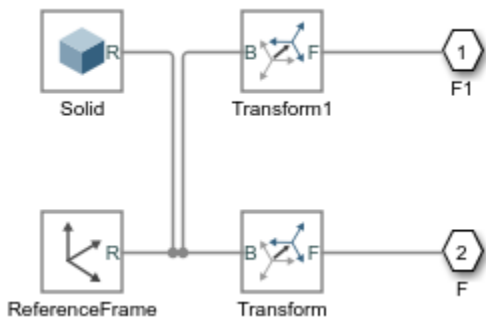
CAD Translation Steps

What’s in a Translated Model?

The translated model represents the CAD parts—referred to as bodies in Simscape Multibody software—using Simulink subsystems that comprise multiple solid and Rigid Transform blocks. The solid blocks provide the body geometries, inertias, and colors. The Rigid Transform blocks provide the frames with the required poses for connection between bodies.

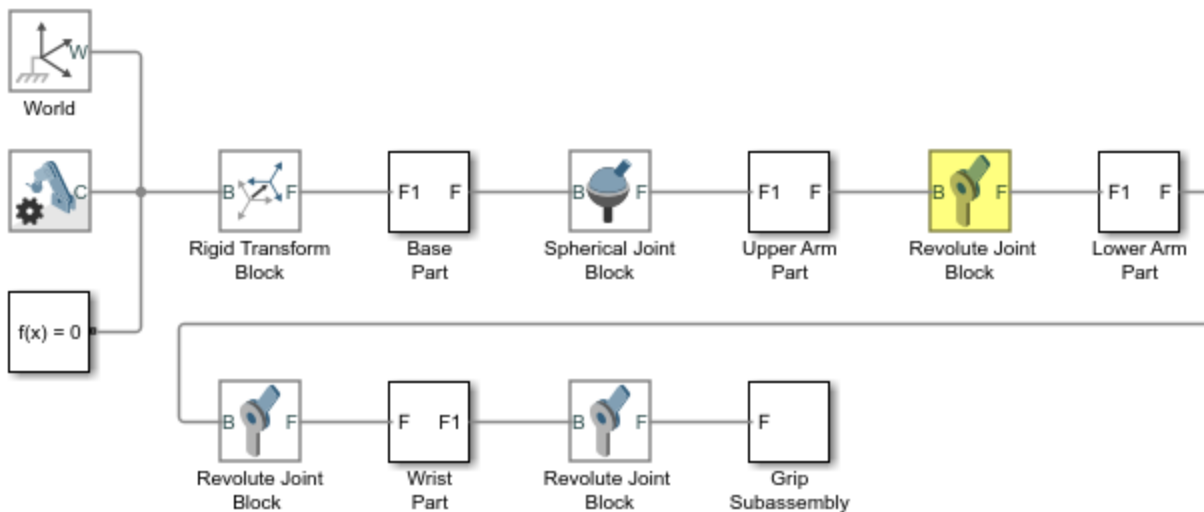


Consider the upper arm body of a CAD robotic arm model, shown in the figure. The Simulink subsystem for this body consists of one solid block connected to a pair of Rigid Transform blocks. The Solid block provides the reference to the upper arm geometry file and the inertial properties derived from the CAD model. The Rigid Transform blocks provide the frames for connection to the robot base and lower arm bodies.



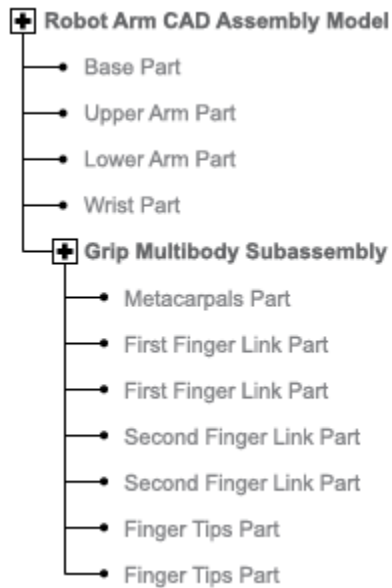
Simulink Subsystem Representing Upper Arm Body

CAD joints, constraints, and mates translate into Simscape Multibody software as combinations of joint and constraint blocks. In the CAD robotic arm example, the constraints between the upper arm and the lower arm translate into a Revolute Joint block. This block sits between the Simulink Subsystem blocks that represent the upper arm and lower arm bodies.



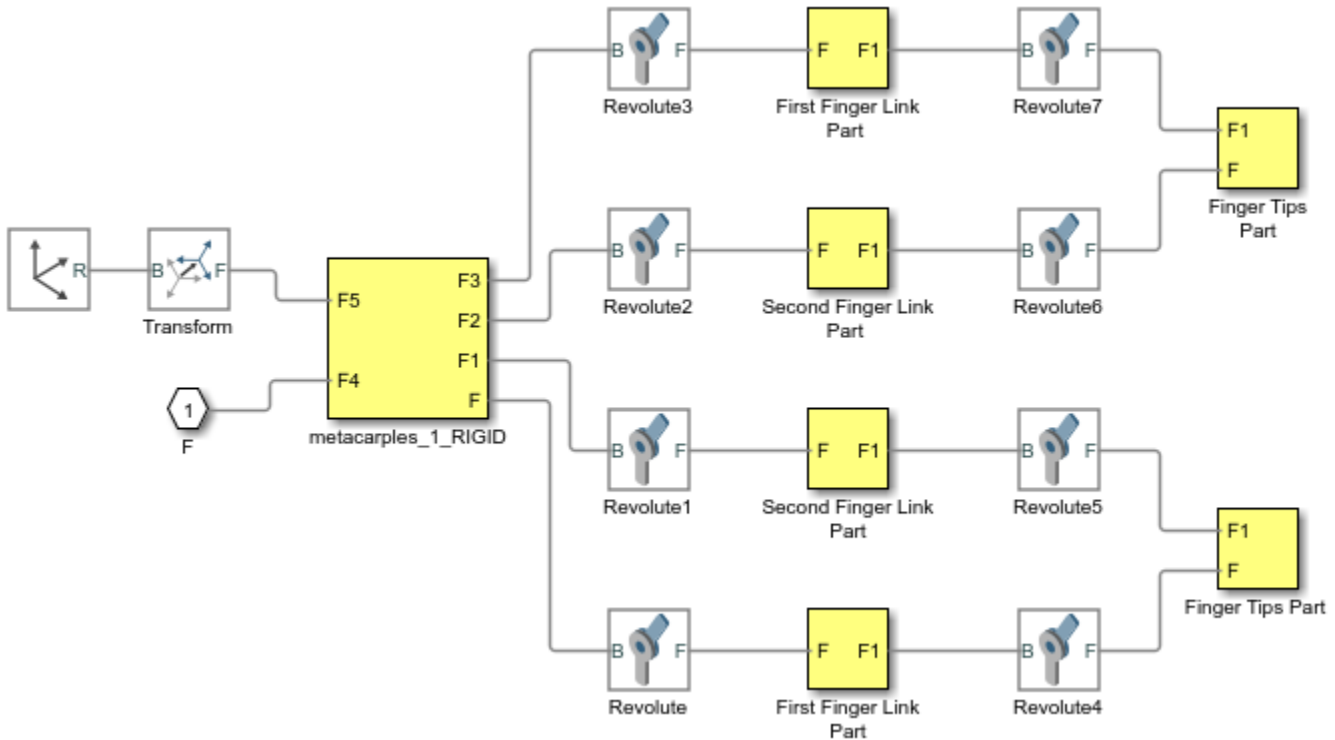
Simulink Subsystem Representing Upper Arm Body

By default, the translated model preserves the structural hierarchy of the original CAD model. If the source model is a CAD model with multibody subassemblies, the subassemblies convert in Simscape Multibody software into multibody Simulink subsystems. Consider again the CAD robotic arm model. The model contains a grip multibody subassembly with seven bodies, shown schematically in the figure.



CAD Robotic Arm Model Hierarchy

During translation, the grip subassembly converts into a Simulink subsystem with seven Simulink subsystems, one for each body.



Multibody Simulink Subsystem with Body Simulink Subsystems

What's in a Data File?

Blocks in the translated model are parameterized in terms of MATLAB variables defined in the data file. These variables are stored in structure arrays named after the various block types. The structure arrays are nested in a parent data structure named `smiData` or a custom string that you specify.

Consider an imported model with a data structure named `smiData`. If the model contains Revolute Joint blocks, the parameter data for these blocks is the structure array `smiData.RevoluteJoint`. This structure array contains a number of data fields, each corresponding to a different block parameter.

The structure array fields are named after the block parameters. For example, the position state target data for the Revolute Joint blocks is in a field named `Rz_Position_Target`. If the model has two Revolute Joint blocks, this field contains two entries—`smiData.RevoluteJoint(1).Rz_Position_Target` and `smiData.RevoluteJoint(2).Rz_Position_target`.

Each structure array index corresponds to a specific block in the imported model. The index assignments can change if you regenerate a data file from an updated XML multibody description file. The `smiimport` function checks the prior data file, when specified, to ensure the index assignments remain the same. See “Updating an Existing Data File” on page 6-8.

Exporting a CAD Model

You can technically export a CAD assembly model from any CAD application. The Simscape Multibody Link CAD plug-in provides one means to export a model in a valid XML format. The plug-in is

compatible with three desktop CAD applications: SolidWorks®, PTC® Creo™, and Autodesk® Inventor®. The plug-in generates not only the XML multibody description file but also any geometry files required for visualization in the final translated model.

The Simscape Multibody `smexportonshape` function provides another means to export a CAD model, from a cloud application named Onshape®. This function exports in a format compatible with the newer Simscape Multibody blocks only. See “Onshape Import” on page 6-17 for more information on exporting (and importing) CAD models from an Onshape account.

If you use an unsupported CAD application, you can create a program that uses the CAD API and Simscape Multibody XML schema to generate the multibody description and part geometry files. This task requires knowledge of XML documents, XSD schema definitions, and CAD APIs. See the schema website for the XSD schema definitions. See MATLAB Central for an example program built on the SolidWorks CAD API.

If a URDF converter exists for your CAD application, you may be able to export your model in URDF format and import the URDF file into the Simscape Multibody environment. Note, however, that the URDF specification forbids closed-chain model topologies, such as those of four-bar linkages and gear assemblies. For more information, see “Import URDF Models” on page 6-33.

A Note About Export Errors

If the Simscape Multibody Link plug-in cannot export a part geometry file or translate a CAD constraint set, the software issues an error message. The error message identifies the bodies with missing geometry files and any unsupported constraints. You can import the generated XML multibody description file into Simscape Multibody software, but the resulting model may not accurately represent the original CAD assembly model.

Importing a CAD Model

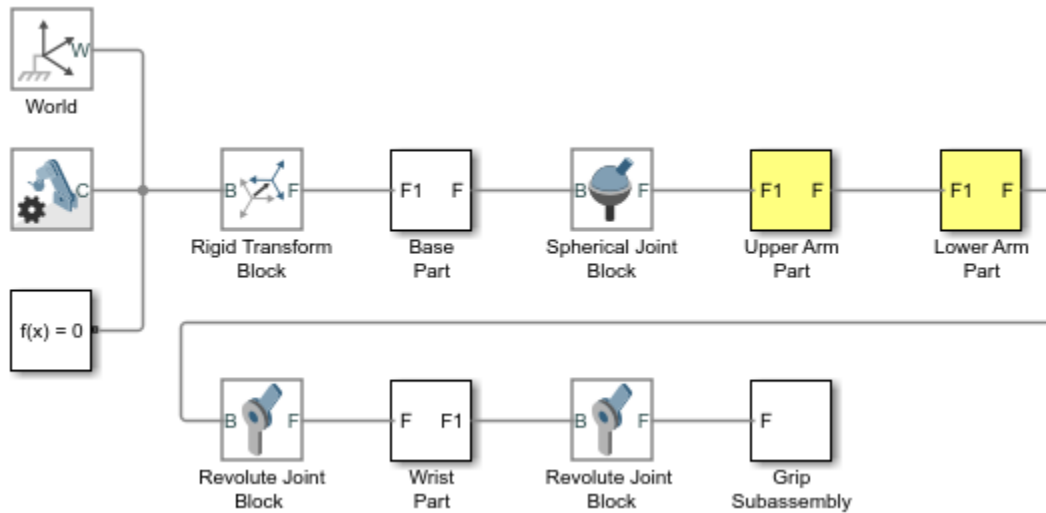
You import an XML multibody description file using the Simscape Multibody `smimport` function in its default import mode. The function parses the file and generates a Simscape Multibody model and associated data file. For step-by-step instructions on to import a CAD assembly model via its XML multibody description file, see “Import a CAD Assembly Model” on page 6-12.

Note Starting with software version R2017b, the Simscape Multibody Link plug-in exports in an XML format compatible only with Simscape Multibody Second Generation software. You must import all such XML files using the `smimport` function. Models generated with this function comprise only second-generation blocks—those accessible by entering the command `sm_lib` at the MATLAB command prompt.

The Simscape Multibody First Generation software is not supported anymore and the `mech_import` function can not be used anymore.

CAD Import Errors

If a part geometry file is invalid or missing, the corresponding body cannot show in the Simscape Multibody visualization utility. If a CAD assembly model contains an unsupported constraint combination between bodies, Simscape Multibody software joins the bodies with a rigid connection. The rigid connection can take the form of a direct frame connection line, Rigid Transform block, or Weld Joint block.



Rigid Connection Due to Unsupported Constraints

If Simscape Multibody software cannot translate a CAD constraint combination, it issues a warning message on the MATLAB command window identifying the affected bodies and their connection frames. For example:

Warning: The set of constraints between upperarm_1_RIGID and forearm_1_RIGID could not be mapped to a joint. A rigid connection has been added between port F of upperarm_1_RIGID and port F1 or forearm_1_RIGID for these constraints.

Simplifying Model Topology

You can import a CAD model with a simplified topology. So that you can do this, the `smimport` function provides the `ModelSimplification` argument. You can set this argument to:

- `bringJointsToTop` to group each set of rigidly connected parts into a new subsystem and promote all joints to the top level of the model hierarchy.
- `groupRigidBodies` to group rigidly connected parts into subsystems (and leave joints in their original places in the model hierarchy).
- `None` to import the model as is, without simplification.

Use the `bringJointsToTop` or `groupRigidBodies` option if your CAD model has many rigidly connected components, such as nuts and bolts, that you prefer to group together—for example, to more intuitively grasp the key components of the model at a glance of the block diagram.

Use the `bringJointsToTop` option if your CAD model has joints inside subassemblies and you prefer to expose them at the top level—for example, to work with joint actuation and sensing signals without having to search for the joints inside different subsystems.

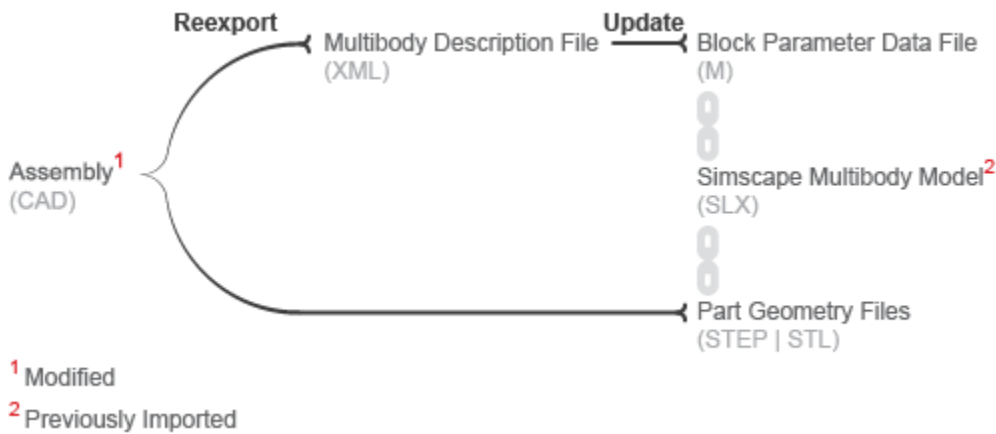
Note that model simplification is available for CAD import only. URDF models have flat topologies with little need for simplification.

Updating an Existing Data File

You regenerate the data file for a previously imported model by running the `smimport` function in `dataFile` mode. You specify this mode using the optional `ImportMode Name, Value` pair argument. The function uses the prior data file to keep the mapping between structure array indices and blocks consistent.

Before regenerating a data file, you must export a new XML multibody description file from the updated CAD assembly model. The `smimport` function uses the data in the new multibody description file to generate the new data file.

The function does not update the block diagram when run in `dataFile` mode. If you add or delete bodies in the source CAD assembly model, you must manually add or delete the corresponding blocks in the previously imported model.



CAD Update

Install the Simscape Multibody Link Plugin

In this section...
“Software Requirements” on page 6-9
“Step 1: Get the Installation Files” on page 6-9
“Step 2: Run the Installation Function” on page 6-10
“Step 3: Register MATLAB as an Automation Server” on page 6-10
“Step 4: Enable the Simscape Multibody Link Plugin in a CAD Application” on page 6-10
“Importing CAD Files from Applications Not Supported by Simscape Multibody Link” on page 6-10

Simscape Multibody Link is a plugin that you can install on CAD applications to export assembly models to Simscape Multibody. Specifically, the plugin exports a CAD assembly model as an XML file and body geometry files that you can convert into Simscape Multibody models using the `smimport` function. The plugin supports:

- SolidWorks
- Autodesk Inventor®
- PTCCreo

Software Requirements

Make sure your software meets the requirements:

CAD Software	Release Supported	MATLAB Release	Simscape Multibody Release
SolidWorks	2001Plus and higher	R2008b and higher	3.0 and higher
PTCCreo	1.0 and higher	R2008b and higher	3.0 and higher
Autodesk Inventor	2009 to 2021	R2009b and higher	3.0 and higher

Note Autodesk Inventor 2022 is not supported.

Your MATLAB and CAD installations must have the same system architecture, such as Windows 64-bit.

Step 1: Get the Installation Files

- 1 Go to the Simscape Multibody Link download page.
- 2 Follow the prompts on the download page.
- 3 Select and save the ZIP and MATLAB files that match your MATLAB version and system architecture, such as release R2020b and Win64 (PC) Platform. Do not extract the ZIP file.

Step 2: Run the Installation Function

- 1 Run MATLAB as administrator. For more information, see [How to Make a Shortcut to Run MATLAB as Administrator](#).
- 2 Add the folder in which you saved the installation files to the MATLAB path. For example, you can use the `addpath` function.
- 3 At the MATLAB command prompt, enter `install_addon('zipname')`, where `zipname` is the name of the ZIP file, such as `smlink.r2020b.win64.zip`.

Step 3: Register MATLAB as an Automation Server

Each time you export a CAD assembly model, the Simscape Multibody Link plugin attempts to connect to MATLAB. To enable the connection, you must register MATLAB as an automation server. You can do this in two ways:

- Open a MATLAB session in administrator mode. At the MATLAB command prompt, enter `regmatlabserver`.
- Open a Windows® command prompt window running in administrator mode. At the command prompt, enter `matlab -regserver`.

Step 4: Enable the Simscape Multibody Link Plugin in a CAD Application

Before you can export a CAD assembly, you must enable the Simscape Multibody Link plugin on your CAD application. To do this, see:

- “Enable Simscape Multibody Link Plugin in SolidWorks”
- “Enable Simscape Multibody Link Plugin in Creo-Pro/E”
- “Enable Simscape Multibody Link Plugin in Inventor Plugin”

Importing CAD Files from Applications Not Supported by Simscape Multibody Link

- To import an Onshape CAD assembly model into the Simscape Multibody, you can use the `smexportonshape` and `smimport` functions.
- If you use a CAD application other than Onshape, SolidWorks, PTC Creo, and Autodesk Inventor, you can create a custom model export application based on the Simscape Multibody XML schema. This approach requires some knowledge of XML. See the [schema web page](#) for more information.
- You can also create a Simscape Multibody model from a URDF file or Robotics System Toolbox™ model. See the `smimport` for more information.

See Also

`smexportonshape` | `smimport`

More About

- “Import URDF Models” on page 6-33

- “Import a URDF Humanoid Model” on page 6-40

Import a CAD Assembly Model

In this section...

“Before You Begin” on page 6-12
 “Example Files” on page 6-12
 “Import a Model” on page 6-12
 “After Model Import” on page 6-13

Before You Begin

You import a CAD model into Simscape Multibody software using the `smimport` function. The function parses an XML multibody description file and automatically generates the corresponding model. Geometry files accompanying the multibody description file format provide the body geometries for visualization purposes. You can import models of assemblies but not parts.

Example Files

You can try the CAD import workflow using the examples in your Simscape Multibody installation. The examples include a four-bar linkage, a robotic arm, and a Stewart platform. Each example comprises an XML multibody description file and a set of part geometry files. The files are in folders with path

```
matlabroot\toolbox\physmod\sm\smdemos\import\modelFolder,
```

where:

- *matlabroot* is the root folder of your MATLAB application, for example:
 C:\Programs\MATLAB\
- *modelFolder* is the name of the folder that contains the example file sets—`four_bar`, `robot`, or `stewart_platform`.

Import a Model

You import a model into Simscape Multibody software using the `smimport` function in its default mode. Consider the example file sets in your Simscape Multibody installation. To recreate the CAD assembly model described by the files as Simscape Multibody block diagrams, enter:

```
smimport(multibodyDescriptionFile);
```

where *multibodyDescriptionFile* is the XML multibody description file name for the example model you want to import, specified as a string. Use `sm_robot` for the robotic arm model and `stewart_platform` for the Stewart platform model. For example, to import the robotic arm model, enter:

```
smimport('sm_robot');
```

The function generates a new Simscape Multibody block diagram and a supporting data file. The block diagram recreates the original CAD assembly model using Simscape Multibody blocks. The data file provides the numerical values of the block parameters used in the model.

After Model Import

Check the imported model for unexpected rigid connections between bodies. Simscape Multibody software replaces unsupported CAD constraints with rigid connections that may appear as direct frame connection lines, Rigid Transform blocks, or Weld Joint blocks.

A warning message in the MATLAB command window identifies the bodies and connection frames affected by the unsupported constraints. Replace the artificial rigid connections between the bodies with suitable Joint, Constraint, or Gear blocks from the Simscape Multibody library.

Update the block diagram to rule out model assembly errors. Run simulation to ensure the model dynamics are as expected. If you update the source CAD assembly model, you can generate an updated data file directly from a new multibody description file.

Import a Robotic Arm CAD Model

In this section...

“Example Overview” on page 6-14

“Example Files” on page 6-14

“Import the Model” on page 6-14

“Visualize the Model” on page 6-15

“Build on the Model” on page 6-16

Example Overview

This example shows how to generate a Simscape Multibody model from a multibody description XML file using the `smimport` function. The example is based on a multibody description file named `sm_robot` and a set of part geometry files included in your Simscape Multibody installation. These files describe the robotic arm model shown in the figure.



Example Files

The multibody description and part geometry files used in this example are located in the folder `matlabroot\toolbox\physmod\sm\smdemos\import\robot`

where `matlabroot` is the root folder of your MATLAB installation, for example:

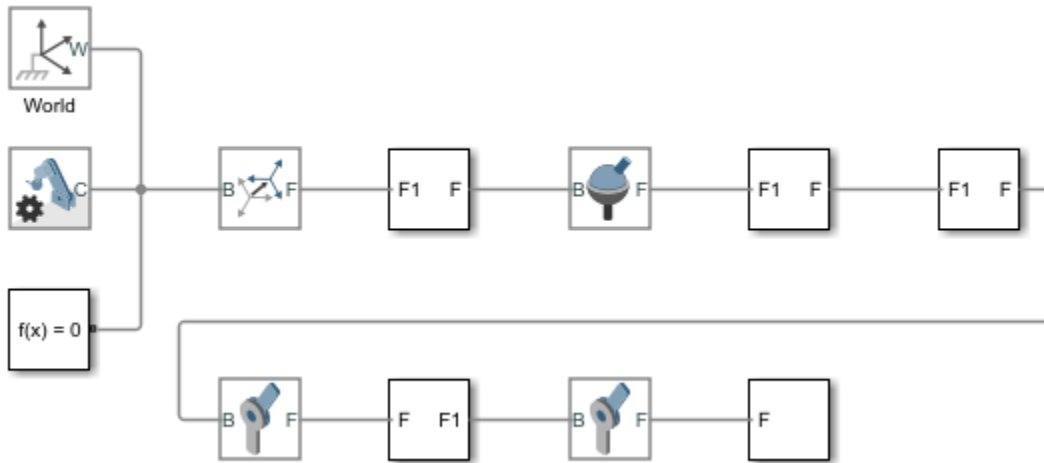
```
C:\Programs\MATLAB\
```

Import the Model

At the MATLAB command prompt, enter the command:

```
smimport('sm_robot');
```

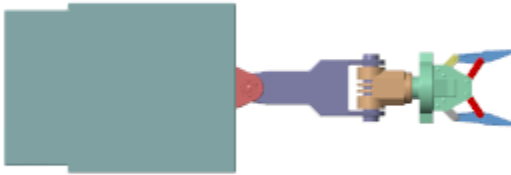
Simscape Multibody software generates the model described in the `sm_robot.xml` file using the default `smimport` function settings.



The blocks in the generated model are parameterized in terms of MATLAB variables. The numerical values of these variables are defined in a data file that is named `sm_robot.m` and stored in the same active folder as the generated model.

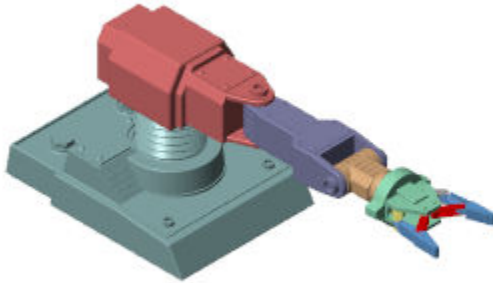
Visualize the Model

Update the diagram to visualize the model. In the **Modeling** tab, click **Update Model**. Mechanics Explorer opens with a static visualization of the robotic arm model in its initial configuration.



The default view convention in Mechanics Explorer differs from that in the CAD application used to create the original assembly model. Mechanics Explorer uses a Z-axis-up view convention while the CAD application uses a Y-axis-up view convention.

Change the view convention from the Mechanics Explorer toolbar by setting the **View convention** parameter to **Y up (XY Front)**. Then, select a standard view from the **View > Standard Views** menu to apply the new view convention.



Build on the Model

Try to simulate the model. Because the robotic arm lacks a control system, it simply flails under gravity. You can use Simulink blocks to create the control system needed to guide the robotic arm motion. A control system would convert motion sensing outputs into actuation inputs at the various joints. You can expose the sensing and actuation ports from the joint block dialog boxes.

See Also

`smimport`

Related Examples

- “Import a CAD Assembly Model” on page 6-12

Onshape Import

In this section...

“What Is Onshape?” on page 6-17
“What’s in an Onshape Model?” on page 6-17
“Preparing a Model for Import” on page 6-18
“Importing an Onshape Model” on page 6-18
“What Can You Import?” on page 6-19
“User Authentication and Account Permissions” on page 6-19
“Mapping to Simscape Multibody Blocks” on page 6-19
“Onshape Import Warnings and Errors” on page 6-20
“Physical Units” on page 6-20
“Obtaining Onshape Models to Import” on page 6-21

What Is Onshape?

Onshape is a third-party CAD application that you can import multibody models from. As with other CAD applications, you use Onshape software to model 3-D parts and articulated assemblies. Onshape is full-cloud software and does not rely on a local installation to run.

You must have an active Onshape account to use the software. The Simscape Multibody `smexportonshape` function replaces the Simscape Multibody Link plug-in as the CAD export means. The plug-in is incompatible with Onshape and cannot be used with Onshape models.

What’s in an Onshape Model?

Onshape models are similar in composition to other multibody models. Parts connect through mates such as Ball, Slider, and Revolute to form articulated linkages, mechanisms, and machines. The models are hierarchical, with parts and mates nested inside subassemblies that can in turn be nested inside larger subassemblies.

Each Onshape model exists in a cloud document. A document can have multiple Part Studio tabs, for modeling parts, and Assembly tabs, for mating parts. A Part Studio tab can have multiple parts and these can be modeled in the relative poses anticipated in the final assembly—for example, to form rigid groups during assembly without the aid of Fixed mates.

Rigid groups are unique to Onshape models and map into Simulink Subsystem blocks with rigidly connected bodies enclosed.

A Word about Terminology

Onshape and Simscape Multibody models have different standard terms for what are often the same things. Parts in an Onshape model are bodies in a Simscape Multibody model. Mates and relations in an Onshape model are joints and constraints in a Simscape Multibody model. These terms are used interchangeably here.

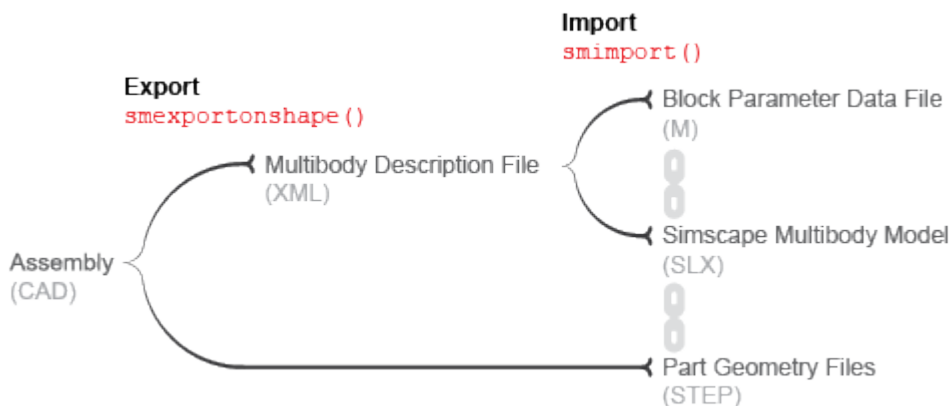
Preparing a Model for Import

Consider fixing one part in each Onshape subassembly. The fixed part determines the location of the subassembly reference frame in the imported model. You must fix the reference part directly in the subassembly tab containing that part. Parts fixed in the root assembly tab—that containing all other subassemblies or bodies—have no impact on the location of the subassembly reference frames.

Ensure that your Onshape parts are free of geometry errors and that your mates have been fully defined. If no mate exists between two parts—that is, if they have six relative degrees of freedom—the imported model will show a 6-DOF Joint block between the corresponding body subsystems. Fix at least one part at the root assembly level to prevent such a block from being added between the assembly and the World frame.

Importing an Onshape Model

You import an Onshape model into the Simscape Multibody environment using the `smexportonshape` and `smimport` functions. The `smexportonshape` function converts the Onshape model into an intermediate representation comprising an XML file and a set of STEP files. The `smimport` function converts the XML file into the final Simscape Multibody model and a supporting data file.



Onshape CAD Import Workflow

About the Intermediate Files

The XML file provides the `smimport` function the data it needs in order to recreate the Onshape model in the Simscape Multibody environment. This file is referred to as the *multibody description* file and is required for model import.

The STEP files provide the imported model, once generated, the 3-D geometries it needs in order to render the bodies in the visualization panes of solid blocks and Mechanics Explorer. The files are referred to as *geometry files* and are optional for model import.

About the Imported Model

The geometry files are referenced in the solid blocks of the imported model. If the geometry files are missing, or if the paths to the files change, the body geometries no longer show in the visualization

panes of the solid blocks and Mechanics Explorer. Simulation is unaffected provided that the model is otherwise still valid.

The remaining block parameters are specified in terms of MATLAB variables defined in the supporting data file generated by the `smimport` function. The variables are stored in a single data structure, with the data field names and indices identifying the block parameters that the variables correspond to—e.g., `smiData.Solid(2).mass`.

What Can You Import?

You can use the `smexportonshape` and `smimport` functions to import the contents of Onshape Assembly tabs only. Part Studio tabs lack the necessary information to generate a complete XML multibody description file and cause the `smexportonshape` function to error. Other Onshape document tabs, such as Drawing and Folder, present the same problem and cannot be exported.

If You Need to Import Single Parts

You can export part geometries directly from an Onshape document using the Onshape Export feature. This feature enables you to save geometries in various formats, but only two are compatible with Simscape Multibody models—STL and STEP. Once exported, the geometries can be individually imported into Simscape Multibody solid blocks. See “Imported Solid Shapes” on page 1-41.

User Authentication and Account Permissions

Before you can export an Onshape model, the `smexportonshape` function needs to authenticate your Onshape account and verify that the Simscape Multibody Exporter app has access permission to your Onshape models.

Authentication occurs once per MATLAB session and is based on a protocol known as OAuth2. So that you can validate your account, the `smexportonshape` function automatically opens an Onshape log-in page on your first export attempt of a session.

You need give the Simscape Multibody Exporter app access permissions only once until these are revoked. So that you can set the permissions for the app, the `smexportonshape` function automatically opens an Onshape application authorization page on your first-ever export attempt.

You can revoke permissions at any time from the Onshape Applications page.

About the OAuth2 Protocol

OAuth2 is an authentication protocol that delegates the authentication process to the service hosting the account—in this case, Onshape. The oauth.net website describes the protocol as a valet key that gives third-party applications such as Simscape Multibody access to some, but not all, aspects of your account. In particular, Simscape Multibody can access your app permissions data, but it cannot see or store your private log-in credentials.

Mapping to Simscape Multibody Blocks

Simscape Multibody can map all Onshape mates but Tangent to equivalent blocks. The mappings are straightforward, with only slight differences between mate and block names standing out. The table shows the mappings used during model import.

Onshape Mate	Simscape Multibody Block or Feature
Ball	Spherical Joint
Cylindrical	Cylindrical Joint
Fastened	Direct connection line on page 1-24
Parallel	Angle Constraint
Pin Slot	Pin Slot Joint
Planar	Planar Joint
Revolute	Revolute Joint
Slider	Prismatic Joint

Onshape relations such as Gear and Linear are not supported. All relations in your model are ignored during import. You can often model the ignored relations using Simscape Multibody blocks—for example, using the Common Gear Constraint block to model a Gear relation between two gears. You may need to create and carefully place new frames before adding such blocks.

Onshape Import Warnings and Errors

Invalid Assembly URLs

The `smexportonshape` function expects the URL of an Onshape Assembly tab as an argument. An Onshape document often comprises Part Studio, Drawing, and other tabs. If you inadvertently specify the URL for the wrong document tab—one not containing an assembly—the function throws an error.

Zero-Mass Bodies

Onshape parts without assigned material translate into Simscape Multibody bodies with zero inertia. Such massless bodies can cause simulation to fail due to degenerate mass errors. A MATLAB warning identifies all massless bodies identified in your model, if any. You can manually specify the mass of a massless body after import using the Brick Solid block for that body. However, as a best practice, always try to assign a material to each body in an assembly before exporting it.

Unsupported Mates and Relations

Onshape mates such as Tangent and relations such as Gear are not supported. The `smexportonshape` function throws a warning identifying all unsupported mates and relations, if any. Unsupported mates map into nothing in a Simscape Multibody model. For a list of Onshape mates that you can import, see “Mapping to Simscape Multibody Blocks” on page 6-19.

Physical Units

The block parameters in the imported model are in the default units of the Onshape model workspace. These units often include a mix derived from SI, CGS, and other unit systems. You can change the units for an entire model in your Onshape model workspace and for an individual block in your imported Simscape Multibody model.

Obtaining Onshape Models to Import

You can create a free account and create your own assembly model in—or import one into—your Onshape account. Many Onshape assembly models are public. You can import all such models into the Simscape Multibody environment.

See Also

`smexportonshape` | `smimport`

More About

- “Import an Onshape Humanoid Model” on page 6-22
- “CAD Translation” on page 6-2
- “Import URDF Models” on page 6-33

Import an Onshape Humanoid Model

In this section...

“Onshape Import” on page 6-22
“Example Overview” on page 6-22
“Export the Model” on page 6-22
“Import the Model” on page 6-23

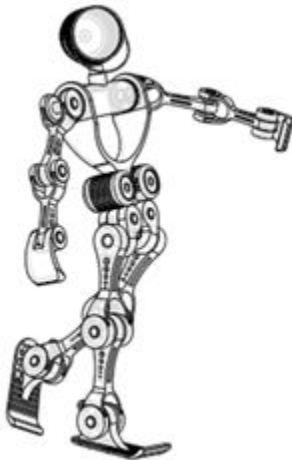
Onshape Import

You can import a CAD assembly model from Onshape software into the Simscape Multibody environment. The import process occurs in two steps based on the `smexportonshape` and `smimport` functions. The `smexportonshape` exports the assembly model in an intermediate XML conforming to the Simscape Multibody XML schema. The `smimport` function converts the intermediate XML file into a Simscape Multibody version of the original Onshape model.

Example Overview

This example shows how to import an Onshape model of a humanoid robot assembly. The model comprises various parts representing the torso, head, and limbs of the robot. The parts connect through Revolute mates that represent the various joints. The model is identical to that shown in “Import a URDF Humanoid Model” on page 6-40. Enter the following URL in your web browser to access the model (Onshape login required):

<https://cad.onshape.com/documents/5817806f96eae5105bfa5085/w/15ab3bfb58cacbf427d77ff3/e/181493813f84966648a8db1b>



Model Schematic

Export the Model

Use the `smexportonshape` function to export the model:

- 1 At the MATLAB command prompt, navigate to a folder for which you have write privileges—for example:

```
cd C:\Users\JDoe\Documents\Models
```

- 2 Store the model URL in a MATLAB variable named `assemblyURL`:

```
assemblyURL = 'https://cad.onshape.com/documents/5817806f96eae5105bfa5085/w/15ab3bfb58cacbf42
```

- 3 Export the model and save the XML file name in a variable named `exportedModel`:

```
exportedModel = smexportonshape(assemblyURL);
```

You may be prompted to log in to your Onshape account. The `smexportonshape` function generates the XML multibody description file for this model and a set of STEP files for the various part geometries.

Import the Model

Use the `smimport` function to import the XML multibody description file:

```
smimport(exportedModel);
```

The function generates a Simscape Multibody model of the humanoid robot.



Build on the model, for example, by adding control systems to actuate the various joints. For a controlled example, at the MATLAB command prompt enter `sm_import_humanoid_urdf`. Simulate the model to view a simple animation.



See Also

smexportonshape | smimport

More About

- “Onshape Import” on page 6-17
- “CAD Translation” on page 6-2
- “Import URDF Models” on page 6-33

URDF Primer

In this section...

“URDF Elements and Attributes” on page 6-25
 “XML Hierarchies and Kinematic Trees” on page 6-26
 “Required and Optional URDF Entities” on page 6-28
 “Create a Simple URDF Model” on page 6-29
 “Obtaining URDF Models to Import” on page 6-31

Unified Robotics Description Format, URDF, is an XML specification used in academia and industry to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots for amusement parks. URDF is especially popular with users of Robotics Operating System, ROS. You can import URDF models into the Simscape Multibody environment for simulations, analyses, or control design tasks. See the “Humanoid Robot” on page 8-63 featured example for a simple use case.



Humanoid Robot URDF Model

URDF Elements and Attributes

Like other types of XML files, URDF files comprise various XML elements, such as `<robot>`, `<link>`, `<joint>`, nested in hierarchical structures known as XML trees. For example, the `<link>` and `<joint>` elements are said to be children of the `<robot>` element and, reciprocally, the `<robot>` element the parent of the `<link>` and `<joint>` elements.

```

<robot>
  <link>
    ...
  </link>
  <link>
    ...
  </link>
  <joint>
    ...
  </joint>
</robot>
  
```

Child elements, such as `<link>` and `<joint>` under `<robot>`, can in turn have child elements of their own. For example, the `<link>` element has the child elements `<inertial>` and `<visual>`. The `<visual>` element has the child elements `<geometry>` and `<material>`. And the `<material>` element has the child element `<color>`. Such chains of child elements are essential to define the properties and behavior of the parent elements.

```
<robot>
  <link>
    <inertial>
      ...
    </inertial>
    <visual>
      <geometry>
        ...
      </geometry>
      <material>
        <color />
      </material>
    </visual>
  </link>
  ...
</robot>
```

In addition to child elements, the XML elements in a URDF model can have attributes. For example, the `<robot>`, `<link>`, and `<joint>` elements all have the attribute `<name>`—a string that serves to identify the element. The `<color>` element has the attribute `rgba`—a numeric array with the red, green, blue, and alpha (or opacity) values of the link color. Attributes such as these help to completely define the elements in the model.

```
<robot name = "linkage">
  <link name = "root link">
    <inertial>
      ...
    </inertial>
    <visual>
      <geometry>
        ...
      </geometry>
      <material>
        <color rgba = "1 0 0 1" />
      </material>
    </visual>
  </link>
  ...
</robot>
```

XML Hierarchies and Kinematic Trees

URDF links connect through joints in hierarchical structures not unlike those formed by nesting XML elements in a URDF file. `<joint>` elements enforce these hierarchies through `<parent>` and `<child>` elements that identify one link as the parent and the other as the child. Parent links can themselves be children—and child links parents—of other links in the model.

```
<robot name = "linkage">
  <joint name = "joint A ... >
    <parent link = "link A" />
```

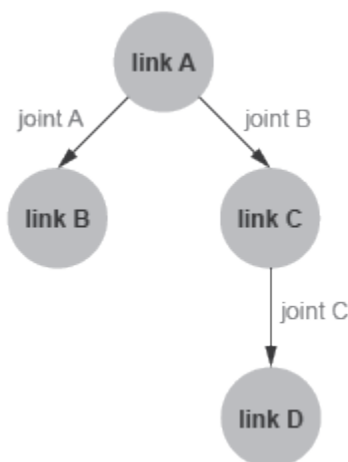
```

    <child link = "link B" />
  </joint>
  <joint name = "joint B ... >
    <parent link = "link A" />
    <child link = "link C" />
  </joint>
  <joint name = "joint C ... >
    <parent link = "link C" />
    <child link = "link D" />
  </joint>
</robot>

```

<parent> and <child> Joint Elements

You can visualize the connections between links using a schematic known as a connectivity graph. The figure shows an example. Circles denote links and arrows denote joints. The arrow roots identify the parent nodes and the arrow tips the child nodes. The connectivity graph reveals the topology of the underlying model—here a simple kinematic tree with two branches.



Connectivity Graph of a Kinematic Tree

Model topology is important in URDF. The connectivity graph of a model can take the shape only of a kinematic tree—a kinematic chain, branched or unbranched, that is always open. Kinematic loops, each a closed chain formed by joining the ends of an otherwise open chain, are disallowed. This restriction impacts how <link> elements can connect in a URDF model.

The restriction translates to the following rule: no <link> element can serve as a child node in more than one <joint> element. Put another way, no <link> element can have more than one parent element in the model's connectivity graph. Only the root link, that at the origin of the connectivity graph, can have a number of parent nodes different from one (zero). Only one root link is allowed in a model.

```

<robot name = "linkage">
  <joint name = "joint A ... >
    <parent link = "link A" />
    <child link = "link B" />
  </joint>
  <joint name = "joint B ... >

```

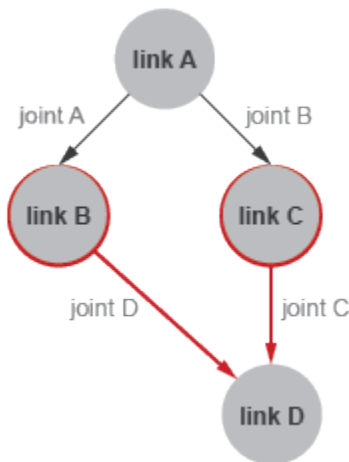
```

    <parent link = "link A" />
    <child link = "link C" />
  </joint>
  <joint name = "joint C ... >
    <parent link = "link C" />
    <child link = "link D" />
  </joint>
  <joint name = "joint D ... >
    <parent link = "link B" />
    <child link = "link D" />
  </joint>
</robot>

```

Kinematic Loop URDF Example

The code declares a link, link D, as a child node in two <joint> elements, joint C and joint D. The link D element has two parents and forms a kinematic loop. The model violates the URDF connection rules and is invalid. The figure shows the connectivity graph of the model.



Connectivity Graph of a Kinematic Loop

Required and Optional URDF Entities

Not all elements and attributes listed in the URDF specification are required. Some, like <inertial> under <link>, are optional. The following code shows the various elements and attributes that you can use, with those that are optional colored green.

Elements and attributes shown as required inside optional elements are so only if the optional elements are used. The default values of optional attributes are shown in parentheses and in italic font. Note that this code is included only as a reference and that it does not represent a valid URDF model. Ellipses ("...") are invalid in URDF models and are used merely to break long lines of code for ease of viewing.

```

<robot name>
  <link name>
    <inertial>
      <origin xyz("0 0 0") rpy("0 0 0") />
      <mass value />

```

```

    <inertia ixx iyy izz ixy ixz iyz />
</inertial>
<visual name>
  <origin xyz("0 0 0") rpy("0 0 0") />
  <geometry>
    <box size />
    <cylinder radius length />
    <sphere radius />
    <mesh filename scale("1") />
  </geometry>
  <material name>
    <color rgba("0.5 0.5 0.5 1") />
    <texture filename />
  </material>
</visual>
<collision name>
  <origin xyz("0 0 0") rpy("0 0 0") />
  <geometry>
    <box size />
    <cylinder radius length />
    <sphere radius />
    <mesh filename scale("1") />
  </geometry>
</collision>
</link>
<joint name type>
  <origin xyz("0 0 0") rpy("0 0 0") />
  <parent link />
  <child link />
  <axis xyz("1 0 0") />
  <calibration rising />
  <calibration falling />
  <dynamics damping("0") friction("0") />
  <limit† lower† upper† effort velocity />
  <mimic joint multiplier("1") offset("0") />
  <safety_controller soft_lower_limit("0") ...
  ... soft_upper_limit("0") k_position("0") k_velocity("0") />
</joint>
</robot>

```

[†]Required for <joint> elements of type prismatic and revolute only.

Create a Simple URDF Model

As an example, create a URDF model of a double pendulum. In your text editor of choice, create a file with the code shown below and save the file as `double_pendulum.urdf` in a convenient folder. Include the file extension in the file name. A separate example shows how to import this model into the Simscape Multibody environment (see “Import a Simple URDF Model” on page 6-36).

```

<robot name = "linkage">
  <!-- links section -->
  <link name = "link A">
    <inertial>
      <origin xyz = "0 0 0" />
      <mass value = "0.5" />
      <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />

```

```

    </inertial>
    <visual>
      <origin xyz = "0 0 0" />
      <geometry>
        <box size = "0.5 0.5 0.1" />
      </geometry>
      <material name = "gray A">
        <color rgba = "0.1 0.1 0.1 1" />
      </material>
    </visual>
  </link>
  <link name = "link B">
    <inertial>
      <origin xyz = "0 0 -0.5" />
      <mass value = "0.5" />
      <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />
    </inertial>
    <visual>
      <origin xyz = "0 0 -0.5" />
      <geometry>
        <cylinder radius = "0.05" length = "1" />
      </geometry>
      <material name = "gray B">
        <color rgba = "0.3 0.3 0.3 1" />
      </material>
    </visual>
  </link>
  <link name = "link C">
    <inertial>
      <origin xyz = "0 0 -0.5" />
      <mass value = "0.5" />
      <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />
    </inertial>
    <visual>
      <origin xyz = "0 0 -0.5" />
      <geometry>
        <cylinder radius = "0.05" length = "1" />
      </geometry>
      <material name = "gray C">
        <color rgba = "0.5 0.5 0.5 1" />
      </material>
    </visual>
  </link>

  <!-- joints section -->
  <joint name = "joint A" type = "continuous">
    <parent link = "link A" />
    <child link = "link B" />
    <origin xyz = "0 0 -0.05" />
    <axis xyz = "0 1 0" />
  </joint>
  <joint name = "joint B" type = "continuous">
    <parent link = "link B" />
    <child link = "link C" />
    <origin xyz = "0 0 -1" />
    <axis xyz = "0 1 0" />

```

```

    <physics damping = "0.002" />
  </joint>
</robot>

```

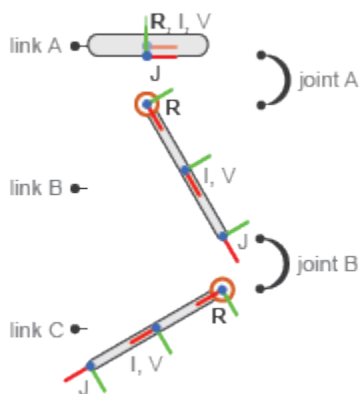
About the URDF Model

The code defines a multibody model named `linkage`. The model contains three links, named `link A`, `link B` and `link C`, that connect via two joints, named `joint A` and `joint B`. The `<parent>` and `<child>` elements of the joints identify how the links connect to each other: `link A` connects to `link B` and `link B` connects to `link C`. `link A` has no parent link—that is, it appears in `<joint>` elements as a child element only—and is therefore the root link.

The `<inertial>` element of `link A` defines the mass and moments of inertia (`ixx`, `iyy`, `izz`) of the link. The products of inertia (`ixy`, `ixz`, and `iyz`) are unspecified and have the URDF default value of zero. The `<visual>` element of `link A` defines the geometry type and material color for use in the model visualization. The geometry in this case is a box with width and thickness of `0.5` m and height of `0.1` m. The `<origin>` elements of the link `<inertial>` and `<visual>` specify the transforms from the link reference frame to the inertial and visual reference frames. Similar elements apply to `link B` and `link C`.

The `type` attribute of the `<joint>` elements defines the joints as continuous—a type of revolute joint without motion limits. The `<origin>` element specifies the location of the joint relative to the reference frame of the parent link element. For example, the `<origin>` element of `joint A` offsets the joint `0.05` m along the `-Z` axis relative to the origin of the `link A` reference frame. The axis element nested inside each `<joint>` element defines the rotational axis of the joint as the Cartesian vector `[0, 1, 0]`, or `+Y`.

The figure shows the components of the model—the links and joints—and the various frames they contain. **R** denotes a link reference frame, **I** a link inertial frame, and **V** a link visual frame. **J** denotes a joint reference frame—by definition held coincident with the reference frame of the child link. The inertial and visual frames are offset to the centers of the links and the joint frames to their lower edges.



Double-Pendulum Model Components

Obtaining URDF Models to Import

You can, but generally do not have to, manually create your own URDF files. For more complex models, it can be preferable to obtain URDF files from other sources. Robotics manufacturers and

consultants often provide URDF models for their robotic systems. CAD applications such as SolidWorks and PTC Creo support URDF exporters that convert your CAD assemblies into URDF models. Consider these options when working with complex robotics models that may not be simple to create manually.

See Also

smimport

More About

- “Import URDF Models” on page 6-33
- “CAD Translation” on page 6-2

Import URDF Models

In this section...

“Supported URDF Elements and Attributes” on page 6-33

“Mapping to Simscape Multibody Blocks” on page 6-34

“Mesh Geometries” on page 6-35

“Physical Units” on page 6-35

“URDF Import Limitations” on page 6-36

“Differences from CAD Import” on page 6-36

“Import a Simple URDF Model” on page 6-36

To import a URDF model into a Simscape Multibody model, use the `smimport` function. You must specify the file extension.

```
smimport('sm_humanoid.urdf')
```

If you omit the file extension, the `smimport` function assumes that the file is in the XML format used to import CAD models. For example, the command

```
smimport('sm_humanoid')
```

tells the function to create a multibody model from the `sm_humanoid` XML file. If the function finds no XML file with the specified name, it returns an error even if there is a URDF file with the same name in the same folder.

Supported URDF Elements and Attributes

The `smimport` function supports a subset of elements and attributes of the URDF specification. When you import a URDF model with unsupported elements or attributes, the `smimport` function recognizes only the supported elements and attributes. The following example shows all the supported elements and attributes in black.

```
<robot name>
  <link name>
    <inertial>
      <origin xyz rpy />
      <mass value />
      <inertia ixx iyy izz ixy ixz iyz />
    </inertial>
    <visual name>
      <origin xyz rpy />
      <geometry>
        <box size />
        <cylinder radius length />
        <sphere radius />
        <mesh filename scale />
      </geometry>
      <material name>
        <color rgba />
        <texture filename />
      </material>
    </visual>
    <collision name>
      <origin xyz rpy />
      <geometry>
        <box size />
        <cylinder radius length />
        <sphere radius />
        <mesh filename scale />
      </geometry>
```

```

    </collision>
  </link>
  <joint name type>
    <origin xyz rpy />
    <parent link />
    <child link />
    <axis xyz />
    <calibration rising />
    <calibration falling />
    <dynamics damping friction />
    <limit lower upper effort velocity />
    <mimic joint multiplier offset />
    <safety_controller soft_lower_limit ...
    ... soft_upper_limit k_position k_velocity />
  </joint>
</robot>

```

Elements are in bold font and attributes are in regular font. Unsupported elements and attributes are highlighted in red. The `smimport` function does not support an element or attribute if the element or attribute is not shown in the above example. Note that the `scale` attribute of the `<visual>/<geometry>/<mesh>` element is shown in orange because the attribute is partially supported. For more information about the `scale` attribute, see the “Mesh Geometries” on page 6-35 section.

Mapping to Simscape Multibody Blocks

The URDF `<robot>` element maps into a Simscape Multibody model. The `<link>` elements nested inside the `<robot>` element map into Simulink Subsystem blocks representing the links or, in Simscape Multibody nomenclature, bodies. The `<joint>` elements map into equivalent Simscape Multibody joint blocks. The `name` attributes of these elements map into the model name, the Subsystem block names, and the joint block names, respectively.

The Subsystem blocks comprise solid, Inertia, Rigid Transform, and Reference Frame blocks. The solid blocks provide the geometries and colors of the body; these blocks correspond to the `<visual>` tags of the URDF model and are named `Visual`. The Inertia block provides the mass, center of mass, moments of inertia, and products of inertia of the body; this block corresponds to the `<inertial>` element of the URDF model and it is named `Inertia`.

The Rigid Transform blocks provide the translational and rotational offsets from the local reference frame of the body to the Inertial and Visual elements. These transforms are derived from the `<origin>` elements of the `<inertial>` and `<visual>` elements of links, as well as from the `<origin>` and `<axis>` elements of joints. The Reference Frame block identifies the local reference frame of the body.

The type of joint block used depends on the `<type>` attribute of the `<joint>` element. The joint mapping between URDF and Simscape Multibody software is largely intuitive. A `<joint>` element of type `prismatic` maps into a Prismatic Joint block. A `<joint>` element of type `fixed` maps into a Weld Joint block. The table shows the mappings for the remaining URDF `<joint>` elements.

Correspondence Between URDF and Simscape Multibody Joints

URDF <joint type> Attribute	Simscape Multibody Joint Block	Degrees of Freedom
revolute	Revolute Joint	One rotational (with limits)
continuous	Revolute Joint	One rotational
prismatic	Prismatic Joint	One translational (with limits)
fixed	Weld Joint	None
floating	6-DOF Joint	Three rotational and three translational
planar	Planar Joint	Two rotational and one translational

Mesh Geometries

To specify the visual geometries of a URDF model by using external geometry files, use the <mesh> element, for example:

```
<link name="link_1.3">
  <visual>
    <origin rpy="0 -1.57079632679 0" xyz="0 0 0.0425"/>
    <geometry>
      <mesh filename="package://xela_models/mesh_simplified/finger_link_3.stl" scale="0.001 0.001 0.001"/>
    </geometry>
  </visual>
</link>
```

The geometry files are not a part of the URDF file, and you must save the geometry files in the same folder as the corresponding URDF file. The geometry files must be in STL or STEP format. Note that the `smimport` function does not support Collada (DAE) files. If you import a model with references to DAE files, the Simscape Multibody model does not render the geometries derived from these files. The lack of visualization may limit your ability to analyze the model but has no impact on the model dynamics.

When you import a URDF model whose visual geometries refer to external files, the vertex data in the files remain unscaled regardless of the value of the `scale` attribute. The `scale` attribute specifies the units for the vertex data in the converted `File Solid` blocks.

To specify the units to cm, set the *x*, *y*, and *z*-axis scale factors of the `scale` attribute to 0.01. To specify the units as mm, set all three factors to 0.001. Note that all three factors must have the same value, and you must specify the value as either 0.01 or 0.001. If the values do not meet these conditions, the `smimport` function specifies the units to meter.

Physical Units

In the converted Simscape Multibody model, the block parameters use the International System of Unit, SI.

URDF Import Limitations

The `smimport` function imports only URDF models with tree topologies and does not support URDF variants such as SDF (Simulation Description Format) and DrakeURDF. However, after importing a URDF, you can add blocks to the Simscape Multibody model to form a kinematic loop.

The `smimport` function does not support the elements and attributes derived from URDF extensions, such as `<transmission>`, `<gazebo>`, `<model_state>`, and `<sensor>` elements.

Differences from CAD Import

Despite their similarities, including their mutual reliance on the `smimport` function, CAD and URDF import differ in some important aspects:

- CAD models are imported in an intermediate XML format. URDF models are imported directly in URDF format.

The intermediate XML files provide the information needed to recreate the CAD models in the Simscape Multibody environment. The same information is provided directly in URDF files when importing URDF models. XML multibody description files must conform to the Simscape Multibody XML schema. See “Exporting a CAD Model” on page 6-5 for ways to generate a valid XML file.

- Imported CAD models have their numerical parameters defined in MATLAB files. Imported URDF models have their numerical parameters hardcoded into the block dialog boxes.

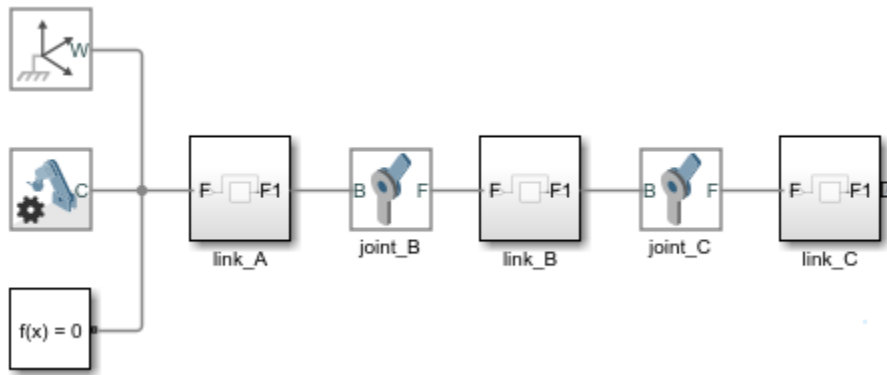
CAD import uses a detached data framework that places all block parameter values in a cell structure defined in a separate MATLAB data file. The detached data framework enables you to update a previously imported model when you modify the source CAD model. URDF import lacks this feature and does not support model update.

Import a Simple URDF Model

As an example, import the double-pendulum URDF model described in the “Create a Simple URDF Model” on page 6-29 section. Create the URDF model if you have not yet done so before proceeding. To import the model, navigate to the folder in which you saved your double-pendulum URDF model. Then, at the MATLAB command prompt, enter the command

```
smimport('double_pendulum.urdf')
```

If you saved your URDF model under a different name, use that name instead. The function imports the URDF model and generates an equivalent Simscape Multibody model. The figure shows the resulting model with the blocks and their connection lines slightly rearranged.



About the Imported Model

The `<link>` elements named `link A`, `link B`, and `link C` in the URDF model map into Simulink Subsystem blocks also named `link A`, `link B`, and `link C`. The `<joint A>` and `<joint B>`—each with type set to `continuous`—map into Simscape Multibody Revolute Joint blocks also named `joint A` and `joint B`.

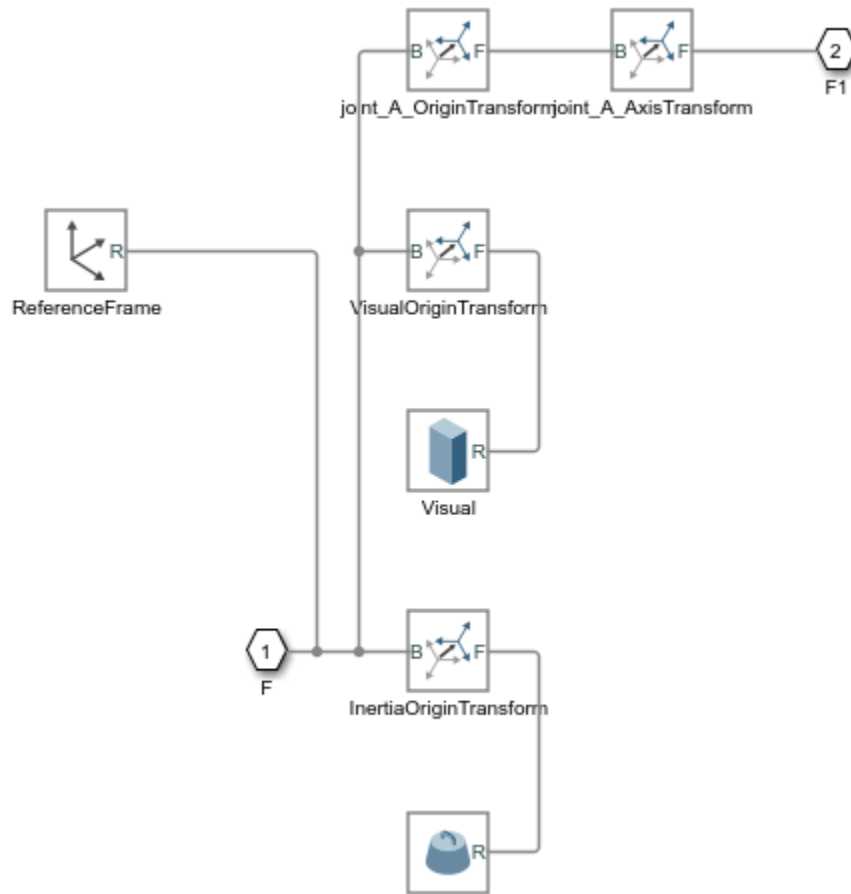
The block diagram reflects the topology of the URDF model—an unbranched kinematic tree. `link C` connects to `link B` as a child of that element. `link B` in turn connects to `link A` as a child of that element. `link A` is the root link and is therefore grounded—a condition reflected in the rigid connection between the `link A` and `World Frame` blocks.



URDF Model Topology

The Subsystem blocks representing the URDF `<link>` elements each comprise a small block diagram with one Reference Frame block, one Brick Solid block, one Inertia block, and multiple Rigid Transform blocks. The Subsystem blocks are not masked and can be opened directly with a double click. The figure shows the block diagram of the `link A` Subsystem block.

The Brick Solid block is the translated equivalent of the `<visual>` URDF element and is named `Visual`. This block contains the relevant parameters of the `<visual>` element, including link geometry and color. The Inertia block is the translated equivalent of the `<inertial>` URDF element and is accordingly named `Inertial`. This block contains the relevant parameters of the `<inertial>` element, including link mass, moments of inertia, and products of inertia.



Link A Subsystem

The Reference Frame block identifies the local reference frame of the <link> URDF element. This frame coincides with the joint connection frame to the parent link or, as in this case of a root link, to the World Frame block. The Rigid Transform blocks specify the translational and rotational transforms to the reference frames of the <inertial>, <visual>, and <joint> URDF elements. An additional Rigid Transform block specifies the rotation transform needed to align the Simscape Multibody joint axis with the URDF joint axis.

Assemble and Simulate the Imported Model

Build on the model to obtain a meaningful simulation. You can, for example, use joint state targets to assemble the double pendulum in an unstable configuration and simulate its fall under gravity:

- 1 In the dialog box of the joint_A block, select the **State Targets > Specify Position Target** check box and set the **Value** parameter to 30 deg. This parameter sets the starting angle of the upper joint.
- 2 In the **Solver** pane of the Configuration Parameters window, click **Additional options** and set the **Max step size** parameter to 0.01. This value keeps the solver step size small enough to produce a smooth animation during simulation. Increase the value if simulation proceeds slowly.

- 3 Update the block diagram and run the simulation. In the **Modeling** tab, click **Update Model**. You can simulate the model by selecting **Run**. Mechanics Explorer shows an animation of the double pendulum fall under gravity.



See Also

smimport

More About

- “CAD Translation” on page 6-2
- “Import a URDF Humanoid Model” on page 6-40
- “URDF Primer” on page 6-25
- URDF specification

Import a URDF Humanoid Model

In this section...

“URDF Import” on page 6-40

“Example Overview” on page 6-40

“Import the Model” on page 6-40

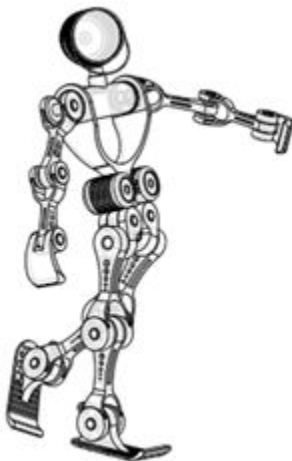
URDF Import

You can import a URDF model into the Simscape Multibody environment. The import process occurs in a single step based on the `smimport` function. The `smimport` function converts the URDF model directly into an equivalent Simscape Multibody model.

Example Overview

This example shows how to import an Onshape model of a humanoid robot assembly. The model comprises various parts (“links” in URDF jargon) representing the torso, head, and limbs of the robot. The parts connect through revolute and weld joints (“continuous” and “fixed” respectively). This model is identical to that shown in “Import an Onshape Humanoid Model” on page 6-22. You can open the model from the MATLAB command prompt by entering the command:

```
open sm_humanoid.urdf
```



Model Schematic

Import the Model

Use the `smimport` function to import the URDF model:

```
urdfModel = 'sm_humanoid.urdf';  
smimport(urdfModel);
```


The function generates a Simscape Multibody model of the humanoid robot. The file extension is required to identify the import file as URDF. Update the imported model (in the **Modeling** tab, click **Update Model**.) to open a static visualization in the initial state. The figure shows the results.



Build on the model, for example, by adding control systems to actuate the various joints. For a controlled example, at the MATLAB command prompt enter `sm_import_humanoid_urdf`. Simulate the model to view a simple animation.



See Also

`smimport`

More About

- “URDF Primer” on page 6-25
- “Import URDF Models” on page 6-33
- “CAD Translation” on page 6-2

Deployment

Code Generation

Code Generation Applications

In this section...

“Code Generation Overview” on page 7-2

“Simulation Acceleration” on page 7-2

“Model Deployment” on page 7-3

Code Generation Overview

Simscape Multibody supports code generation through Simulink Coder. You can generate C/C++ code from a Simscape Multibody model to accelerate simulation in the Simulink environment or to deploy a model onto external hardware. Model deployment requires an active Simulink Coder license while simulation acceleration does not.



Code Generation Applications

Simulation Acceleration

Simulink can generate C/C++ executable code to shorten simulation time. Two simulation modes rely on code generated from a model:

- Accelerator
- Rapid Accelerator

Simscape Multibody supports the two accelerator modes. You can access the simulation accelerator modes from the **Debug** tab. Accelerator modes do not require additional Simulink code generation products.

Note Simulation accelerator modes do not support model visualization. When you simulate a Simscape Multibody model in Accelerator or Rapid Accelerator modes, Mechanics Explorer does not open with a 3-D display of your model.

Model Deployment

With Simulink Coder, you can generate standalone C/C++ code for deployment outside the Simulink environment. The code replicates the source Simscape Multibody model. You can use the stand-alone code for applications that include:

- Hardware-In-Loop (HIL) testing
- Software-In-Loop (SIL) testing
- Rapid prototyping

Note Simscape Multibody supports, but does not perform, code generation for model deployment. Code generation for model deployment requires the Simulink Coder product.

See Also

Related Examples

- “Generate Code for a Multibody Model” on page 7-6

Code Generation Setup

In this section...

“Before You Begin” on page 7-4
 “Solver Selection” on page 7-4
 “Target Selection” on page 7-4
 “Run-Time Parameters” on page 7-4
 “Compiler Optimization” on page 7-5

Before You Begin

Simscape Multibody software supports code generation for fast simulation in the Simulink environment or for model deployment onto external targets. If your goal is to obtain standalone C/C++ code for real-time simulation on an external target, you must have an active Simulink Coder installation.

Solver Selection

Simscape Multibody models have continuous states and require a continuous or hybrid Simulink solver. You can change solvers from the **Solver** pane of the **Model Configuration Parameters** window. Select any solver but that marked `discrete (no continuous states)`. Consider the ODE1 fixed-step solver if you need to approximate the behavior of a discrete solver.

Target Selection

The choice of code generation target depends on the Simulink solver used. If you select a variable-step solver, you must set `rsim.tlc` as the system target file. You can specify the system target file from the **Model Configuration Parameters** window. Look for the **System target file** parameter in the **Target selection** area of the **Code Generation** pane.

Run-Time Parameters

You can configure most numerical block parameters as `Compile-time` (default) or `Run-time` using a drop-down list that appears beside configurable parameters. The figure shows the run-time drop-down list in a solid block dialog box. All parameters are by default `Compile-time`. The drop-down list is disabled when the model is in Fast Restart mode.



`Compile-time` parameters update in value when you recompile the model. Leave parameters as `Compile-time` when performing tasks that rely on inlined parameters such as model optimization.

Run-time parameters update in value without the need for extra compilations. Set parameters to Run-time when tuning their values in Fast Restart mode or when simulating models that rely at least in part on generated C code.

To set a parameter as Run-time from the block dialog box, you must configure your Simscape preferences. Open the MATLAB Preferences window, select the **Simscape** node, and check the **Show run-time parameter settings** check box. Parameters without a run-time option or with a run-time option that is inactive (i.e., “grayed out”) cannot be configured. The **Length** parameter in the figure is an example.

Geometry				
Extrusion Type	Regular		▼	
Number of Si...	3			
Outer Radius	1	m	▼	Compile-time ▼
Length	1	m	▼	Compile-time ▼
+ Export				
+ Inertia				
+ Graphic				
+ Frames				

For more information about Simscape run-time parameters, see “About Simscape Run-Time Parameters”.

Compiler Optimization

You can set your C/C++ compiler to optimize generated code. Optimized code runs faster but compiles slower. Compilation can be especially slow in large models with many bodies. The choice of compiler can exacerbate the slow compilation times. With certain versions of Microsoft Visual C++, Simulink software may appear to hang as the model is compiled.

If a model takes unusually long to compile, consider switching to a different installed compiler or disabling compiler optimization for your model. The Clang compiler provides a suitable alternative to Microsoft Visual C++. You can perform both tasks from the **Code Generation** menu of the **Model Configuration Parameters** window.

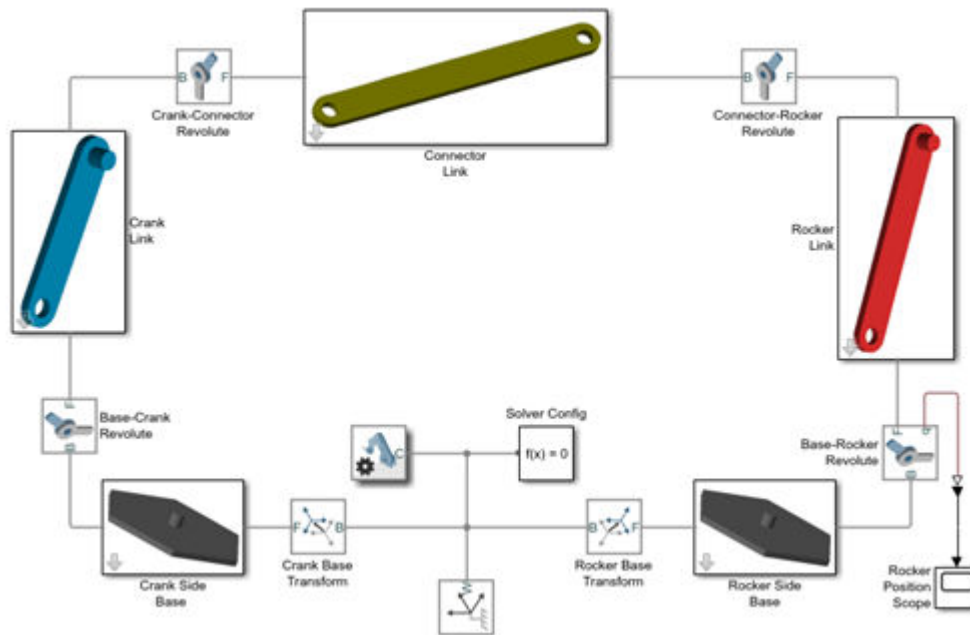
To switch compilers, in the **Toolchain settings** area of the **Code Generation** menu, set the **Toolchain** parameter to a different compiler. To disable code optimization, set the **Build configuration** parameter to **Faster Builds**.

Generate Code for a Multibody Model

This example shows how to configure and generate C code for a simple Simscape Multibody model. The example is based on a four-bar model named `sm_four_bar`. The model uses a variable-step solver, `ode45` (Dormand-Prince), and therefore requires the `rsim` target to generate code.

- 1 At the MATLAB command prompt, enter `sm_four_bar`.

MATLAB software opens the four-bar example model. Save the model with a different name in a convenient folder.



- 2 In the **Modeling** tab, click **Model Settings**.

The Model Configuration Parameters window enables you to specify a code generation target and set the code generation report options for your model.

- 3 In the **Code Generation** node of the Model Configuration Parameters window, set the **System target file** parameter to `rsim.tlc`.

The `rsim.tlc` target file is compatible with Simscape Multibody models that have variable-step solvers.

- 4 In the **Code Generation > Report** node of the Model Configuration Parameters window, check the **Create code generation report** check box and click **OK**.

MATLAB software creates and opens a code generation report when you build your model.

- 5 In the **Apps** tab, click **Simulink Coder**.

Simulink Coder software generates C code for the four-bar model. The code generation report for your model opens with a list of generated code and data files.

See Also

More About

- “Code Generation Applications” on page 7-2

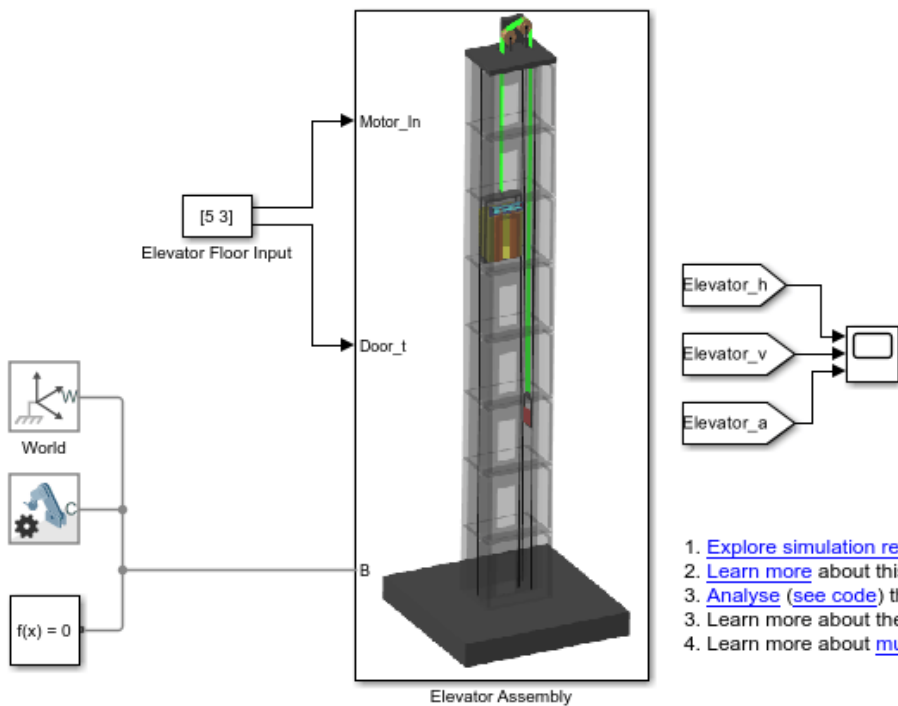
Examples

Simscape Multibody Examples

Elevator

This example models an elevator system in Simscape Multibody™. The system comprises of belt-cable pulley circuits which control the movement of the elevator and the door mechanism. The cable is approximated to be extensible by using high stiffness springs between the belt cable ends and the elevator. The motor pulley is motion actuated based on the necessary elevator kinematics computed from the Floor Number inputs. Effects of people entering and leaving the elevator are modeled using general variable mass blocks.

Model

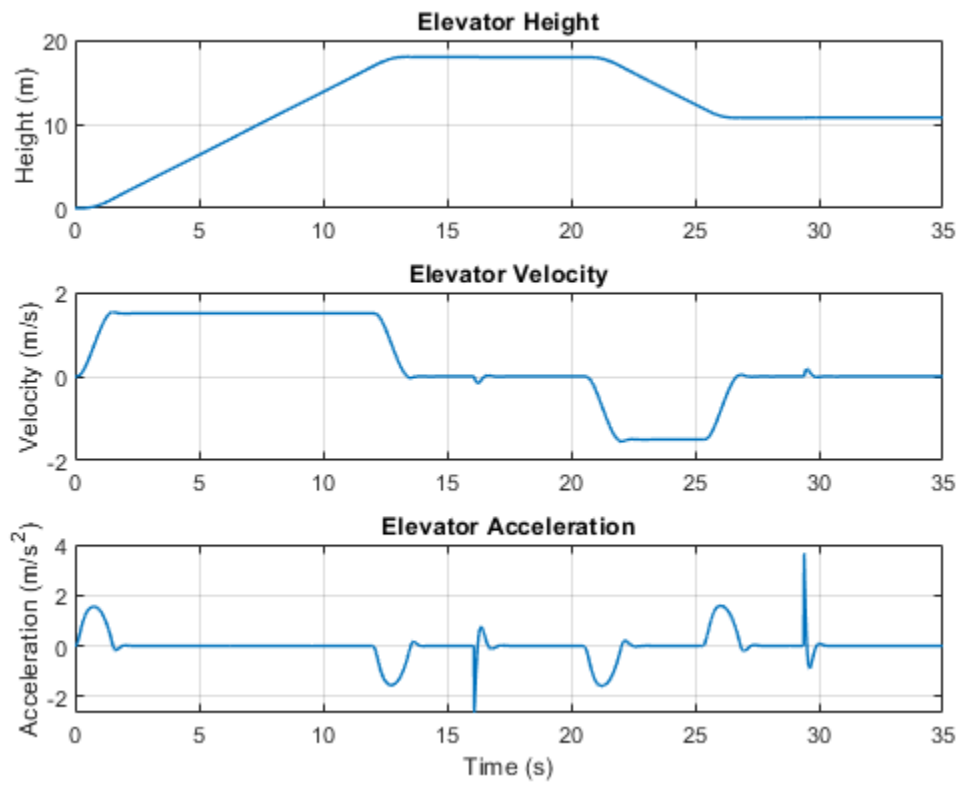


Elevator

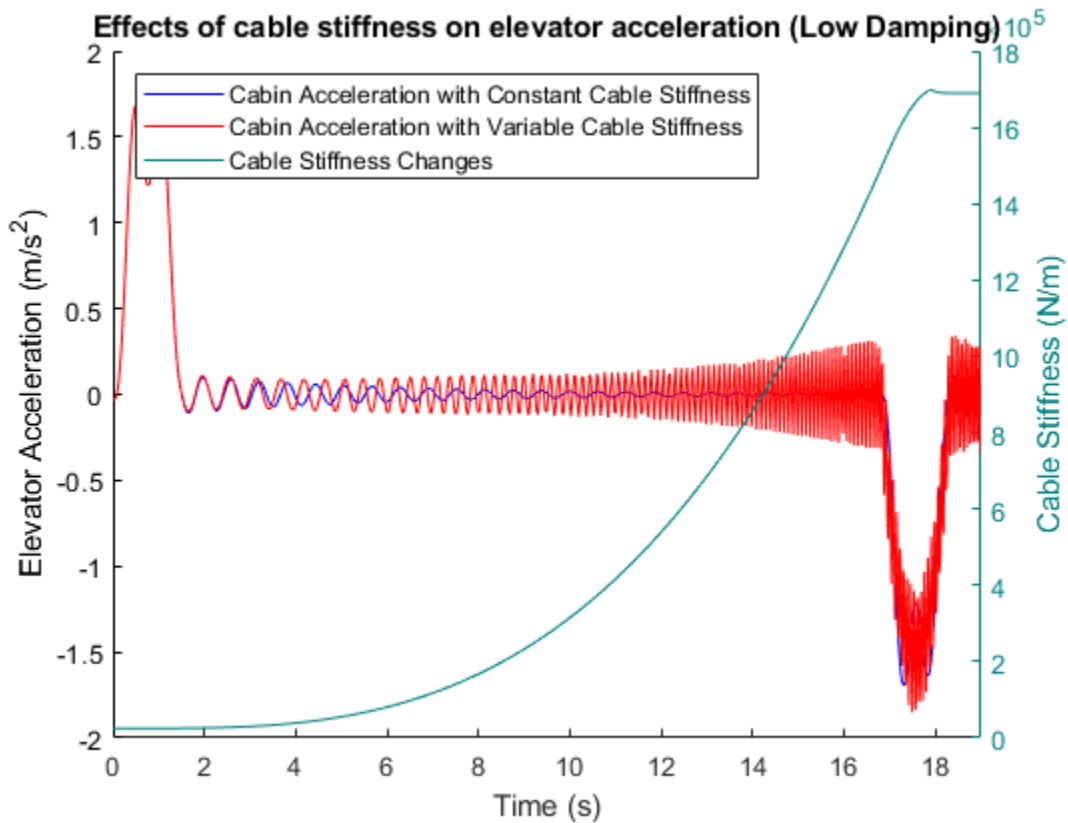
1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Learn more](#) about this example
3. [Analyse \(see code\)](#) the effects of cable stiffness on elevator dynamics
3. Learn more about the [Belt-Cable Domain](#) blocks
4. Learn more about [multibody modeling](#)

Simulation Results

Plot shows the elevator height, velocity and acceleration



Analyse the effects of cable stiffness on elevator dynamics[1].



[1] Vlastic, J.; Dokic, R.; Kljajin, M.; Karakasic, M.(2011). Modelling and simulations of elevator dynamic behaviour

See Also

Belt-Cable End | Belt-Cable Properties | Belt-Cable Spool | Pulley | Revolute Joint | Spatial Contact Force | Spherical Joint

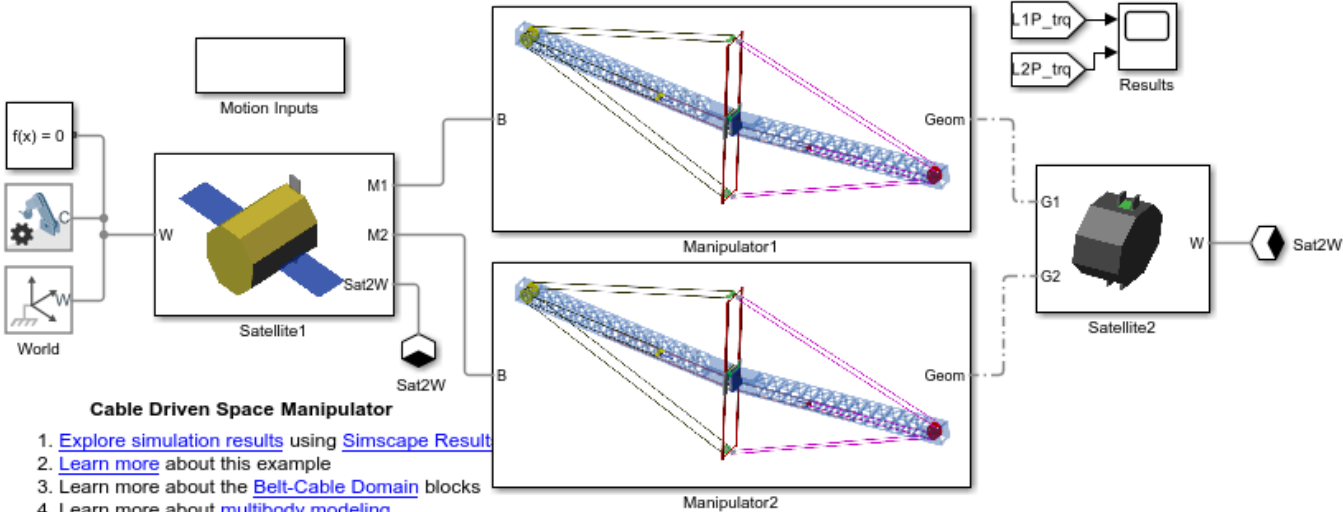
More About

- “Cable Driven Space Manipulator” on page 8-5
- “Cable Robot” on page 8-80
- “Block and Tackle with Four Pulleys” on page 8-95

Cable Driven Space Manipulator

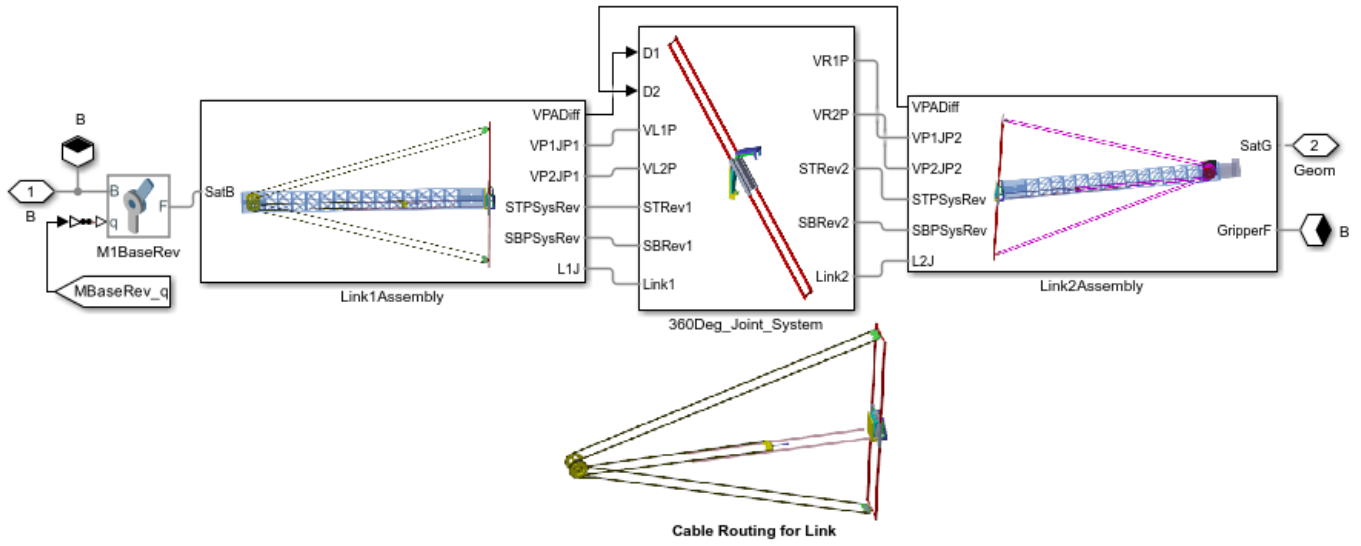
This example models a cable driven space manipulator. The manipulator comprises of 2 links connected via a system of revolute joints. Each link consists of belt-cable circuits which drive the movements of the manipulator. It also consists of a spring-damper system which provides different stiffness requirements. A space application is shown in this example where the objective of the manipulators is to capture a small satellite. The manipulators start from folded states and then perform necessary maneuvers to extend and reach the desired position. The pulleys are motion actuated from which necessary belt-cable kinematics are computed.

Model

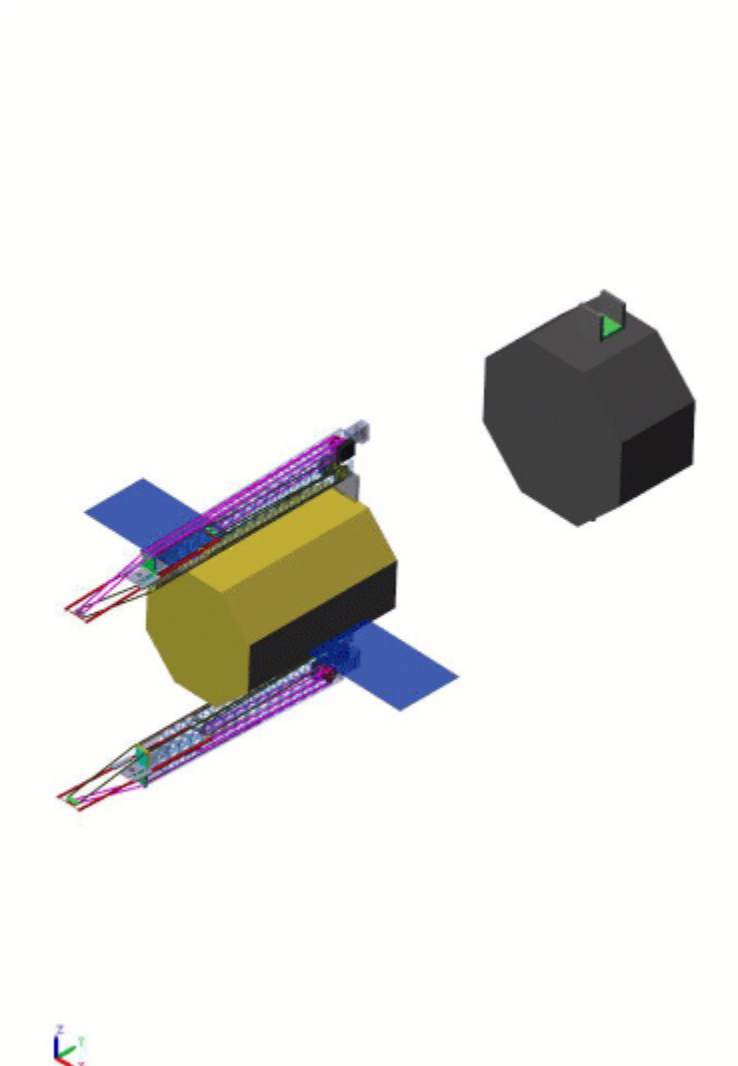


Manipulator Subsystem

Open Subsystem

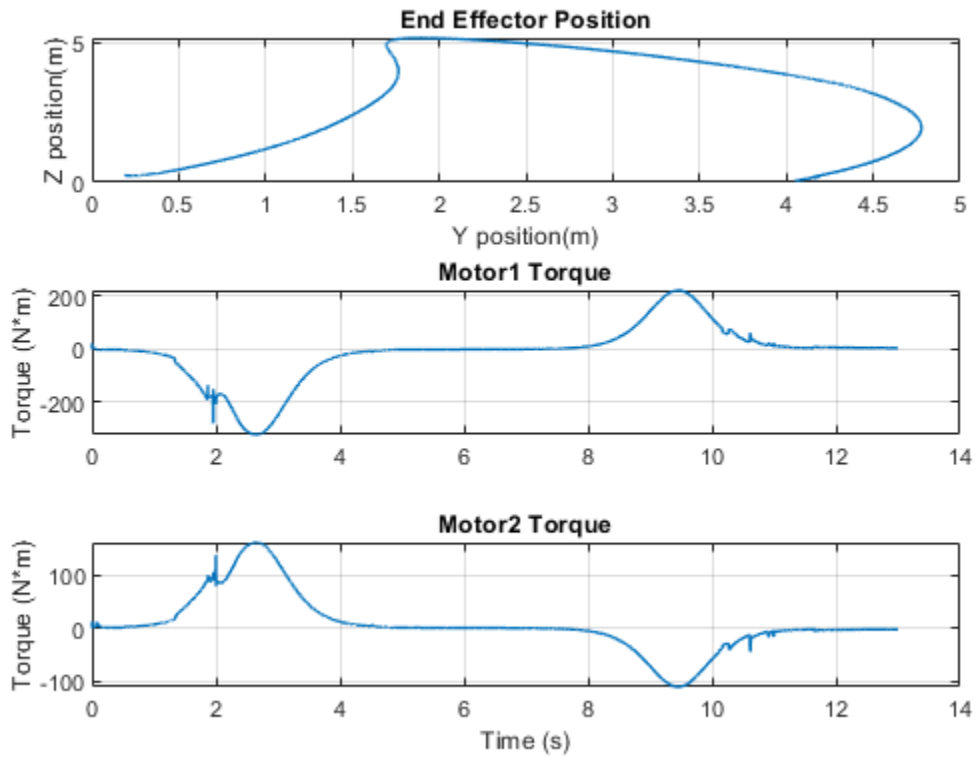


Mechanics Explorer Animation



Simulation Results from Simscape Logging

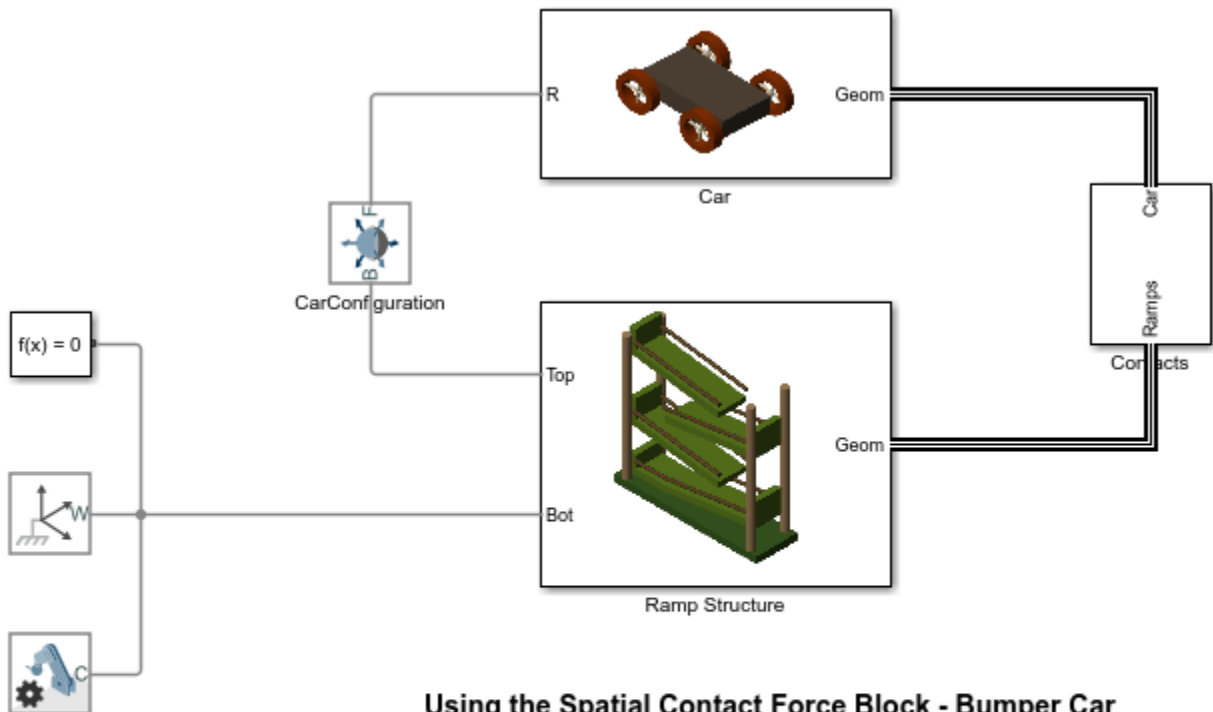
The plot below shows the Gripper Position and the torque applied to the motor pulleys.



References : [1] W.R. Doggett, J.T. Dorsey, T.C. Jones, B. King (2014). Development of a Tendon-Actuated Lightweight In-Space MANipulator (TALISMAN)

Using the Spatial Contact Force Block - Bumper Car

This example shows a toy bumper car traveling down a series of ramps while undergoing intermittent collisions. Spatial Contact Force blocks are used to model the friction and normal forces between every pair of geometries that may potentially come into contact during the simulation (e.g., between one of the car's wheels and a railing). Each Spatial Contact Force block is able to generate brief high-impact contact forces to model collisions, as well as sustained contact forces to model rolling and sliding.

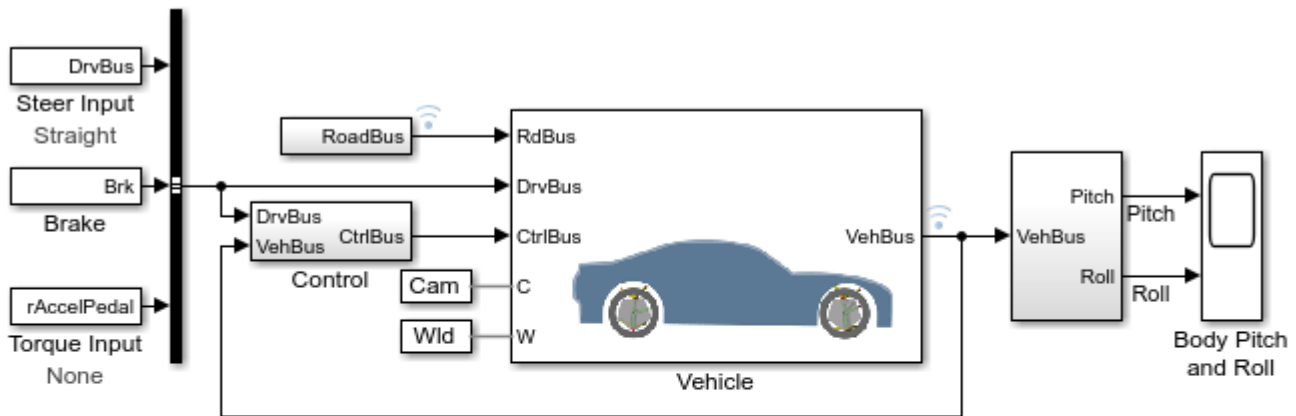


Using the Spatial Contact Force Block - Bumper Car

1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Open the subsystem](#) containing the Spatial Contact Force blocks between the car and the top ramp
3. [Learn more](#) about this example
4. Learn more about the [Spatial Contact Force Block](#)
5. Learn more about [multibody modeling](#)

Full Vehicle on Four Post Testrig

This example models a passenger vehicle on a four-post testrig. The posts move up and down to replicate the vertical movement of the wheels as it travels along a road. The simulation results and animation show the response of the vehicle body and suspension as it is subjected to the motions from the testrig. The roll and pitch of the vehicle body can be observed, and by varying the inputs wheel hop frequencies can be determined. The vehicle model can be configured to use different suspension types for the front suspension with different linkage combinations.



Full Vehicle on Four Post Testrig

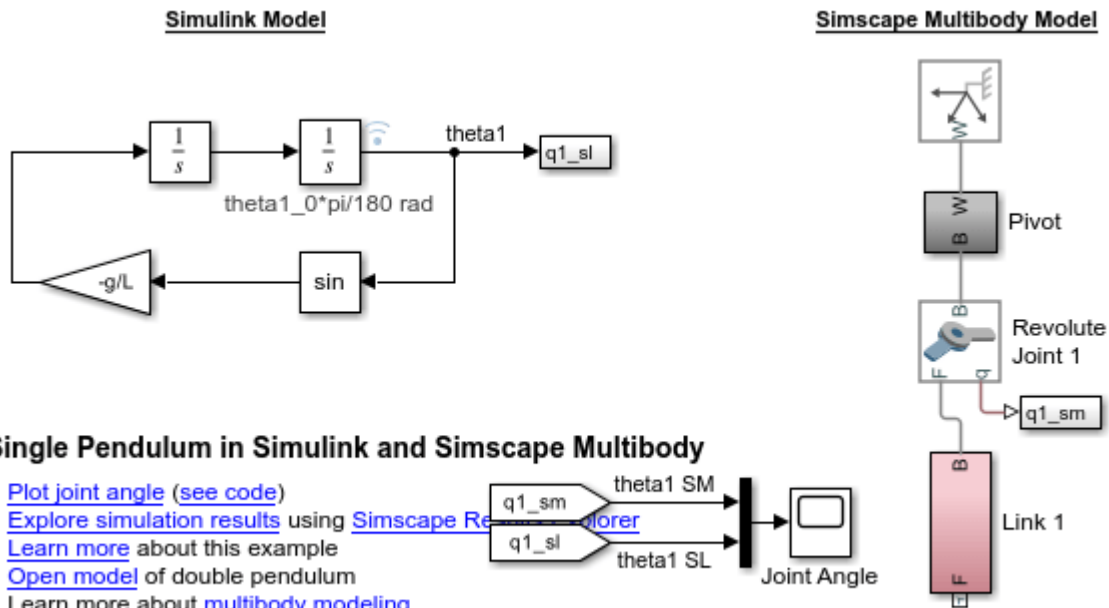
1. Plot pitch and roll of [current suspension](#) (see code), [all suspensions](#) (see code)
2. Configure Front Suspension
 - [Double Wishbone](#)
 - [Split Lower Arm, Shock to Front](#)
 - [Five Link, Shock to Rear](#)
3. [Explore simulation results](#) using [Simscape Results Explorer](#)
4. [Learn more](#) about this example

Single Pendulum in Simulink and Simscape Multibody

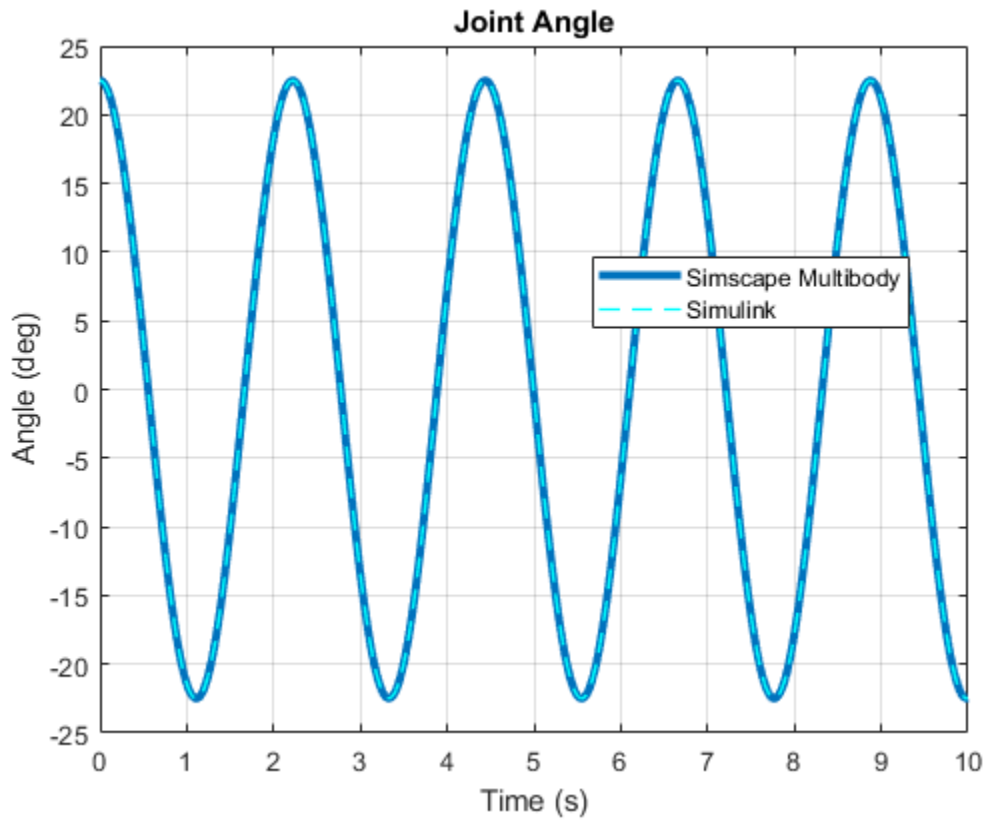
This example shows a single pendulum modeled using Simulink® input/output blocks and using Simscape™ Multibody™. The initial angle for the joint is defined by a MATLAB® variable. The annotations on the integrator block show the initial angle of the joint with respect to the world frame.

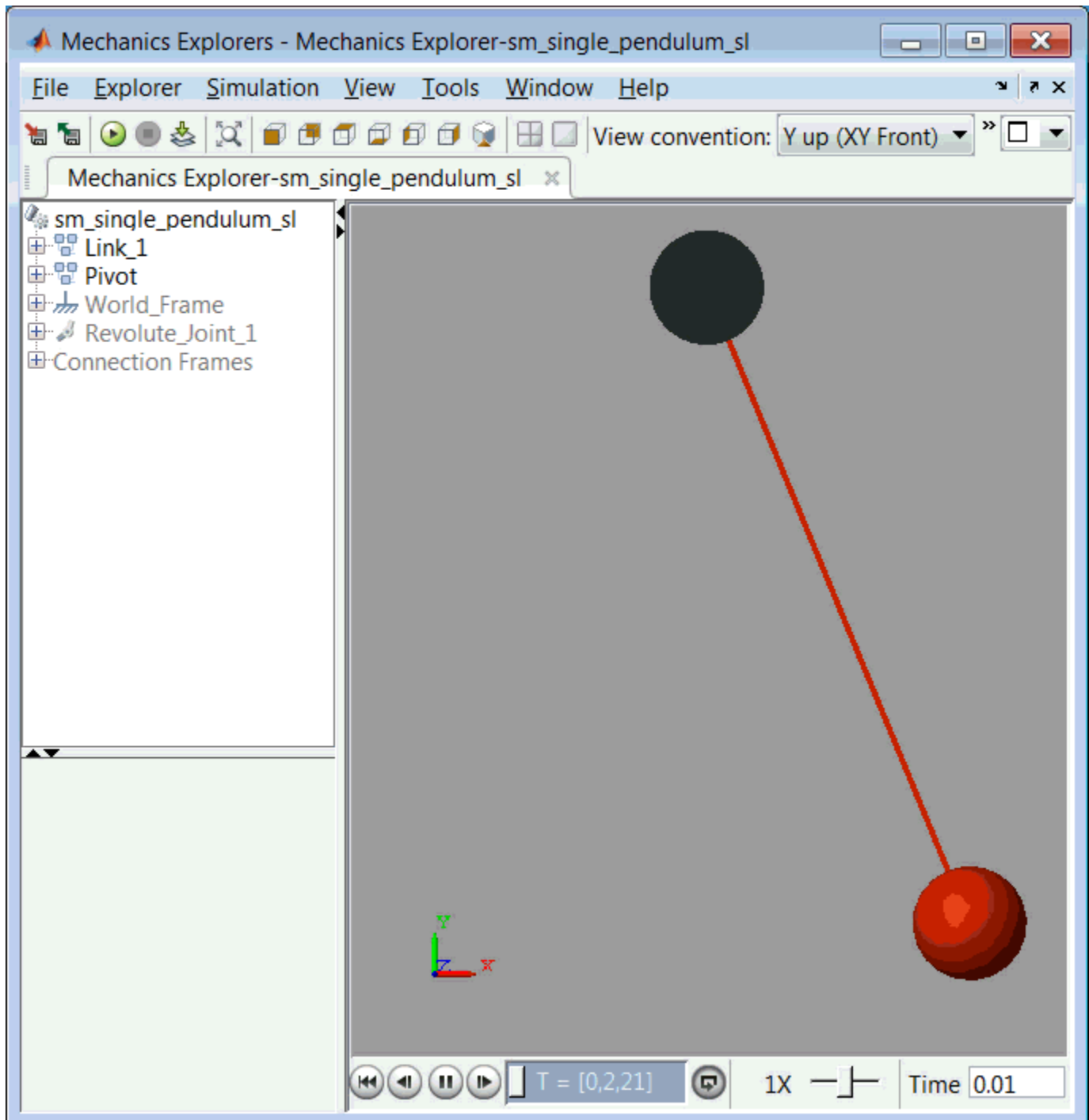
The Simulink model is built using signal connections, which define how data flows from one block to another. The Simscape Multibody model is built using physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the pendulum simply by using copy and paste. Input/output connections require rederiving and reimplementing the equations.

Model



Simulation Results from Simscape Logging



Mechanics Explorer Animation

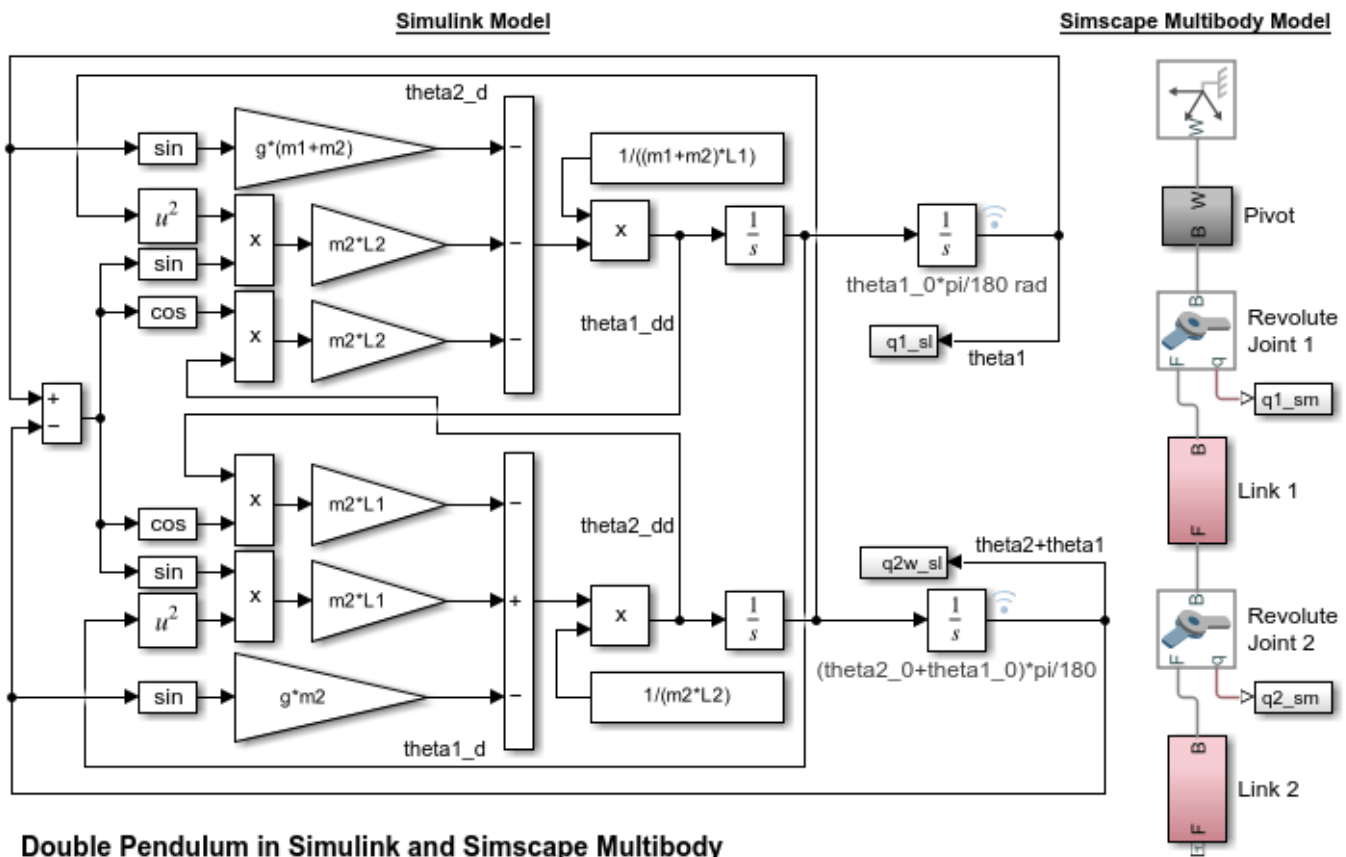
Double Pendulum in Simulink and Simscape Multibody

This example shows two models of a double pendulum, one using Simulink® input/output blocks and one using Simscape™ Multibody™.

The Simulink model uses signal connections, which define how data flows from one block to another. The Simscape Multibody model is built using physical connections, which permit a bidirectional flow of energy between components. Physical connections make it possible to add further stages to the pendulum simply by using copy and paste. Input/output connections require rederiving and reimplementing the equations.

The initial angle for each joint is defined by a MATLAB® variable. The annotations on the Integrator blocks show the initial angles of the joints with respect to the world frame.

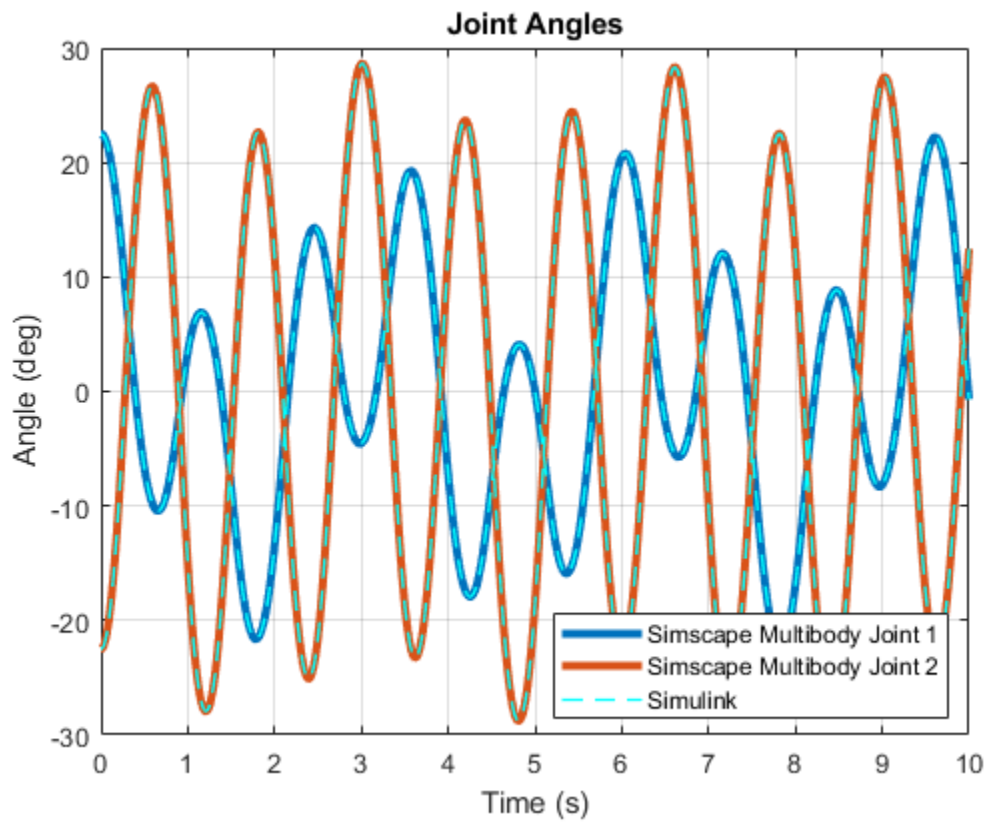
Model



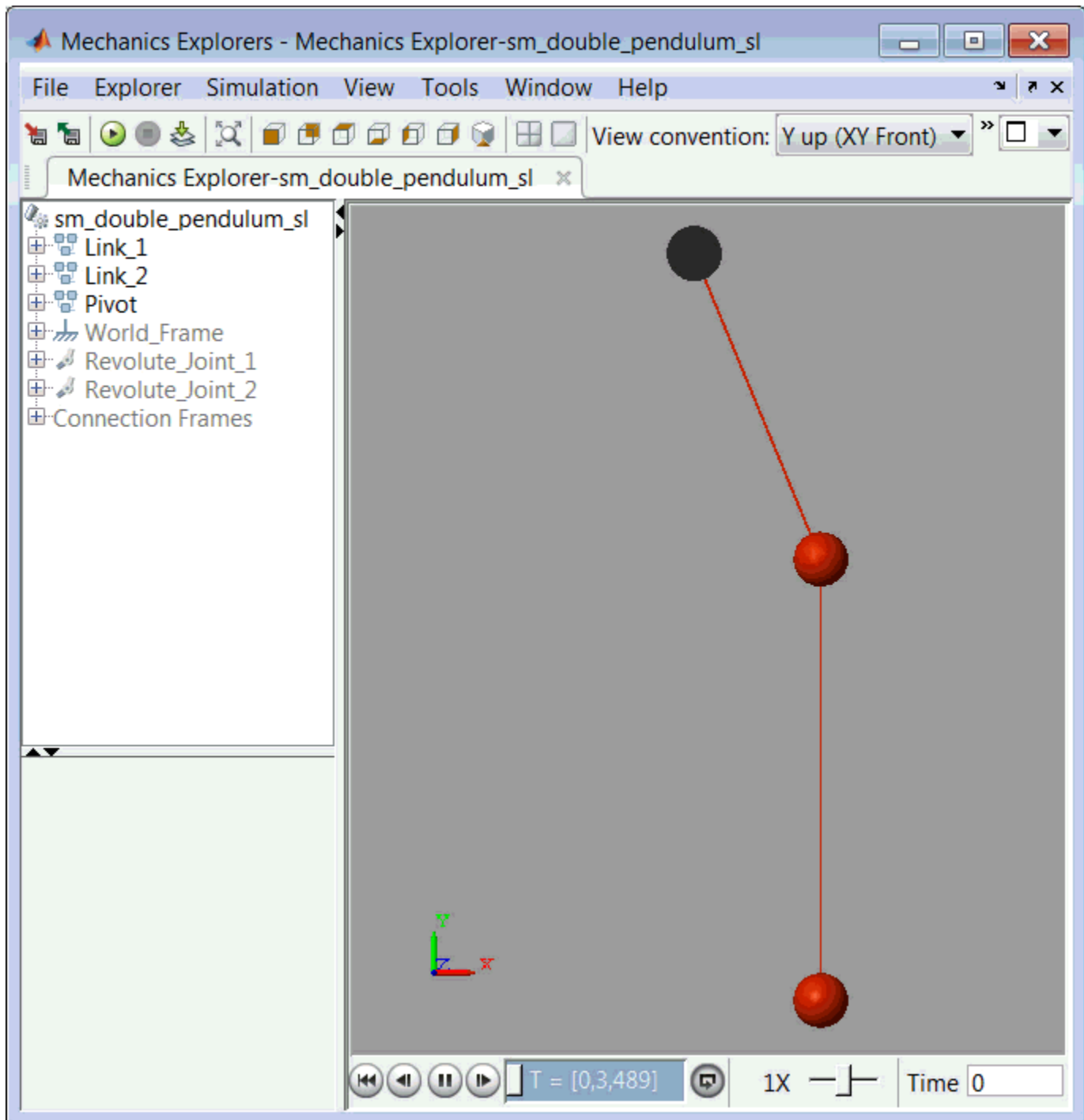
Double Pendulum in Simulink and Simscape Multibody

1. [Plot joint angles \(see code\)](#)
2. [Explore simulation results](#) using [Simscape Results Explorer](#)
3. [Learn more](#) about this example
4. [Open model](#) of single pendulum
5. Learn more about [multibody modeling](#)



Simulation Results from Simscape Logging

Mechanics Explorer Animation



See Also

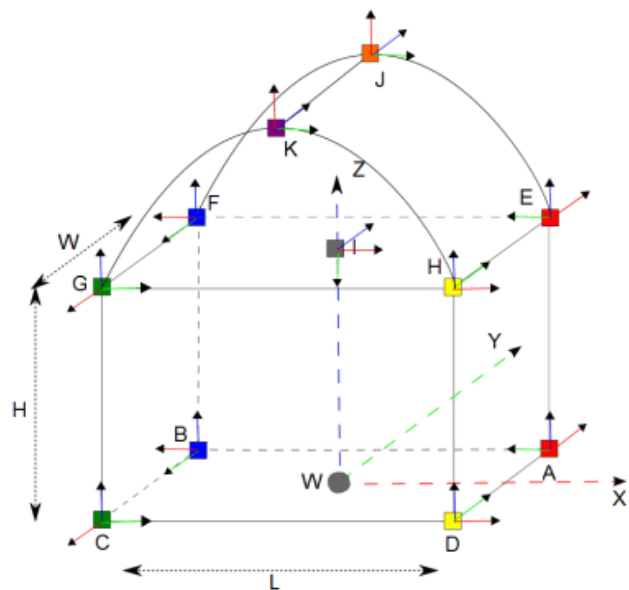
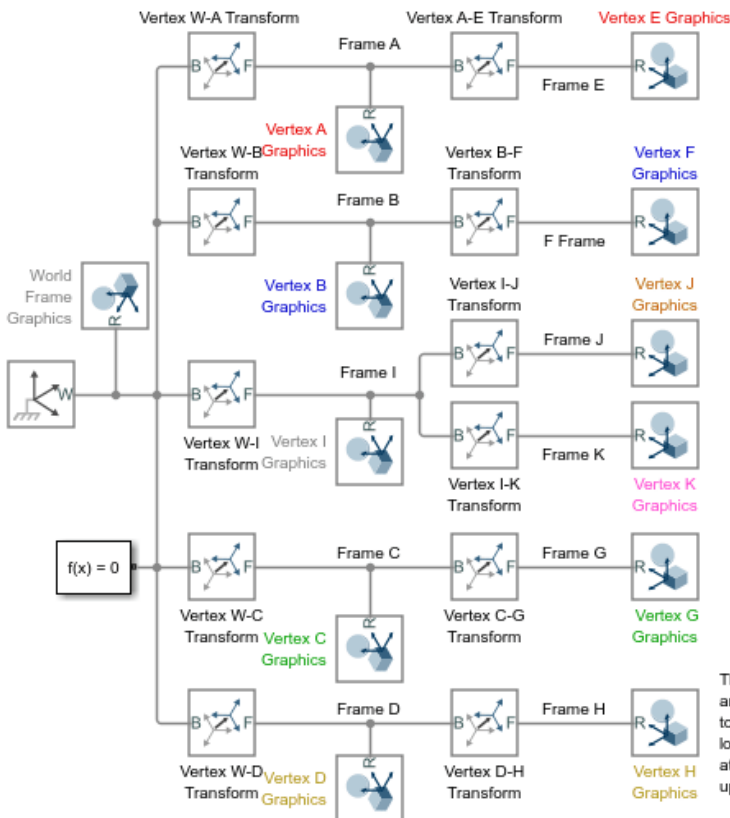
Planar Joint

More About

- “Assembling Parts into a Double Pendulum” on page 8-21

Creating Frames Using Rigid Transforms

This example shows the correspondence of coordinate frames to connection lines and frame ports. It highlights the Rigid Transform block as the fundamental method to rigidly relate nonidentical frames. The network reference frame is the world frame, located at the center of the bottom face of the cube. The other frames are the frames at the eight vertices of the cube, one at the center of the top face, and two on the upper curved section.



Creating Frames Using Rigid Transforms

This example shows the correspondence of coordinate frames to connection lines and frame ports. It highlights the Rigid Transform block as the fundamental method to rigidly relate nonidentical frames. The network reference frame is the world frame, located at the center of the bottom face of the cube. The other frames are the frames at the eight vertices of the cube, one at the center of the top face, and two on the upper curved section.

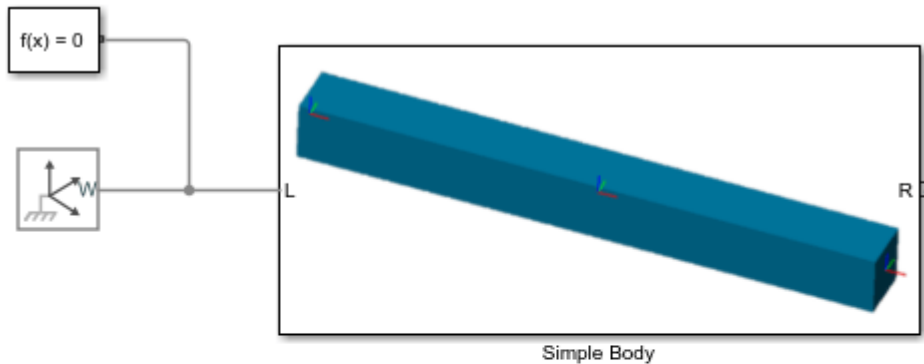
Model parameters are defined in the model workspace.

See Also

Rigid Transform | World Frame

Creating a Simple Part

This example shows the first step in modeling a rigid body. This model is a simple body (brick) with a frame at each of two ends and a reference frame at the center of mass. The inertia for the body is specified as Geometric Inertia with a constant density. The block automatically computes the appropriate inertia components. This serves as a first approximation of the actual rigid body. In subsequent iterations more detail can be added to obtain a more accurate model of the actual rigid body. See the example `sm_compound_body` for a more detailed model of the rigid body.



Creating a Simple Part

This example shows the first step in modeling a rigid body. This model is a simple body (brick) with a frame at each of two ends and a reference frame at the center of mass. The inertia for the body is specified as Geometric Inertia with a constant density. The block automatically computes the appropriate inertia components. This serves as a first approximation of the actual rigid body. In subsequent iterations more detail can be added to obtain a more accurate model of the actual rigid body. See the example `sm_compound_body` for a more detailed model of the rigid body.

Model parameters are defined in the model workspace.

See Also

Brick Solid

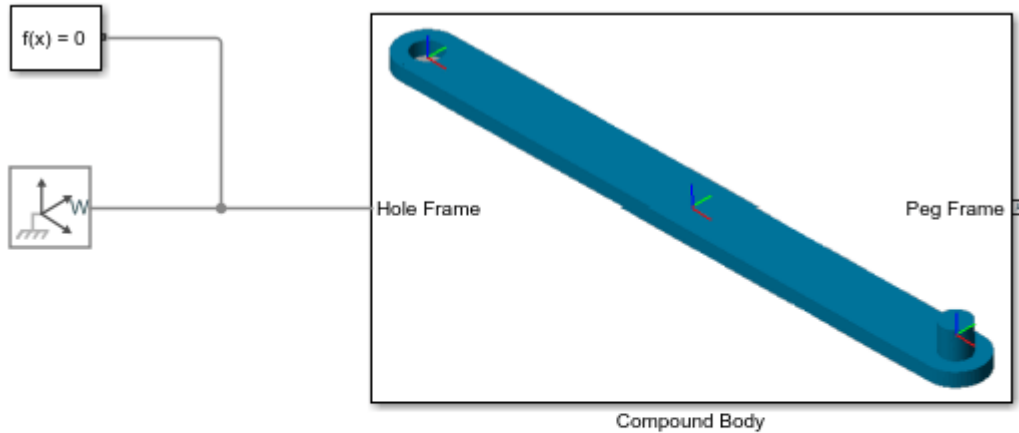
More About

- “Model a Simple Pendulum”

Creating a Complex Part

This example shows a moderately complex mechanical link with a frame at each end and a reference frame at the center. The link has a hole at one end and a peg at the other end. The link is a composite of three simple solids.

Model parameters are defined in the model workspace.



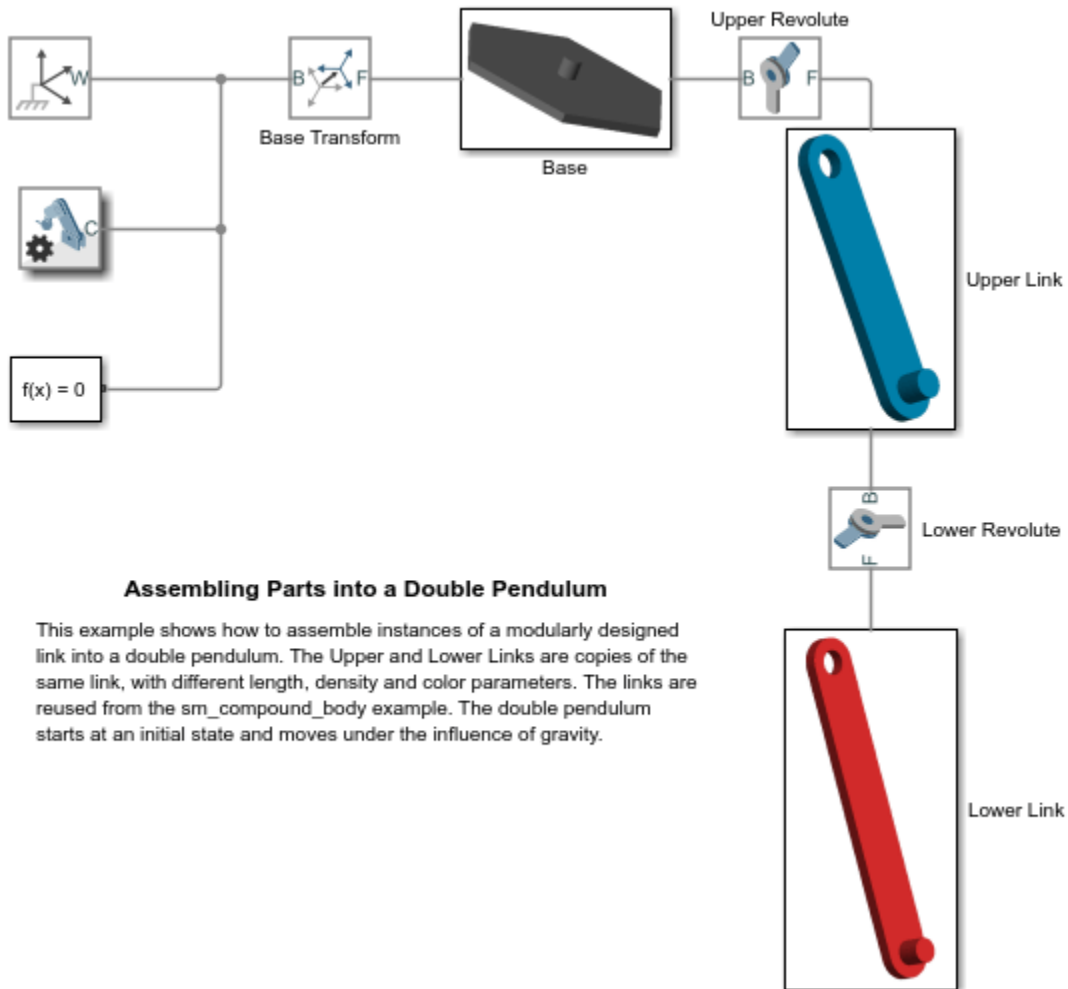
Creating a Complex Part

This example shows a moderately complex mechanical link with a frame at each end and a reference frame at the center. The link has a hole at one end and a peg at the other end. The link is a composite of three simple solids.

Model parameters are defined in the model workspace.

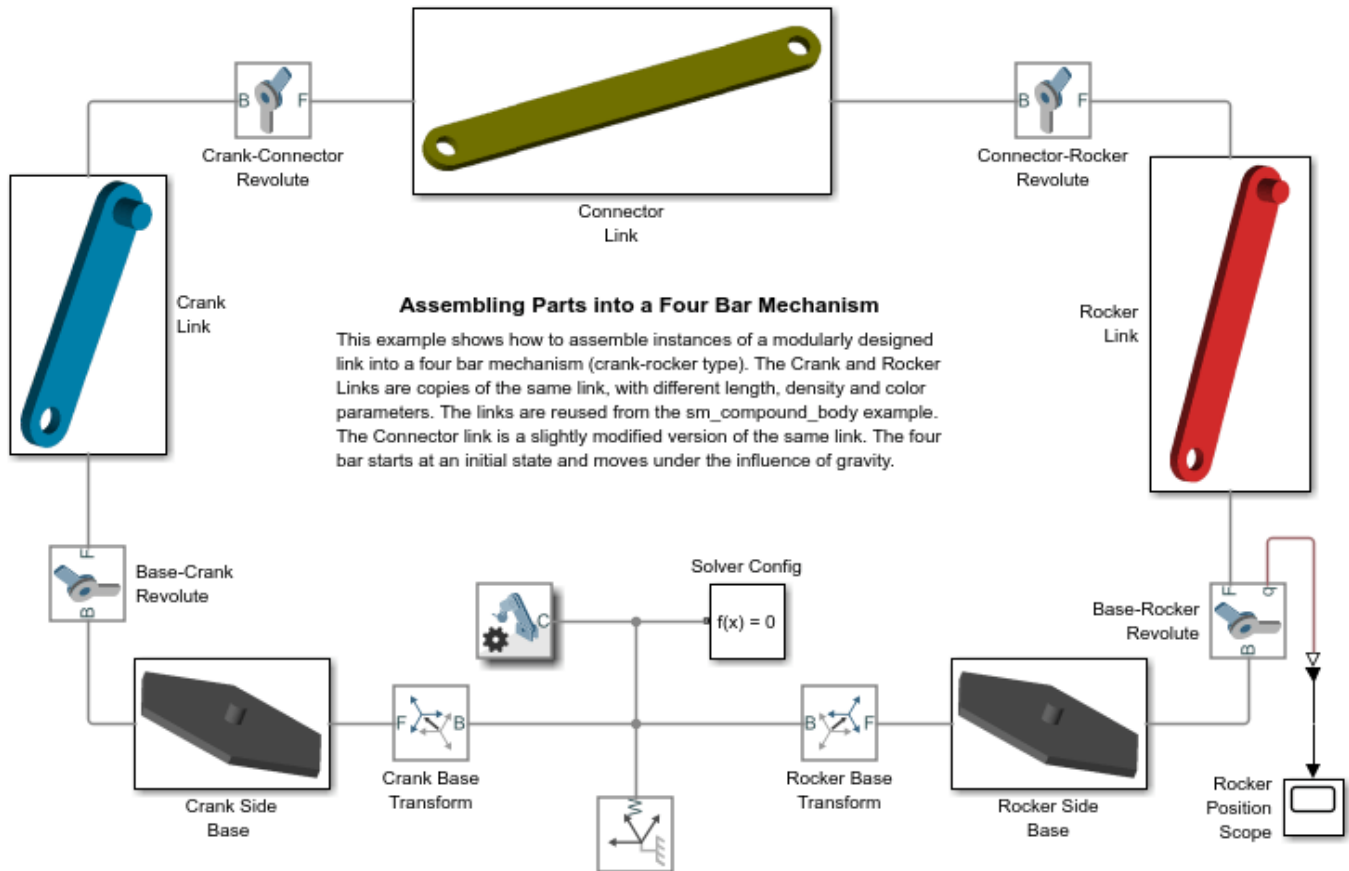
Assembling Parts into a Double Pendulum

This example shows how to assemble instances of a modularly designed link into a double pendulum. The Upper and Lower Links are copies of the same link, with different length, density and color parameters. The links are reused from the `sm_compound_body` example. The double pendulum starts at an initial state and moves under the influence of gravity.



Assembling Parts into a Four Bar Mechanism

This example shows how to assemble instances of a modularly designed link into a four bar mechanism (crank-rocker type). The Crank and Rocker Links are copies of the same link, with different length, density and color parameters. The links are reused from the `sm_compound_body` example. The Connector link is a slightly modified version of the same link. The four bar starts at an initial state and moves under the influence of gravity.

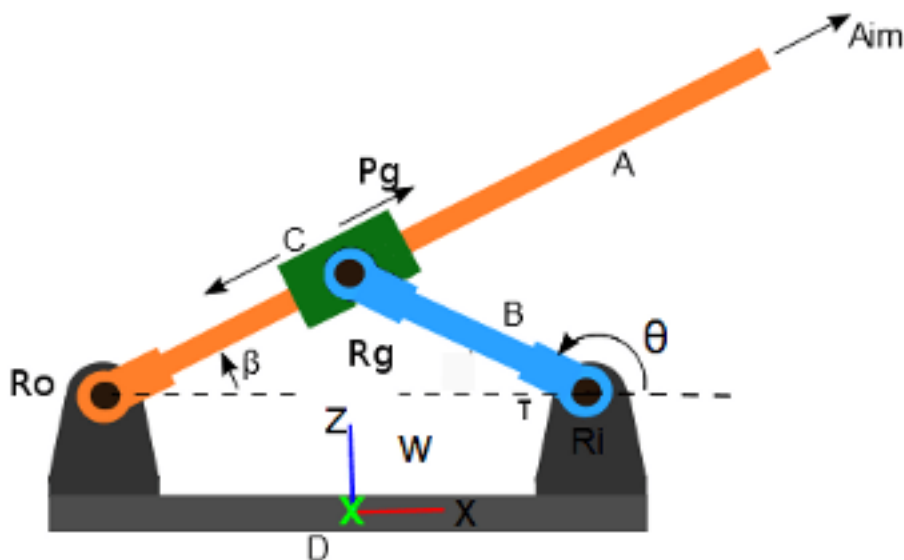


How to Build a Model

This example highlights key concepts and recommended steps for building a mechanical model using **Simscape™ Multibody™**. A simple design problem has been chosen to serve this purpose. The following section describes the design problem and subsequent sections discuss how to solve it.

Problem Description

The following figure shows a mechanism which functions as an aiming system.



The problem is simplified to aiming within the plane of the mechanism. The figure shows the schematic sketch of the mechanism and only captures the essentials of how the mechanism operates (which is usually the case during the early stages of a design process). The link **C** can slide on the link **A**. A motor applies torque τ at the revolute joint **Ri** and the task is to track a particular trajectory of the revolute angle β .

Building the Model

A key principle to follow while building models is to begin with a simple approximation to get the basic mechanism working. In subsequent iterations add complexity to the model. The recommended model building process in Simscape Multibody can be broken down into the following steps:

- 1 Identify the rigid bodies in the mechanism.
- 2 Identify how the rigid bodies are connected to each other (joints, constraints etc).
- 3 Consider each rigid body in isolation. Build a simple approximation of the rigid body, and define the frames rigidly attached to it.
- 4 Assemble the rigid bodies using joints and/or constraints. Utilize the **Model Report** to identify any issues with the model assembly.

- 5 Utilize the **Mechanics Explorer** to identify and fix other issues with the model.
- 6 Set **Joint Targets** to guide assembly to desired configuration.
- 7 Hook up inputs and outputs to the mechanism. Test and Validate the model. If applicable, attach a controller and test the model.
- 8 Add detail to the individual rigid bodies to make the model a more accurate representation of the actual mechanism.

The following sections describe these steps in more detail.

Identifying Rigid Bodies and Joints

The mechanism has four rigid bodies

- **Rigid Body A** (orange)
- **Rigid Body B** (blue)
- **Rigid Body C** (black)
- **Rigid Body D** (grey)

The mechanism has the following joints

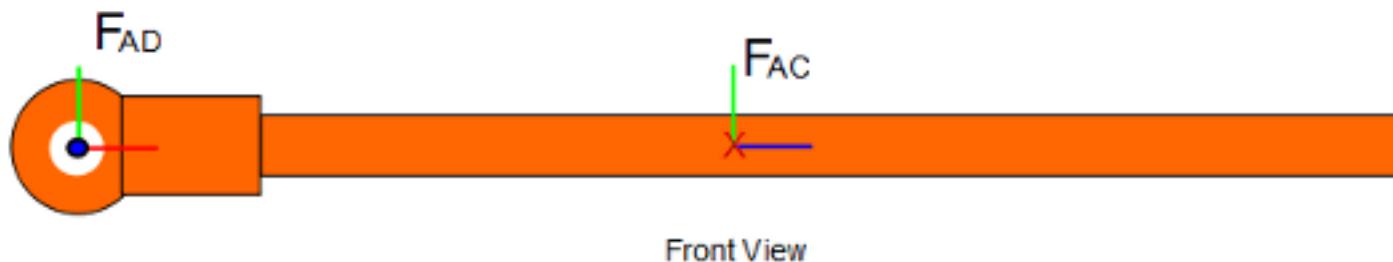
- Rigid bodies A and D are connected via a revolute joint **Ro**.
- Rigid bodies A and C are connected via a prismatic joint **Pg**.
- Rigid bodies C and B are connected via a revolute joint **Rg**.
- Rigid bodies B and D are connected via a revolute joint **Ri**.

In addition, the rigid body **D** is rigidly connected to the world frame **W** since it is motionless.

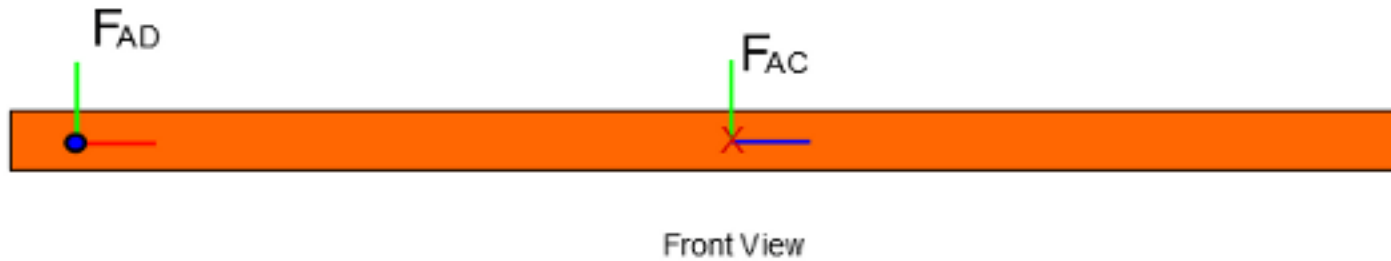
Defining the Rigid Bodies and their Interface

You define a rigid body by specifying its shape, mass properties and interface with other parts. Each rigid body is identified and defined in isolation. In the above example, the mechanism is composed of four rigid bodies: **A**, **B**, **C** and **D**.

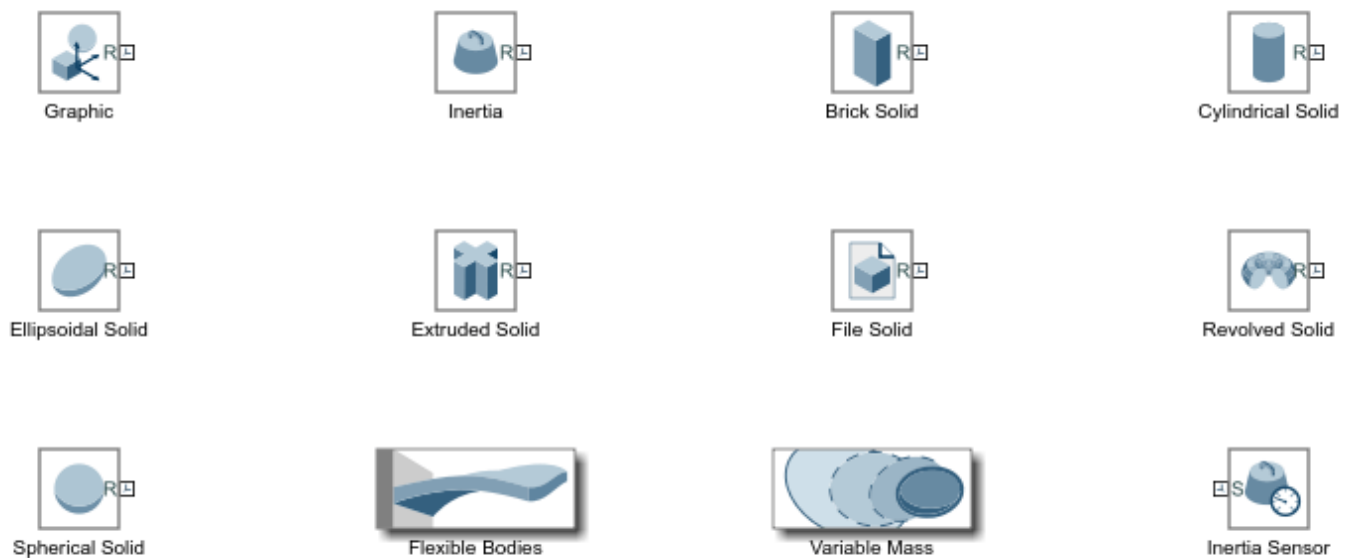
The rigid body **A** is shown in isolation below.



First, define the shape of the rigid body **A** in Simscape Multibody. Once the shape of the object is defined and its density is specified, Simscape Multibody can compute the inertia automatically. Instead of defining the fairly complicated shape shown above, as a first approximation, you can define the shape of the rigid body as a simple cylinder with a length equal to that of the original part.

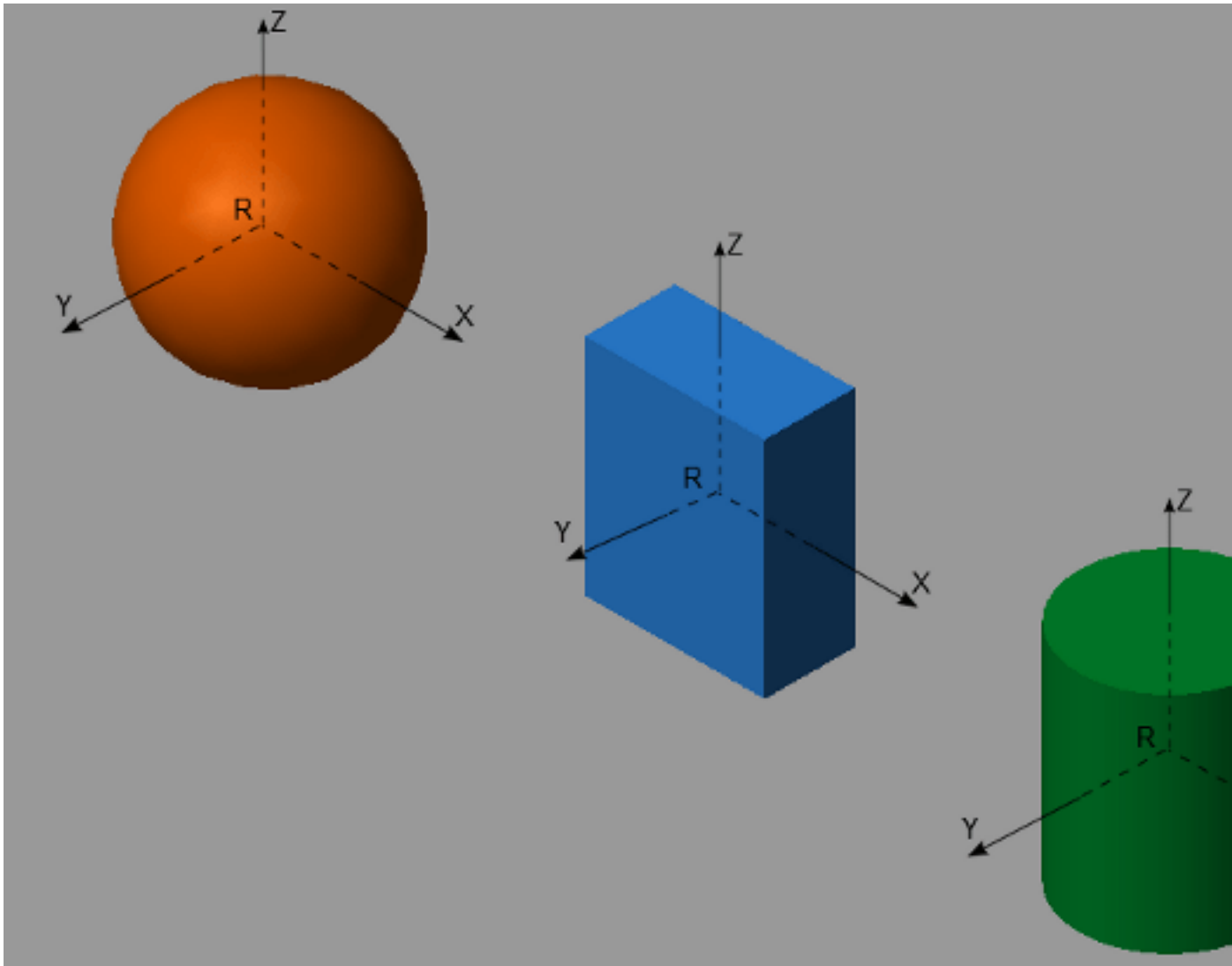


Once you have defined the shape (first approximation) of the rigid body **A**, specify its density. Simscape Multibody now has enough information to compute the inertial properties required for dynamic simulation. In Simscape Multibody, you define a simply shaped rigid body using the **Solid** block.



Body Elements Library

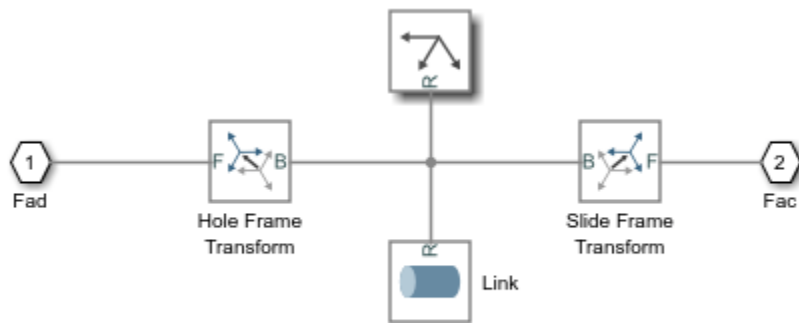
The **Solid** block lets you define simple solids with fixed parameterizations. These include: bricks, cylinders, polygonal extrusions, regular prisms, spheres, ellipsoids, etc. You define each parameterized solid with respect to a reference frame. The diagram below shows the reference frames for some of the parameterized solids. The **Solid** block exposes this reference frame as a frame port labelled "R" on the block.



The interface of a rigid body is established by defining frames attached to the rigid body. A rigid body is connected to other parts of the mechanism via the rigidly attached frames. In Simscape Multibody, joints establish a time-varying relationship between two frames. For instance, the **Revolute Joint** establishes the relationship that the **Z**-axes of the attached frames are parallel and the origins of the frames are coincident. The **Prismatic Joint** establishes the relationship that the **Z**-axes of the attached frames are collinear and the **X** and **Y** axes are always parallel. Note that the frames themselves are defined independently of the joint; the joint only establishes a relationship between the already existing frames. Note also that the **Z**-axis is the axis of rotation in the case of the revolute joint and is the axis of sliding in the case of the prismatic joint. This information is essential when we define the interface of a rigid body by defining the frames rigidly attached to it.

In this example, the rigid body **A** has a cylindrical hole at one end that fits onto a peg so that **A** can rotate about the axis of the cylindrical hole. This suggests that a frame should be defined at the hole center with its **Z**-axis aligned with the axis of the hole (the axis of rotation). This frame is labelled as F_{AD} above. The choice of orientation of the **X** and **Y** axes of F_{AD} partly determines the zero configuration of the joint to which F_{AD} would be connected (see discussion on Zero Configuration below). **A** also acts as the shaft on which part **C** slides. This suggests that a frame should be defined

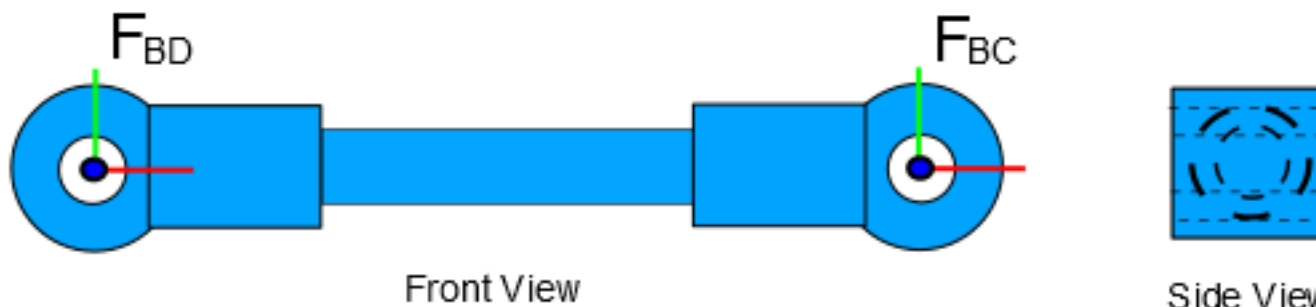
at the center of **A** (an arbitrarily selected position) with its **Z**-axis aligned along the length of **A** (along the direction of sliding). This frame is labelled as F_{AC} above. The frames F_{AD} and F_{AC} define the interface for the rigid body **A**. The model `sm_dcrankaim_approx_body_A` shows how the **Solid** and **Rigid Transform** blocks have been used to define the shape, inertia and interface of the rigid body **A**. The **Body A Ref** is a **Reference Frame** block. This block is not required but serves as a modeling convenience that fixes a certain frame as the frame to which other frames are referenced to. The frames F_{AC} and F_{AD} are defined with respect to the frame to which the **Body A Ref** block is connected. For a more complicated network of blocks defining a rigid body, such a reference frame serves as a starting point for defining the positions and orientations of all other frames.



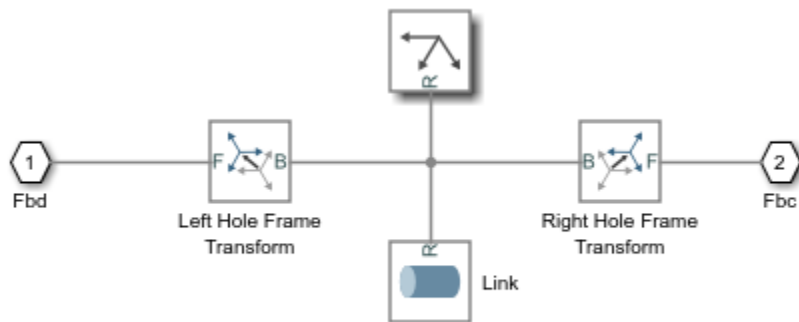
Components that make up rigid body A

Run the model `sm_dcrankaim_approx_body_A` to visualize the model in the **Mechanics Explorer**.

Consider the rigid body **B**. The shape of the rigid body can again be approximated with a simple cylinder. The rigid body has cylindrical holes at both ends that fit onto pegs. The rigid body **B** can rotate about either hole axis. This suggests that two frames should be defined: one at each hole center with its **Z**-axis aligned with the axis of the hole.



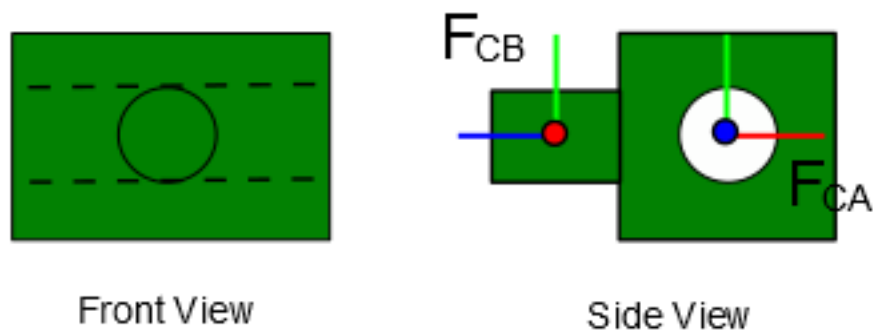
The model `sm_dcrankaim_approx_body_B` shows how the **Solid** and **Rigid Transform** blocks have been used to define the shape, inertia and interface of the rigid body **B**.



Components that make up rigid body B

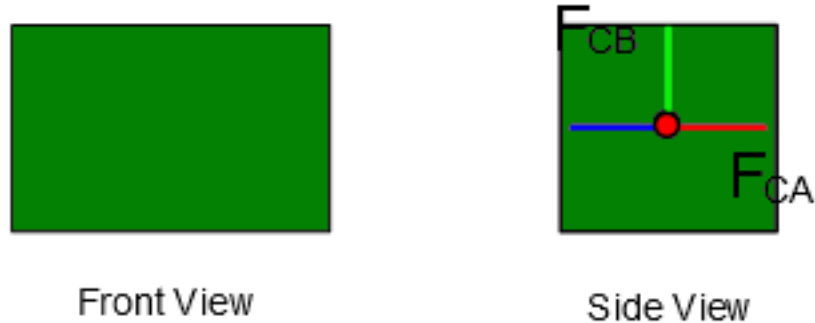
Run the model `sm_dcrankaim_approx_body_B` to visualize the model in the **Mechanics Explorer**. A similar approach can be taken for building a first approximation of the rigid body **D**.

Consider the rigid body **C**.

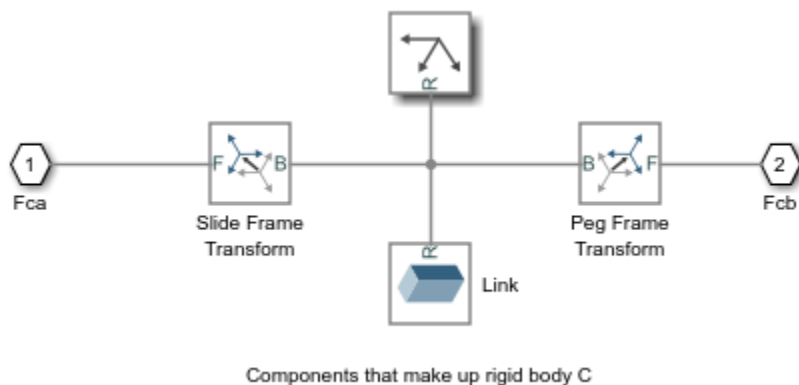


This rigid body has a cylindrical hole that slides on a peg. It also has a peg about which another body can rotate. This suggests the need to define two frames: one at the center of the hole with its **Z**-axis along the axis of the hole, and the other at the center of the peg with its **Z**-axis along the axis of the peg. These are marked as F_{CA} and F_{CB} above.

The shape of the rigid body **C** can be approximated with a simple cuboid. In the first approximation of the rigid body, the offset between the origins of frames F_{CB} and F_{CA} can also be made zero. This results in the simplified representation of rigid body as shown below.



The model **sm_dcrankaim_approx_body_C** shows how the **Solid** and **Rigid Transform** blocks have been used to define the shape, inertia and interface of the rigid body **C**.



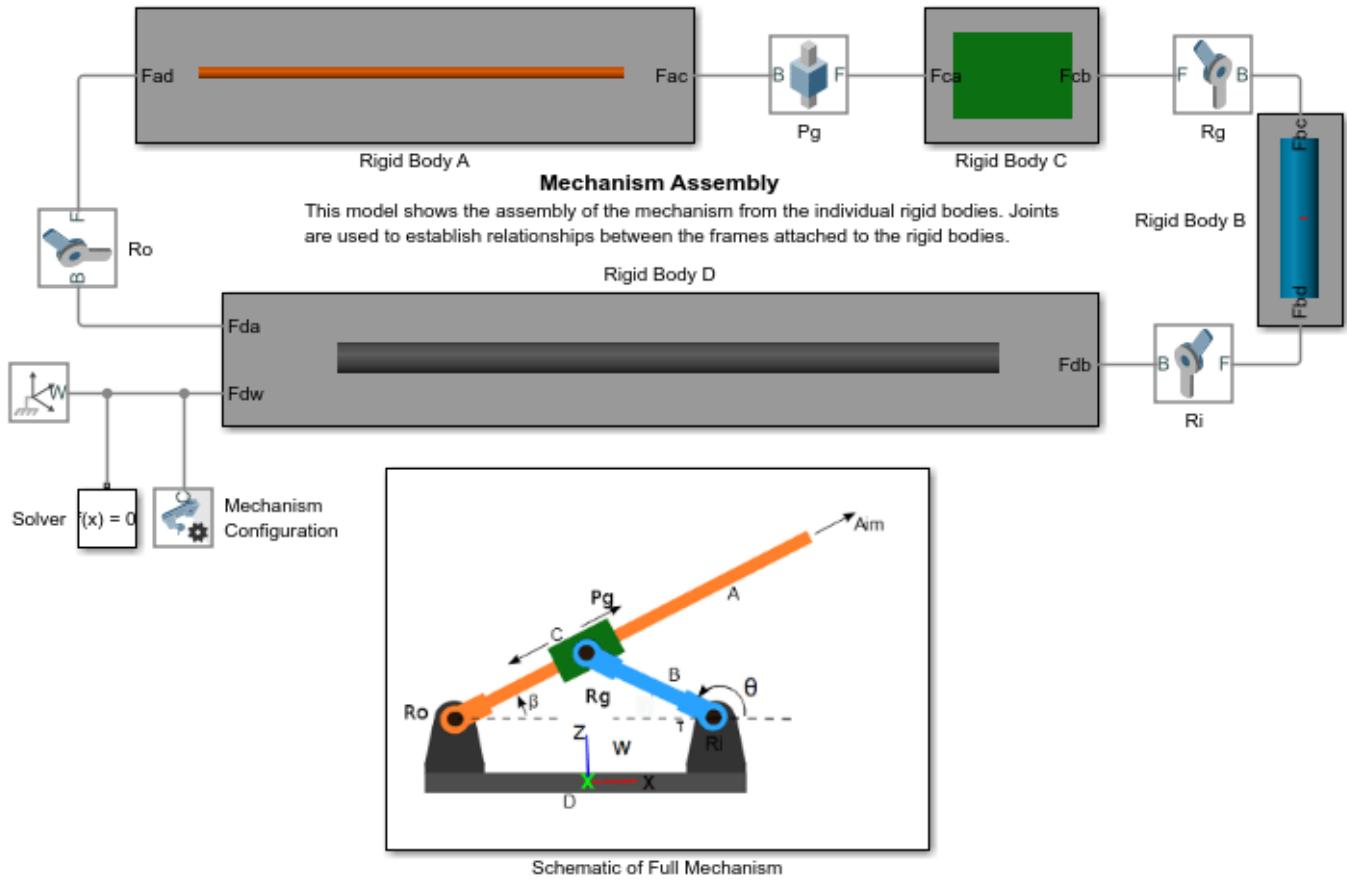
Assembling the Individual Bodies Using Joints

All the individual bodies were built in isolation. The process of assembly involves establishing relationships (using joints) between the frames attached to the rigid bodies. The following joints establish all of the necessary relationships between the frames to assemble the mechanism.

- A **Revolute Joint** between the frames F_{DA} and F_{AD}
- A **Prismatic Joint** between the frames F_{AC} and F_{CA}
- A **Revolute Joint** between the frames F_{CB} and F_{BC}
- A **Revolute Joint** between the frames F_{BD} and F_{DB}

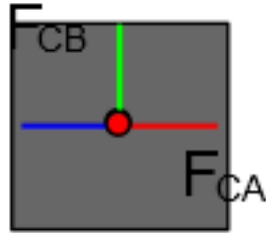
The effort that went into carefully defining the interfaces of all of the rigid bodies (i.e. the frames attached to them) makes it very easy to complete the mechanism by simply adding joints between the appropriate frames. There is no need to customize the joints to achieve a default assembly of the mechanism. The resulting assembly may or may not be in the desired configuration since the

mechanism can be assembled into multiple configurations. The model `sm_dcrankaim_assembly_with_error` shows the assembled mechanism.

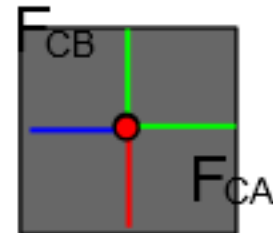


Using the Model Report to Identify Problems

In the model an intentional mistake has been made in the definition of the frame F_{CA} attached to rigid body C. This causes the assembly to fail. The figure below shows the desired and actual orientations of the frame F_{CA} .



Desired Orientation of Fca

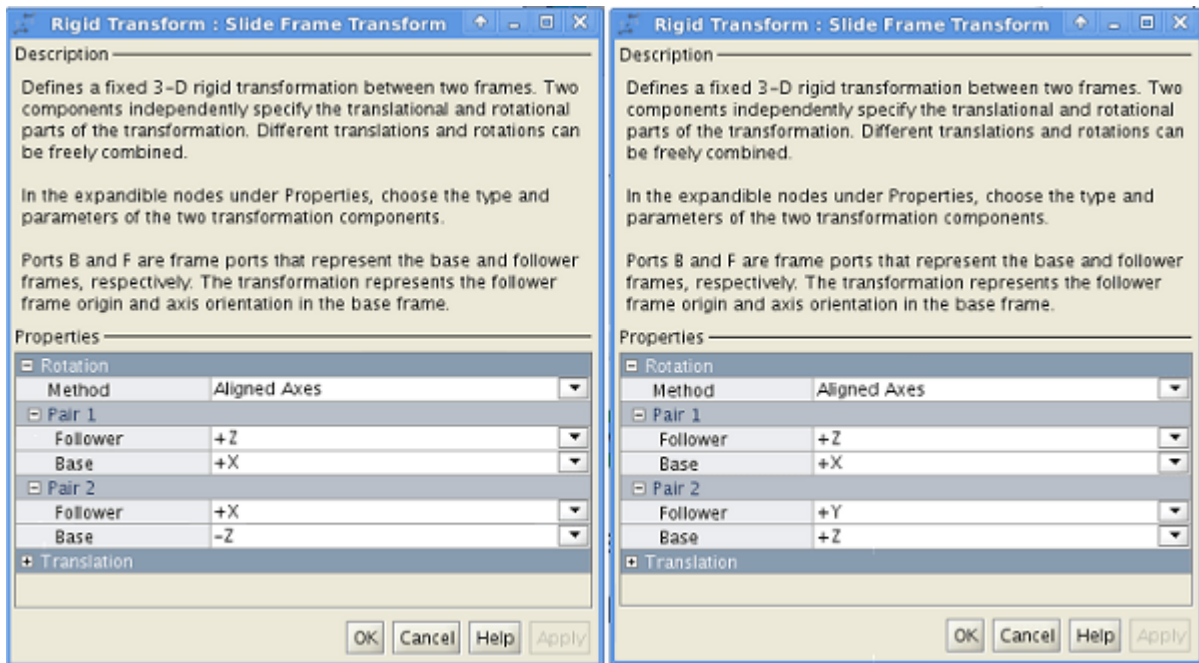


Actual Orientation of Fca

The orientation of F_{CA} has to be corrected by a rotation of 90 deg about the **Z**-axis. Update the model (Ctrl-D) **sm_dcrankaim_assembly_with_error** to visualize the mechanism. An error is reported indicating that the assembly failed. In the **Mechanics Explorer**, select **Model Report** option from the **Tools** pulldown menu. In the **Model Report** the **Joints** section will show that the joint **Pg** has failed to assemble. This indicates that there might be an error in the specification of the frames attached to the joint **Pg**. In this example it is in fact true that an error was made in the specification of the frame F_{CA} .

Joint	Assembled	Primitive	Position				Velocity					
			Actual	Specified	Unit	Priority	Status	Actual	Specified	Units	Priority	Status
Pg	<input checked="" type="checkbox"/>	Pz	N/A		m			N/A		m/s		
Rg	<input checked="" type="checkbox"/>	Rz	-0.0044...		deg			+0		deg/s		
Ri	<input checked="" type="checkbox"/>	Rz	+0.007...		deg			+0		deg/s		
Ro	<input checked="" type="checkbox"/>	Rz	+0.003...		deg			+0		deg/s		

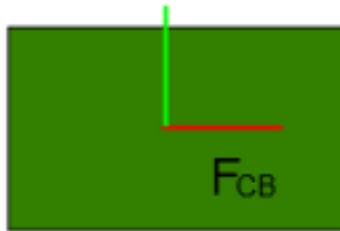
Changing the parameters of the rigid transform **sm_dcrankaim_assembly_with_error/Rigid Body C/Slide Frame Transform** as shown below fixes the problem allows the assembly to succeed.



Zero Configuration of Joints

The **Zero Configuration** of a joint is defined as the relative position and orientation between the base and follower frames when all of the joint angles are zero. For almost all of the joints in Simscape Multibody, the base and follower frames are identical in the zero configuration: their origins are coincident, and their axes are aligned. One defines the relative position and orientation between two bodies connected by a joint when the joint angles are zero by adjusting the positions and orientations of the base and follower frames on their respective bodies.

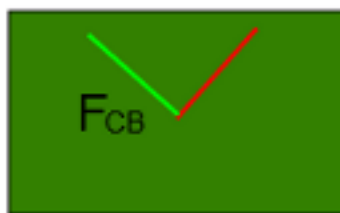
Consider, for example, the rigid bodies **B** and **C** and the joint **R_g** connecting them. The frames F_{CB} and F_{BC} are the base and follower frames of the joint **R_g**. The figure below shows how different choices of orientations for the frame F_{CB} attached to rigid body **C** result in different assembled configurations when the joint angle is zero. The choice of orientation of the frames must be made with the desired zero configuration in mind.



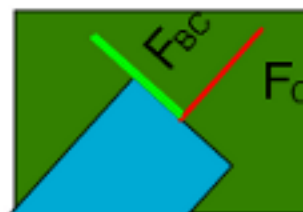
Revolute joint Added Between F_{CB} and F_{BC}



Assembled Zero Configuration



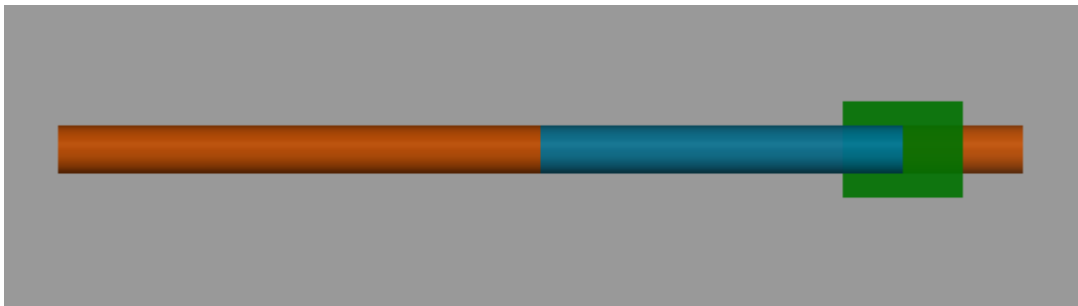
Revolute joint Added Between F_{CB} and F_{BC}



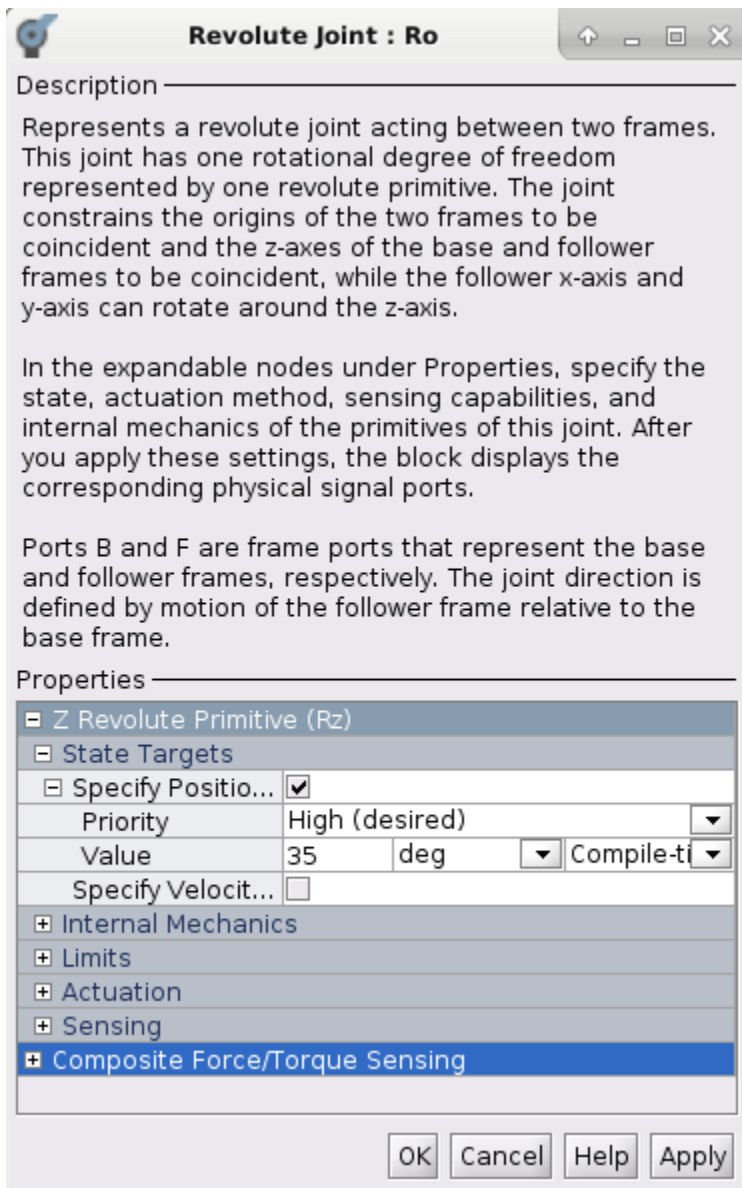
In the aiming mechanism, the choice of frame orientations leads to a default assembled configuration in which the central axes of all of the bodies lie along the same line.

Guiding Assembly Using Joint Targets

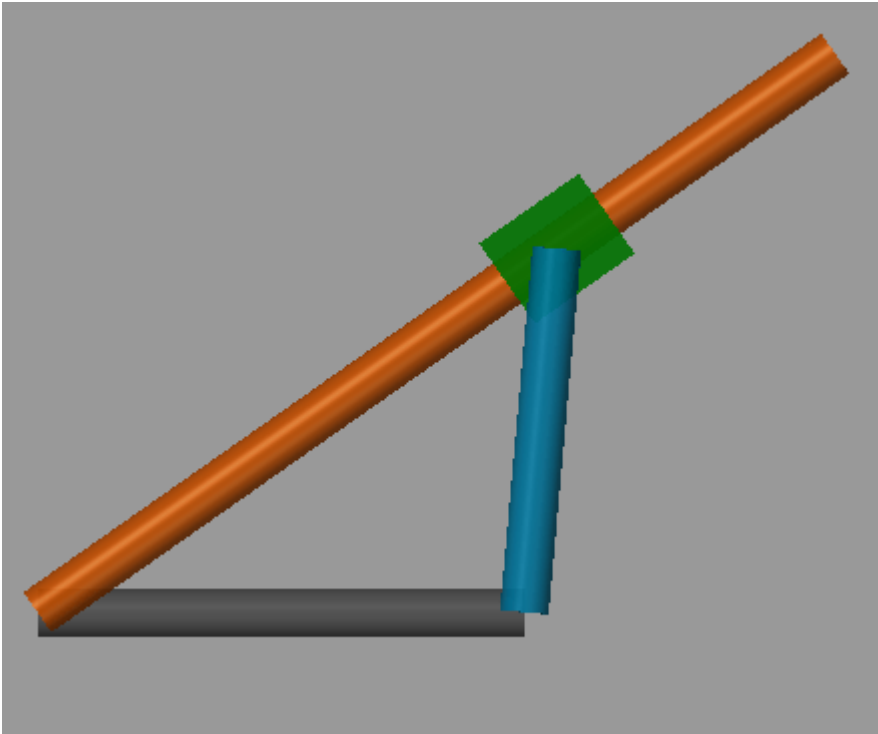
Update the model (Ctrl-D) `sm_dcrankaim_assembly_with_error` (after the errors have been fixed) to visualize the assembled mechanism. It can be seen that all of the bodies are collapsed onto a common line; this is the default assembly configuration. In this configuration, all of the revolute joint angles are zero. Thus, the base and follower frames of each revolute joint are coincident and aligned with each other; the corresponding frame pairs are: F_{DA} and F_{AD} , F_{CB} and F_{BC} and F_{BD} and F_{DB} . In contrast, the frame F_{CA} is translated from frame F_{AC} , thus the joint **Pg** is not in its zero state. Open the **Model Report** to see the values of the joint positions in this assembled configuration. This is not a desirable assembly configuration.



The configuration depicted in the schematic diagram of the mechanism is the desired initial assembly configuration. From the schematic diagram we can see that in the initial configuration the angle β is about 35 deg. The assembly algorithm can be guided by specifying joint position and velocity targets. In this example, the position target for the joint **Ro** can be set to guide assembly into the desired initial configuration (see figure below). The target priority has been set as **High**. Since this is the only target in the model, Simscape Multibody is able to achieve it exactly.

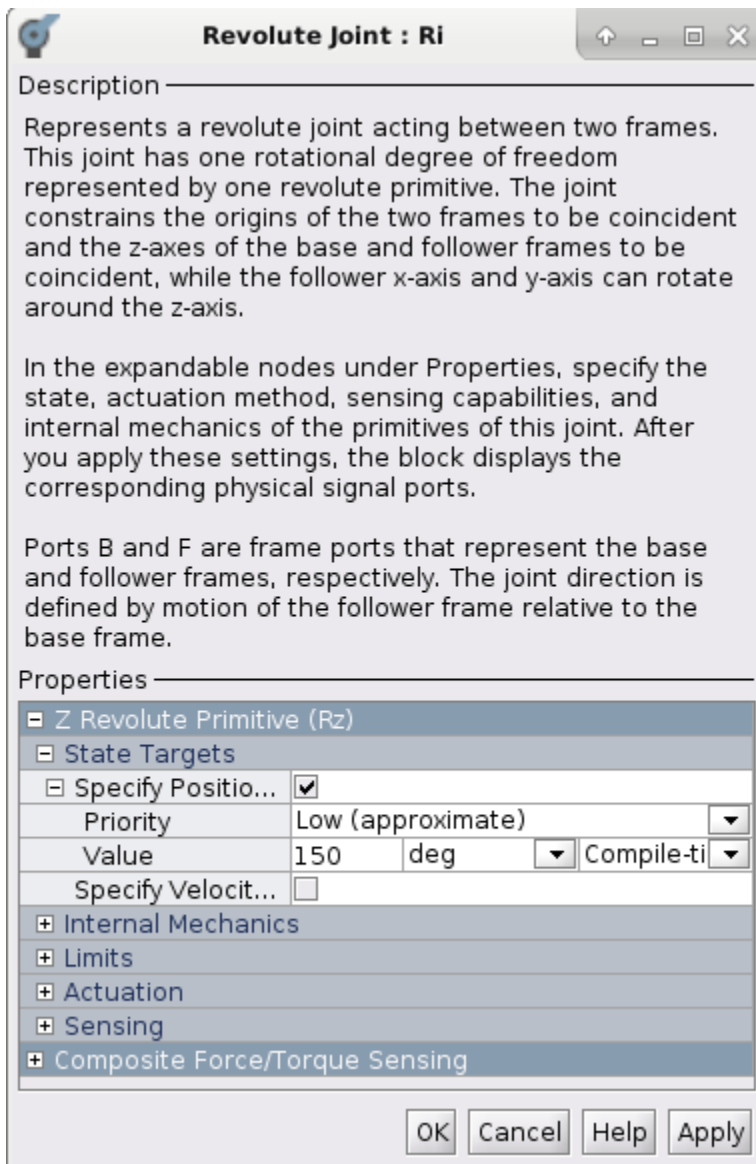


Update the model (Ctrl-D) to update the visualization with the changes. The assembly is closer to the configuration in the schematic diagram. Check the **Model Report** to see that the joint target for **Ro** has been met exactly.

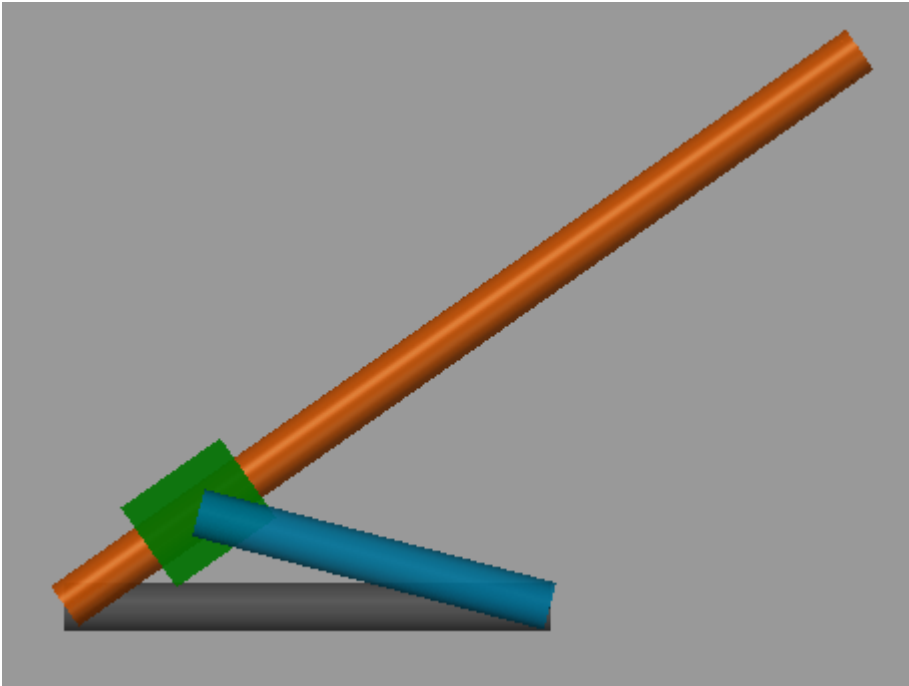


Unfortunately, the assembled configuration is not the intended one because the rigid body **B** is not aligned as indicated in the schematic diagram. Attempting to specify the joint angles of both ***R_o*** and ***R_i*** exactly is an over-specification for this one degree-of-freedom mechanism. This is not prohibited, but if there is a conflict, neither target may be met. Moreover, the desired angle of joint ***R_i*** is not even known exactly.

In this situation, a convenient approach is to leave the high-priority target of 35 deg on ***R_o*** but to specify the angle of ***R_i*** through a low-priority position target. The latter provides an approximate value, or hint, for the desired joint angle. In this case, it is obvious that the angle θ should be obtuse; 150 deg is a rough estimate of its desired value. This target is set for joint ***R_i*** with a priority of **Low**.



The assembled configuration after setting the new target is shown below.

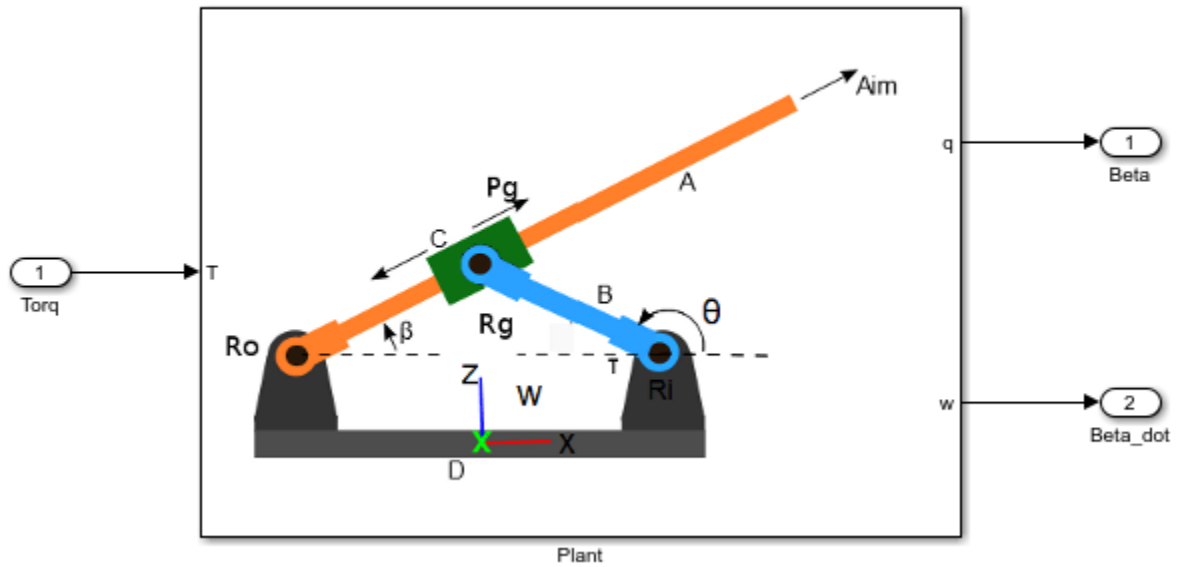


Simulate the model (Ctrl-T) to view the motion of the mechanism under gravity.

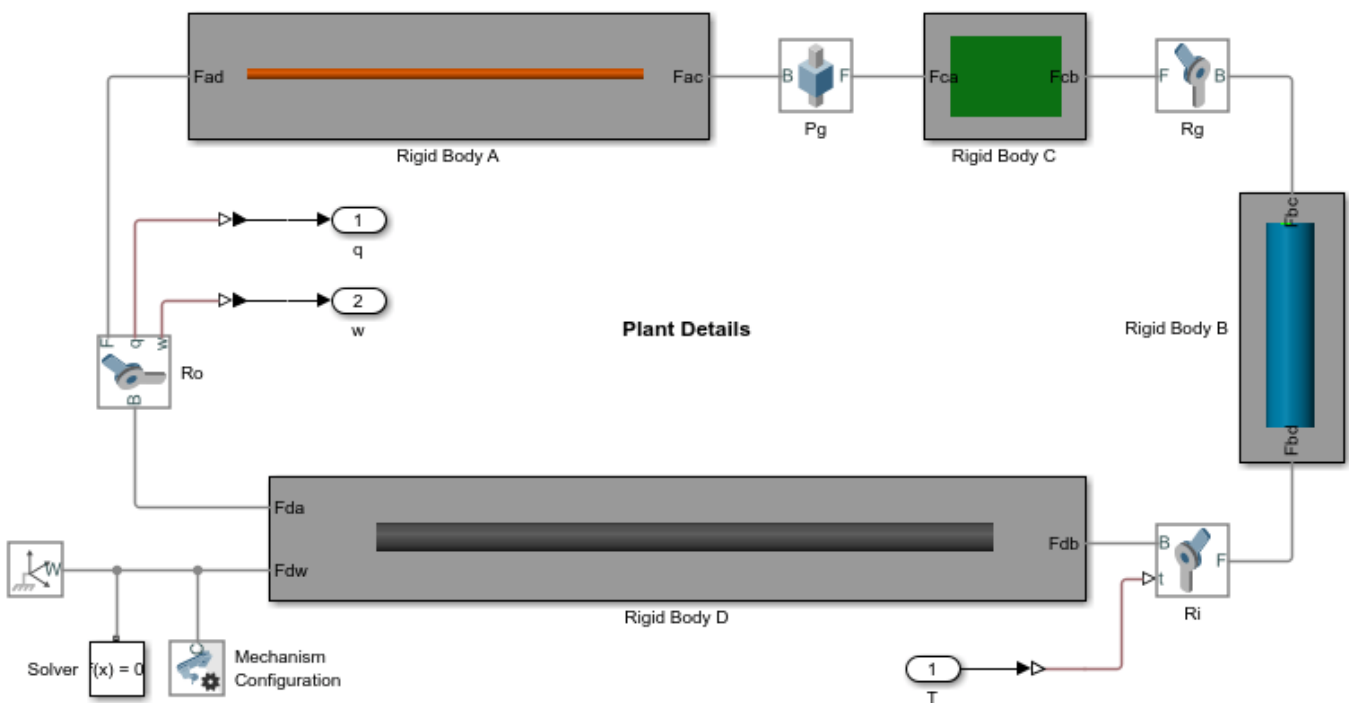
Setting up the Model for Control Design

In this example, the goal is to make the angle β track a desired trajectory by applying a torque at the joint **Ri**. The joint **Ri** will be torque actuated and the joint angle β and its derivative (angular velocity) will be sensed from the joint **Ro**. The entire mechanism can be enclosed within a subsystem that takes a torque input and outputs the angle β and angular velocity $\dot{\beta}$. This subsystem is the canonical **Plant** in Control Design parlance. The model `sm_dcrankaim_plant` shows the mechanism setup for control design.

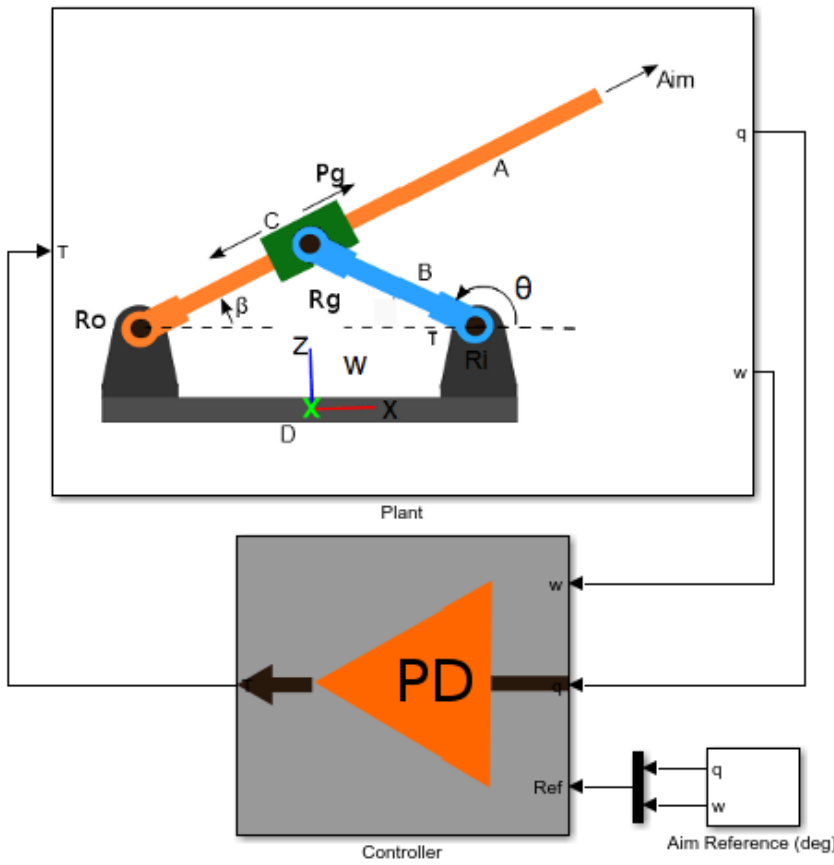
Mechanism Setup for Control Design



The details of the **Plant** subsystem is shown below.

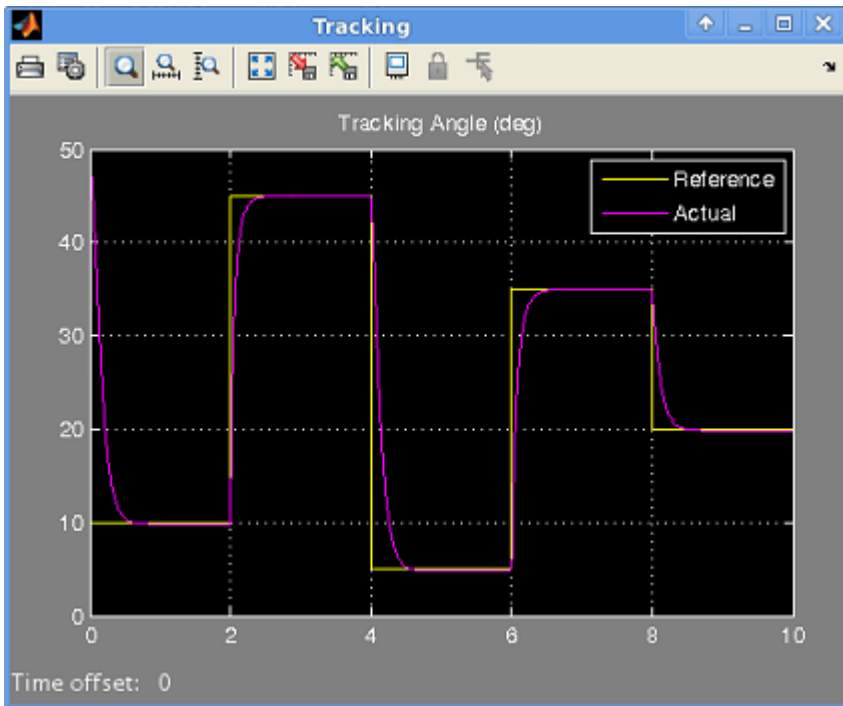
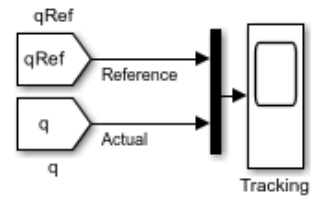


The model **sm_dcrank_aiming_mechanism_v1** shows the **Plant** hooked up to a **Controller**. The tracking performance of the controller can be viewed in the scope. A simple PD controller has been designed to achieve tracking.



Double Crank Aiming Mechanism

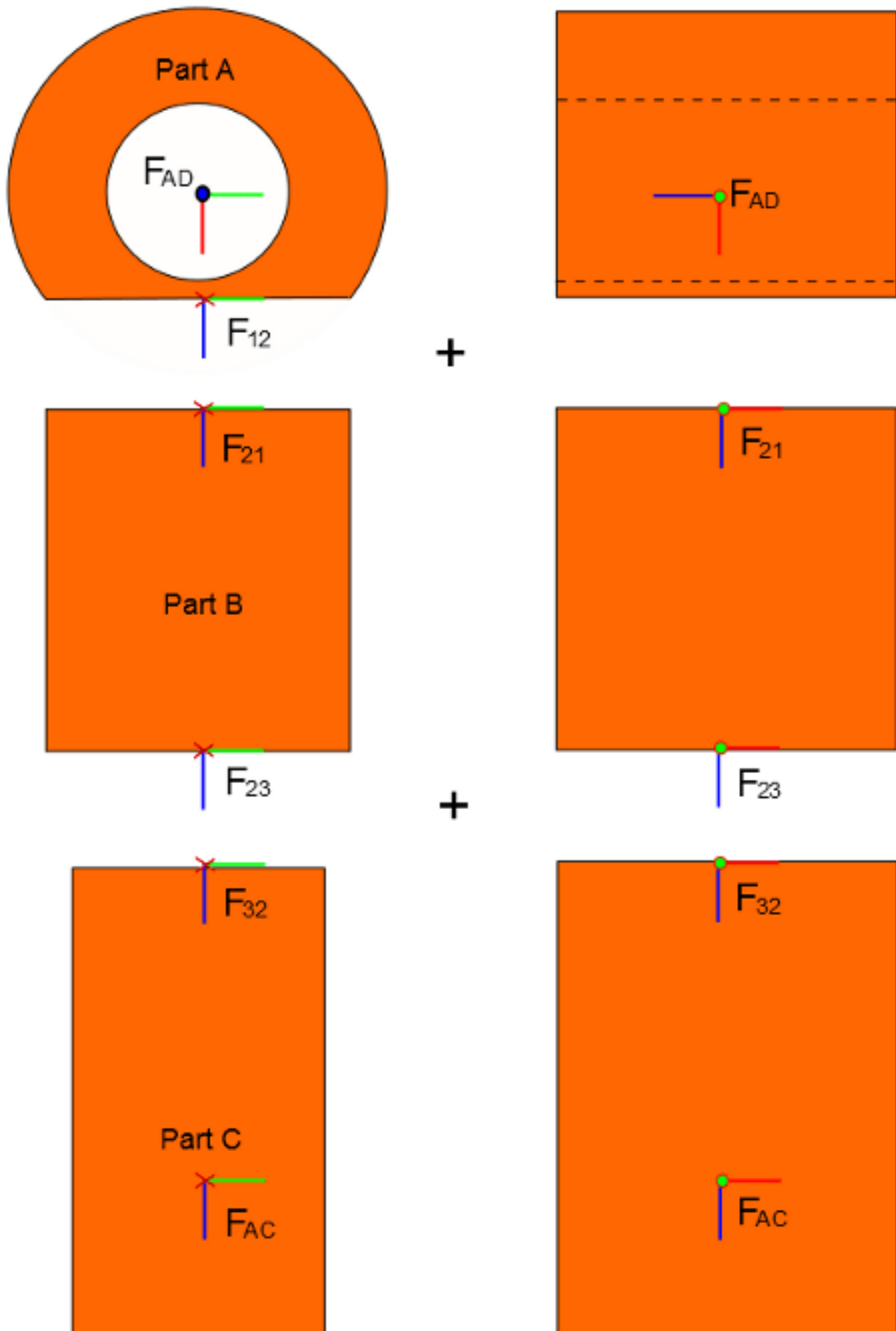
Aiming mechanism that points link A at a certain angle (beta). The mechanism is controlled by a PD controller that exerts a control torque at joint Ri.



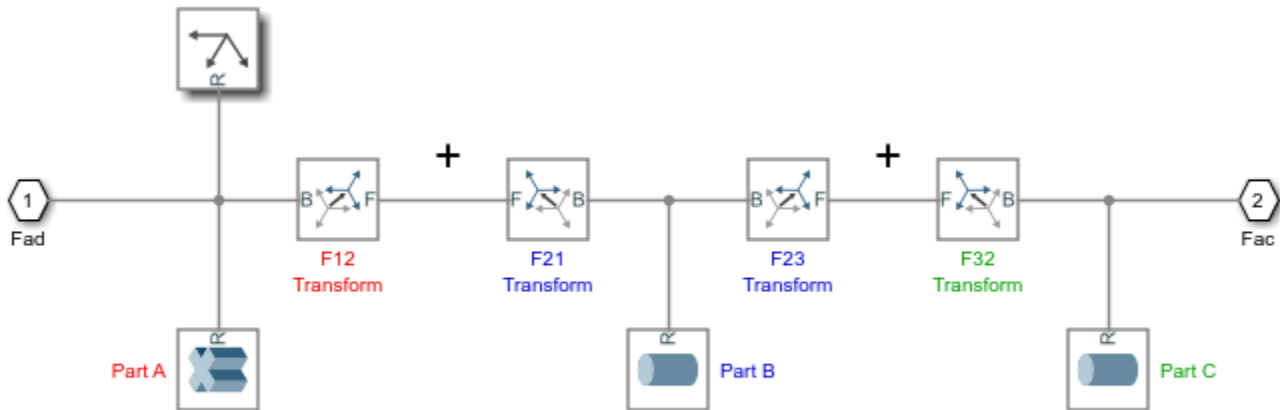
Adding Detail to the Rigid Bodies

Now that the basic model is working, the next step is to add detail to make the model more realistic and accurate. Perhaps the first version of the model was created when detailed information about the geometry of the rigid bodies was not yet available. Having carefully established the interfaces of the rigid bodies, it is fairly easy to add detail to each of the rigid bodies without affecting/changing the rest of the model.

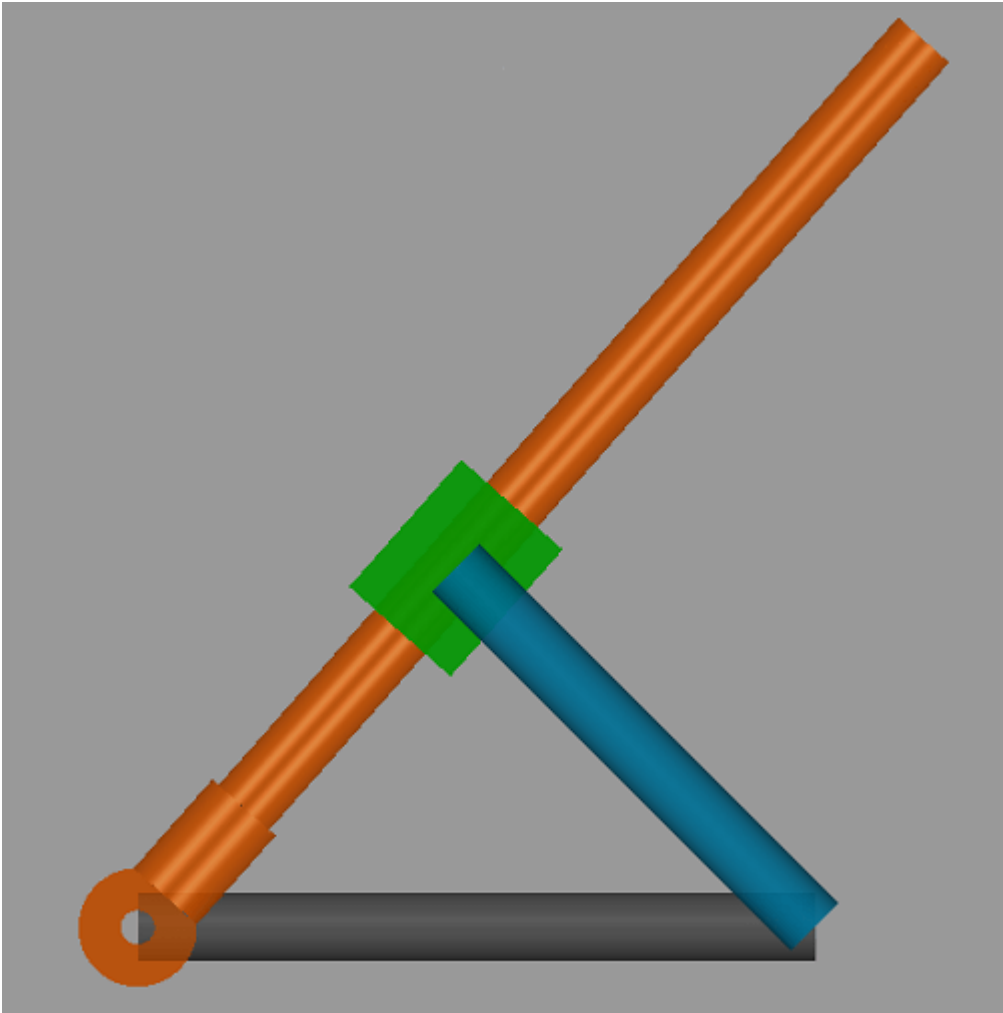
As an example, consider adding detail to the rigid body **A** while keeping its interface unchanged. The figure below shows rigid body **A** as a composition of simpler bodies. The interface exposed by rigid body **A** is still the pair of frames F_{AD} and F_{AC} . Their positions and orientations remain unchanged. The frames F_{12} , F_{21} , F_{23} and F_{32} are internal to the rigid body and should be created to assemble the individual pieces of the rigid body into a whole. The model **sm_dcrankaim_cplx_body_A** shows the construction of the complex version of the rigid body **A**.



The second version model **sm_dcrank_aiming_mechanism_v2** was obtained from **sm_dcrank_aiming_mechanism_v1** by just replacing the subsystem corresponding to rigid body A with the complex version from **sm_dcrankaim_cplx_body_A**. Because the interface remained constant, it was a simple operation of replacing the blocks. Simulate the model **sm_dcrank_aiming_mechanism_v2** to visualize the modified mechanism.



Components that make up detailed rigid body A



The tracking performance is similar because the controller is sufficiently robust to handle the slight differences in inertia between the simple and detailed version of rigid body **A**. Following a similar process, we can also add detail to the other parts. Different versions of each rigid body with varying levels of detail can be maintained in a library, and the model can be tested with these various alternatives. Configurable Subsystems would be useful here.

Summary

In summary, we took the following steps:

- Started with a schematic diagram of the mechanism and identified the rigid bodies and joints in the mechanism.
- Built a first approximation of each rigid body in isolation
- Assembled the rigid bodies together using joints to achieve the first version of the assembled mechanism.
- Used the **Model Report** tool to identify problems with the assembly
- Used **Joint Position Targets** to guide the assembly into a desirable configuration.
- Added a simple controller to the model to achieve target angle tracking.

- Once a full first version of the model was complete, added details to one of the rigid bodies without changing the interface of the rigid body. Details could be added to other rigid bodies as well.

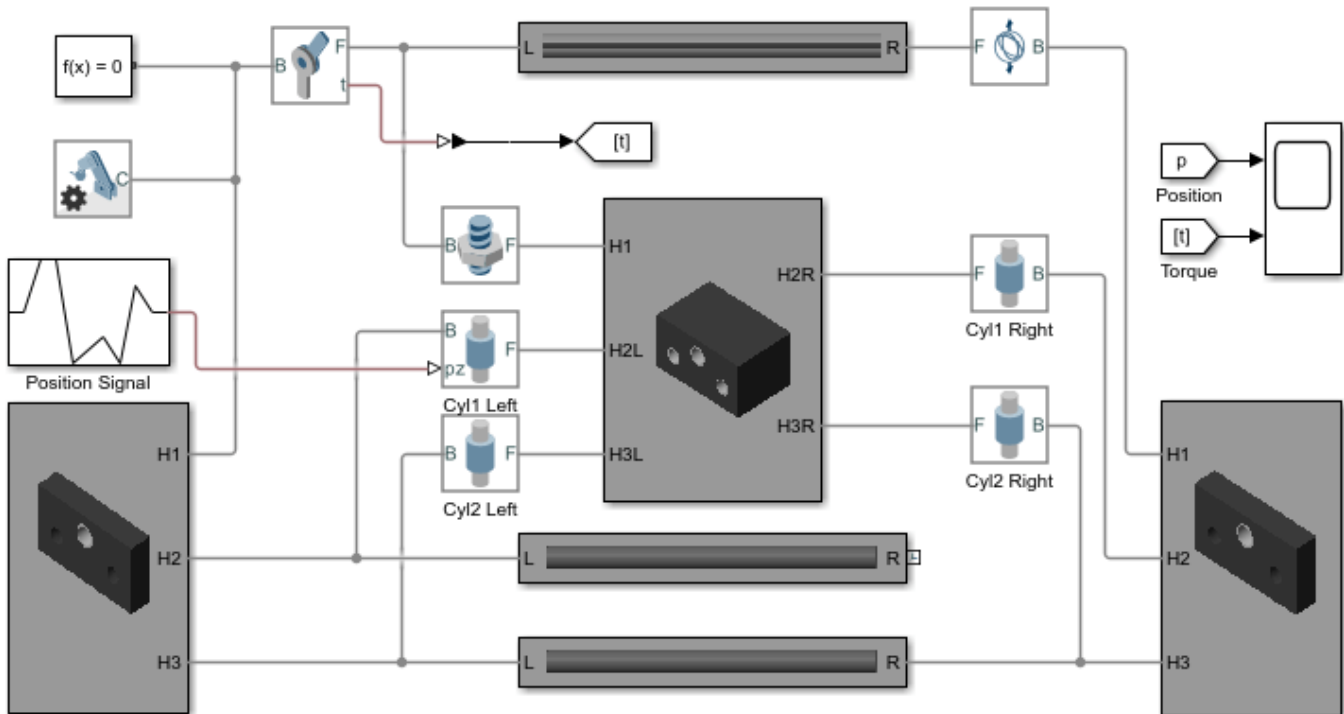
This method of starting simply and adding complexity in subsequent iterations is recommended when building models in Simscape Multibody.

See Also

Revolute Joint | Prismatic Joint

Using the Lead Screw Joint Block - Linear Actuator

This example illustrates the use of the Lead Screw Joint block to model a linear actuator. The Lead Screw Joint block converts rotational motion at the Revolute Joint block to translational motion at the four Cylindrical Joint blocks. The translational motion is specified as a motion input to a cylindrical joint and the necessary actuator torque is automatically computed at the revolute joint.



Using the Lead Screw Joint Block - Linear Actuator

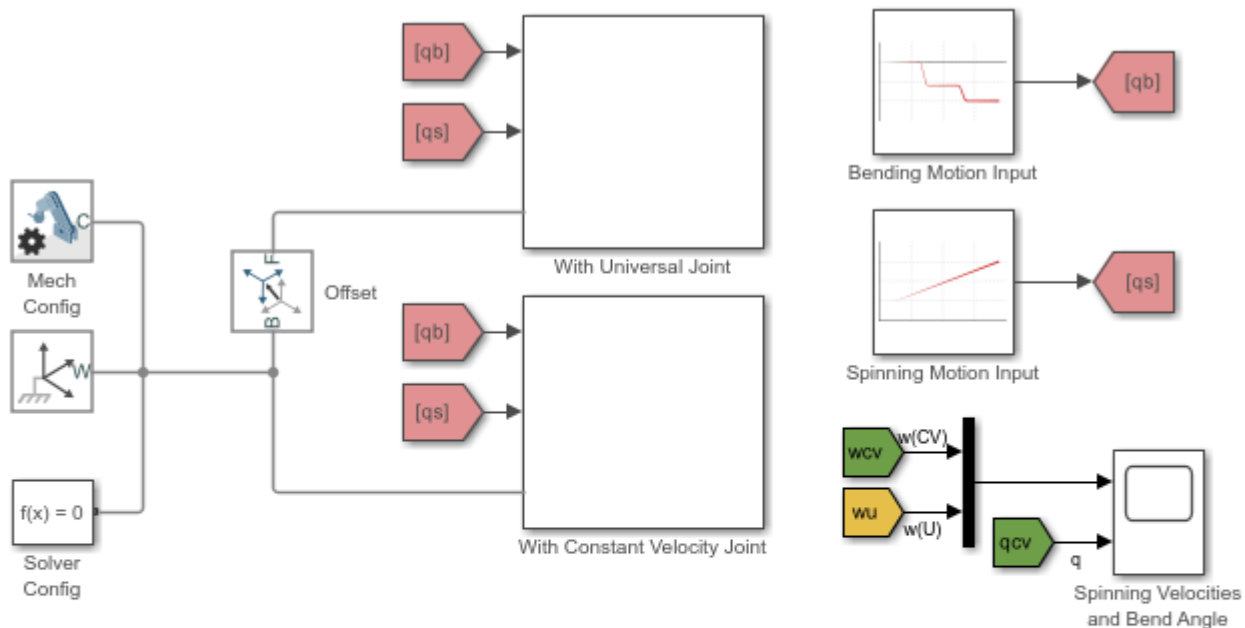
This example illustrates the use of the Lead Screw Joint block to model a linear actuator. The Lead Screw Joint block converts rotational motion at the Revolute Joint block to translational motion at the four Cylindrical Joint blocks. The translational motion is specified as a motion input to a cylindrical joint and the necessary actuator torque is automatically computed at the revolute joint.

Modeling Constant Velocity Joints - Power Take-Off Shaft

This example shows a Power Take-Off (PTO) shaft, a device for transferring power from tractor engines to auxiliary equipment such as soil tillers and wood chippers. The model includes two PTO subsystems identical in every sense but their joints. One contains universal (U) joints, the other constant-velocity (CV) joints.

At large bend angles, U joints lead to uneven rotations, subjecting the adjoining shafts to relatively large vibrations and elevated internal stresses. CV joints eliminate these by allowing the shafts to spin at constant velocity, resulting in a smooth motion profile no matter the bend angle.

A single set of motion inputs governs the behavior of the PTO subsystems. The Scope block plots the resulting angular velocities and bend angles of the shafts, enabling you to compare the kinematics of the two joint types.



Modeling Constant Velocity Joints - Power Take-Off Shaft

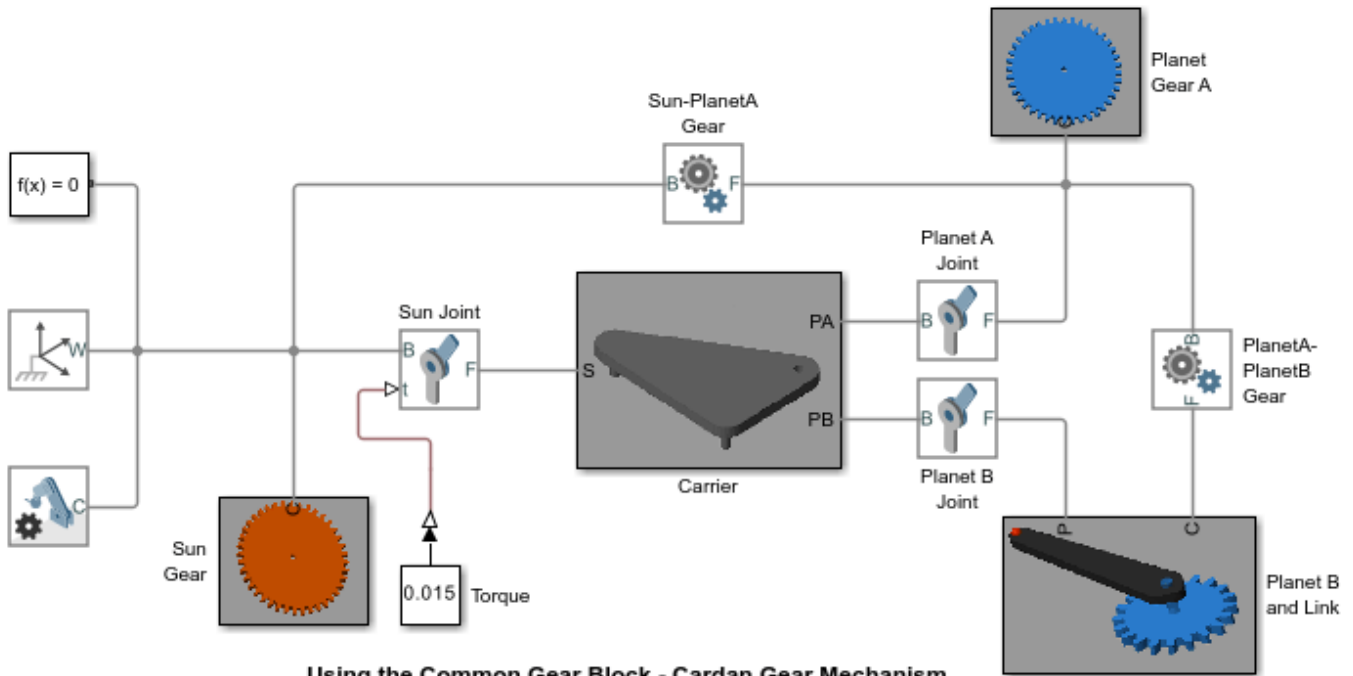
This example shows a Power Take-Off (PTO) shaft, a device for transferring power from tractor engines to auxiliary equipment such as soil tillers and wood chippers. The model includes two PTO subsystems identical in every sense but their joints. One contains universal (U) joints, the other constant-velocity (CV) joints.

At large bend angles, U joints lead to uneven rotations, subjecting the adjoining shafts to relatively large vibrations and elevated internal stresses. CV joints eliminate these by allowing the shafts to spin at constant velocity, resulting in a smooth motion profile no matter the bend angle.

A single set of motion inputs governs the behavior of the PTO subsystems. The Scope block plots the resulting angular velocities and bend angles of the shafts, enabling you to compare the kinematics of the two joint types.

Using the Common Gear Block - Cardan Gear Mechanism

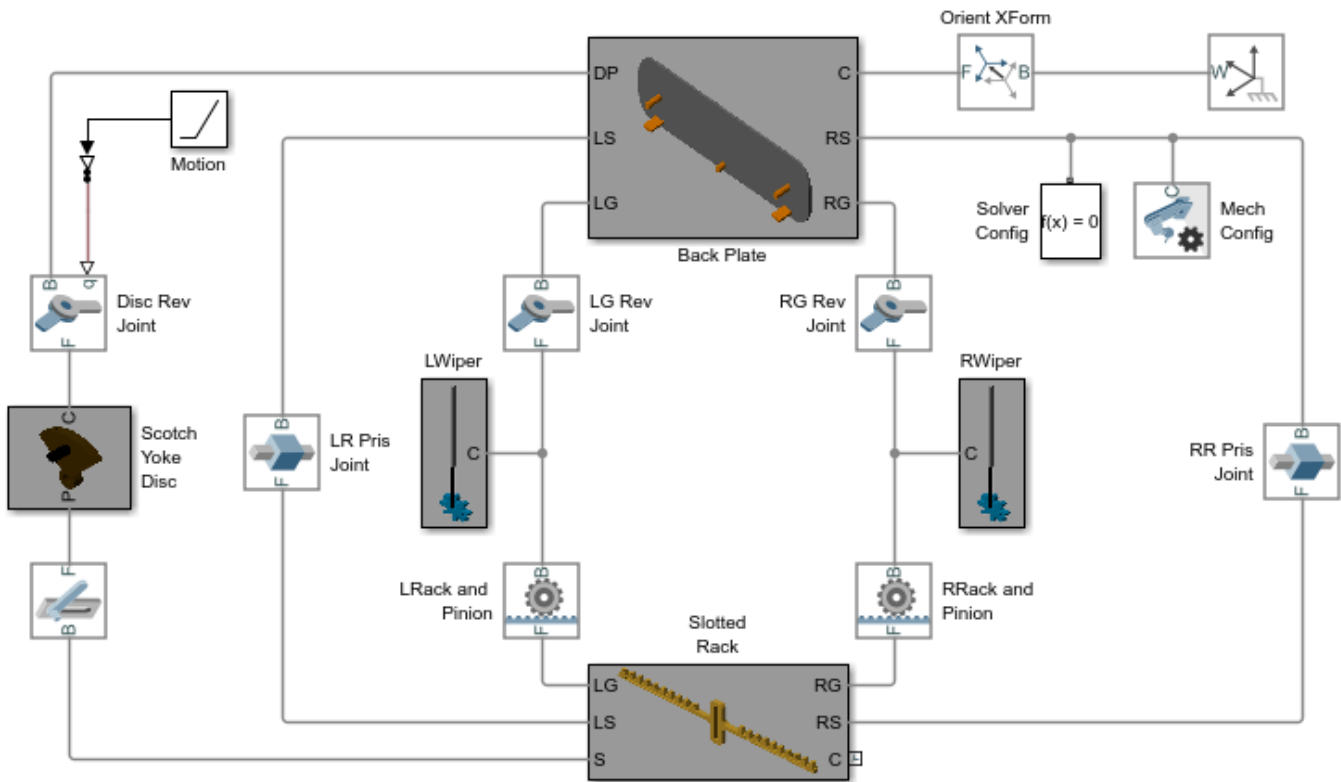
This example shows the Cardan Gear mechanism that converts rotational motion into reciprocating linear motion without using linkages or slideways. The mechanism uses three gears - one sun and two planet gears. The sun gear is twice as large as the planet gears (which are of the same size). The red pointer on the link traces a straight line as the gears rotate.



This example shows the Cardan Gear mechanism that converts rotational motion into reciprocating linear motion without using linkages or slideways. The mechanism uses three gears - one sun and two planet gears. The sun gear is twice as large as the planet gears (which are of the same size). The red pointer on the link traces a straight line as the gears rotate.

Using the Rack-Pinion Block - Windshield Wiper Mechanism

This example shows a windshield wiper mechanism. The mechanism utilizes a rack and pinion to drive the wiper blades in a synchronized fashion. The rack is actuated using a scotch yoke coupling (modeled using a pin-slot joint) that converts the rotary motion of the motor to reciprocating motion of the rack. The rack and pinion arrangement converts the reciprocating linear motion of the rack into reciprocating angular motion of the wiper blades (which are rigidly attached to the pinion).

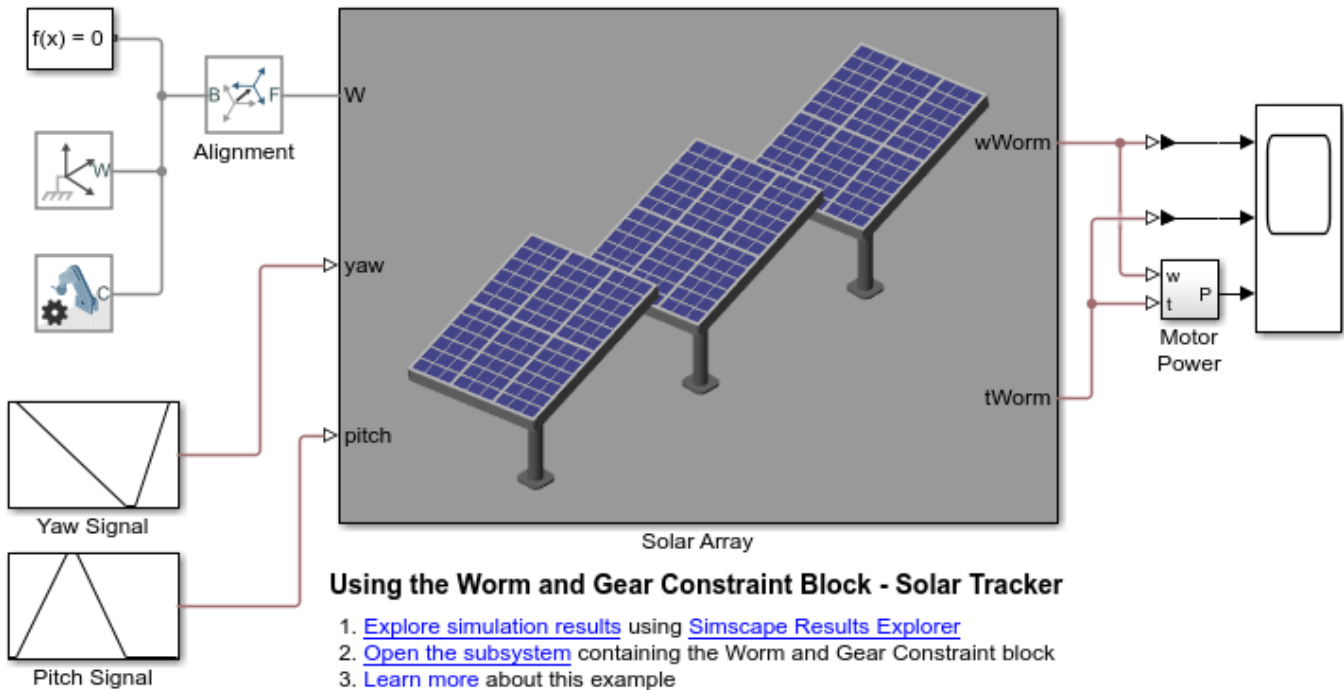


Using the Rack-Pinion Block - Windshield Wiper Mechanism

This example shows a windshield wiper mechanism. The mechanism utilizes a rack and pinion to drive the wiper blades in a synchronized fashion. The rack is actuated using a scotch yoke coupling (modeled using a pin-slot joint) that converts the rotary motion of the motor to reciprocating motion of the rack. The rack and pinion arrangement converts the reciprocating linear motion of the rack into reciprocating angular motion of the wiper blades (which are rigidly attached to the pinion).

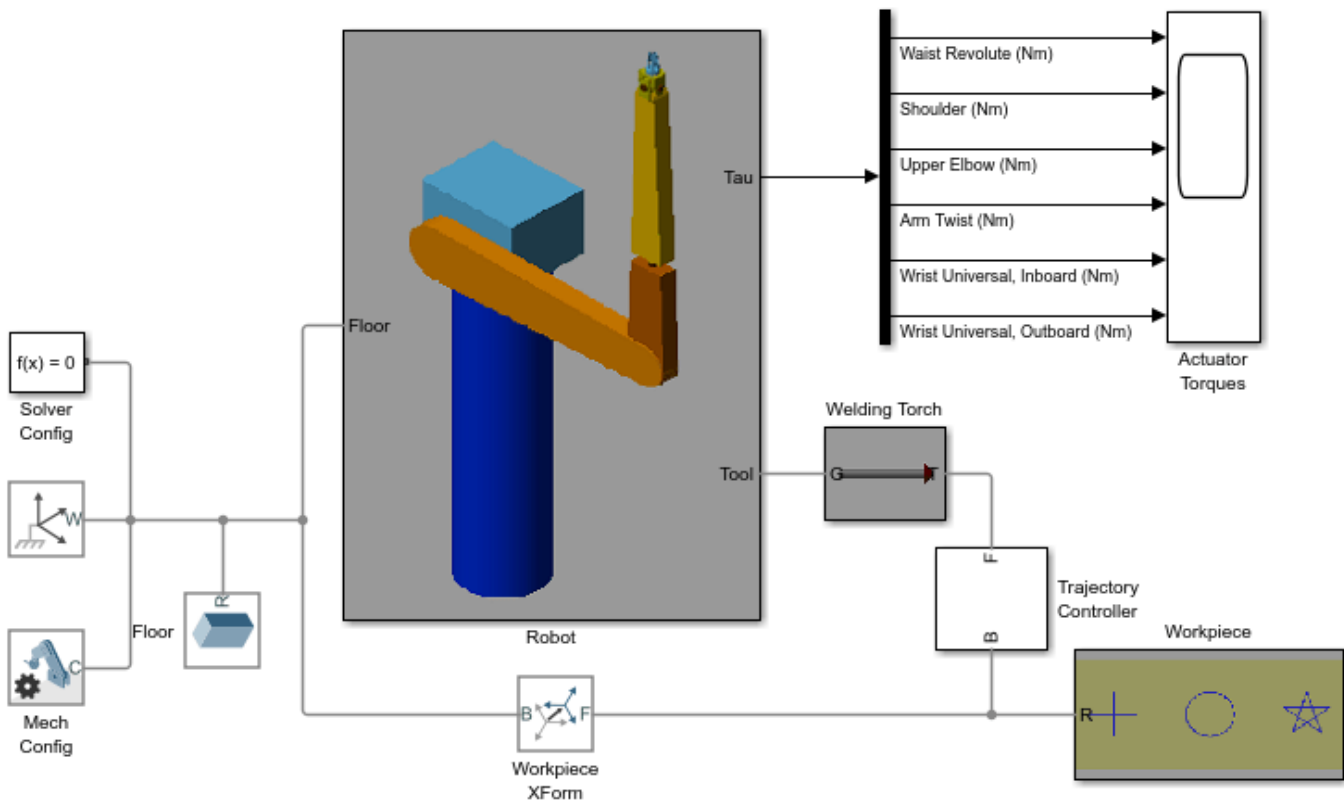
Using the Worm and Gear Constraint Block - Solar Tracker

This example illustrates the use of the Worm and Gear Constraint block to model a solar tracker. A slew drive containing a worm and gear constraint powers the yaw rotation of the solar trackers. The worm and gear geometry gives a large reduction in a single stage of gearing which provides precision tracking and high torque output. The yaw rotation is specified as a motion input to the gear revolute joint and the necessary actuator torque is automatically computed at the worm revolute joint.



Computing Actuator Torques Using Inverse Dynamics

This example illustrates the use of motion actuation to determine the actuator torques needed for the robot to achieve a given welding task. The system consists of a seven degree of freedom robot carrying a welding torch. The tip of the torch needs to trace the joints being welded. In this example the tip of the torch is made to trace (using motion actuation) a plus sign, a circle and a star sign on the workpiece. The torch is lifted off the workpiece when transitioning between the different shapes. The motion of the welding torch is specified and the actuator torques required at the various joints of the robot to achieve this motion is computed.



Computing Actuator Torques Using Inverse Dynamics

This example illustrates the use of motion actuation to determine the actuator torques needed for the robot to achieve a given welding task. The system consists of a seven degree of freedom robot carrying a welding torch. The tip of the torch needs to trace the joints being welded. In this example the tip of the torch is made to trace (using motion actuation) a plus sign, a circle and a star sign on the workpiece. The torch is lifted off the workpiece when transitioning between the different shapes. The motion of the welding torch is specified and the actuator torques required at the various joints of the robot to achieve this motion are computed.

See Also

Bushing Joint | Revolute Joint

More About

- “Cartesian 3D Printer” on page 8-121
- “Joint Actuation Limitations” on page 3-26

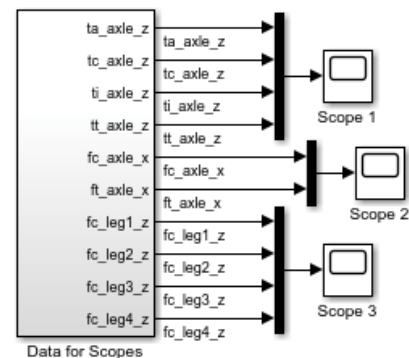
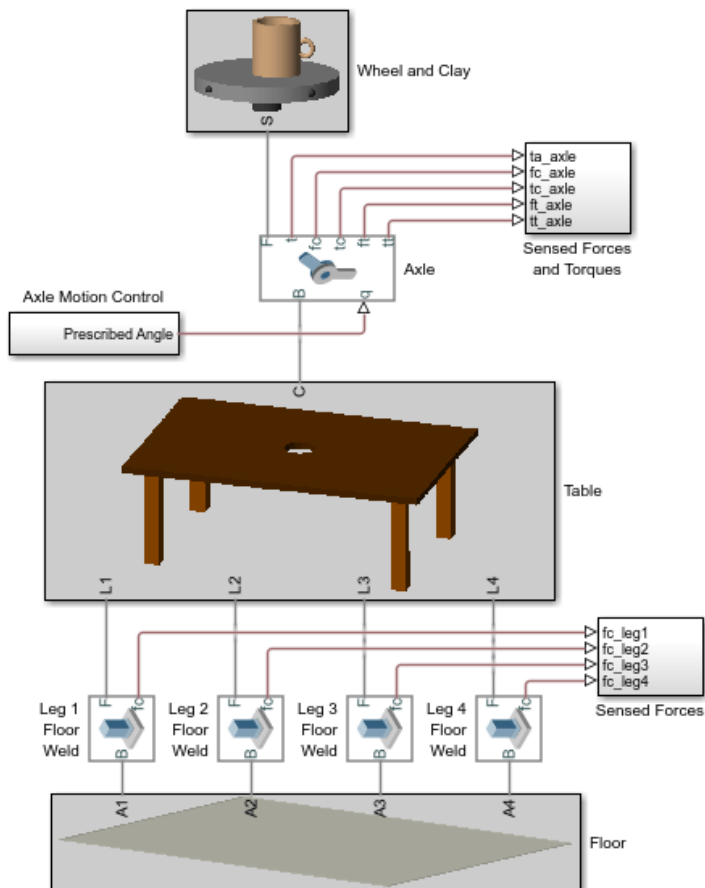
- “Motion Sensing” on page 3-56
- “Specifying Joint Actuation Inputs” on page 3-19

Sensing Composite Forces and Torques in Joints - Potter's Wheel

The example shows how you can sense forces and torques acting at joints. A potter's wheel spins with a piecewise linear velocity profile while holding a piece of clay off center. The clay creates a dynamic imbalance, generating periodic constraint forces at the joints. Viscous damping accounts for energy dissipation in the axle.

Scopes 1 and 2 show the composite forces and torques that you can sense at the joints. These include the constraint forces and torques and also the total forces and torques, which are the net sum of constraint, actuation and internal components. The constraint components are zero in the direction of motion, while the actuation and internal components are zero in the constrained directions.

If a multibody system is overconstrained, Simscape™ Multibody™ distributes the constraint forces and torques over the joints enforcing the redundant constraints. The vertical constraint forces acting at the leg weld joints are one example. These forces, shown in Scope 3, add up to the total weight of the table. They are nearly equal, with only slight discrepancies among them due to asymmetries in the shape and position of the clay.



Sensing Composite Forces and Torques in Joints - Potter's Wheel

The example shows how you can sense forces and torques acting at joints. A potter's wheel spins with a piecewise linear velocity profile while holding a piece of clay off center. The clay creates a dynamic imbalance, generating periodic constraint forces at the joints. Viscous damping accounts for energy dissipation in the axle.

Scopes 1 and 2 show the composite forces and torques that you can sense at the joints. These include the constraint forces and torques and also the total forces and torques, which are the net sum of constraint, actuation and internal components. The constraint components are zero in the direction of motion, while the actuation and internal components are zero in the constrained directions.

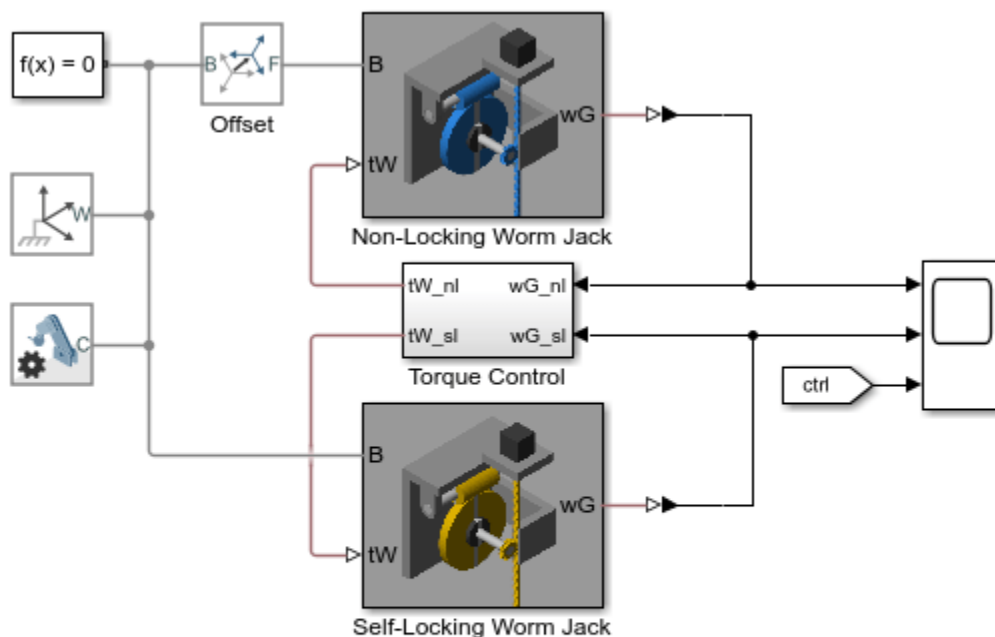
If a multibody system is overconstrained, Simscape Multibody(TM) distributes the constraint forces and torques over the joints enforcing the redundant constraints. The vertical constraint forces acting at the leg weld joints are one example. These forces, shown in Scope 3, add up to the total weight of the table. They are nearly equal, with only slight discrepancies among them due to asymmetries in the shape and position of the clay.

Modeling Self-Locking Worm and Gear Constraints - Worm Jack

This example models a self-locking worm and gear constraint. The model shows a mechanical jack driven by a torque applied to a worm. The model includes two worm jack subsystems identical in every sense except for the value of the worm lead angle. Both subsystems have friction models applied to their Worm and Gear Constraint blocks.

A worm and gear constraint is self-locking when the static friction coefficient is greater than the tangent of the worm lead angle. One subsystem has a worm lead angle slightly less than this threshold (self-locking) and the other subsystem has a worm lead angle slightly greater than this threshold (non-locking).

The torque applied to the worm lifts a load via a rack and pinion. When the load reaches its highest point the torque signal is disabled. This effectively models a motor malfunction. The non-locking subsystem back-drives and the load falls under gravity. The self-locking subsystem remains at the highest point.

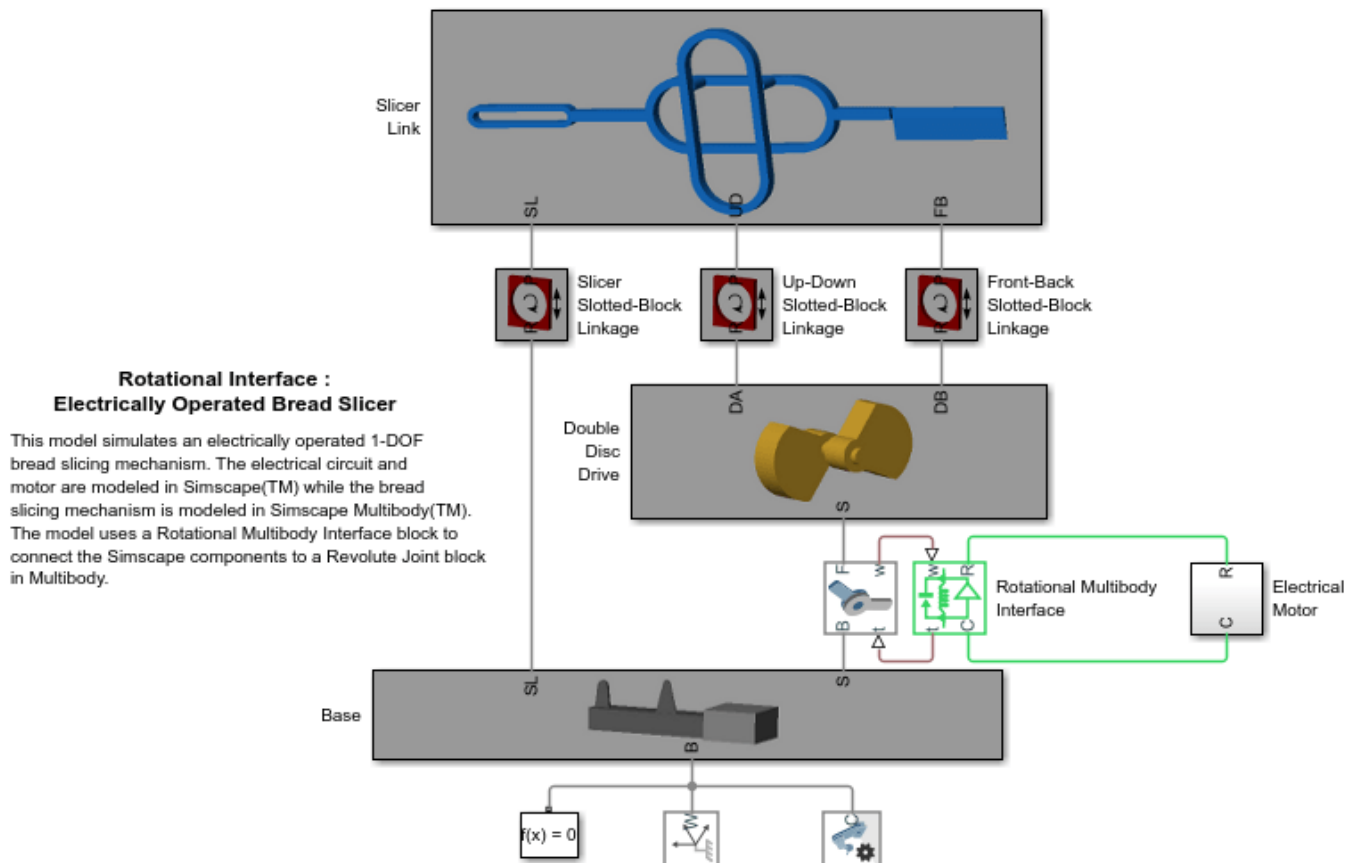


Modeling Self-Locking Worm and Gear Constraints - Worm Jack

1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Open the subsystem](#) containing the friction model
3. [Learn more](#) about this example and the [friction model](#)
4. Learn more about the [Worm and Gear Constraint block](#)
5. Learn more about [multibody modeling](#)

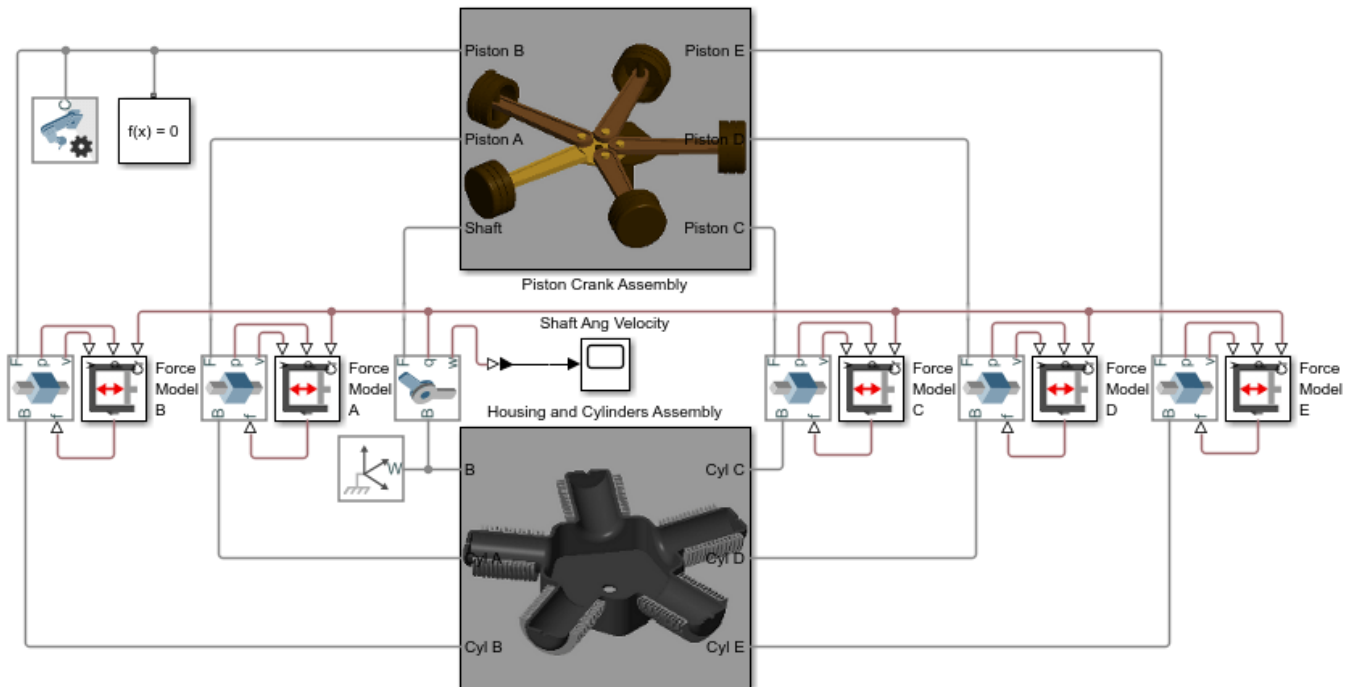
Rotational Interface : Electrically Operated Bread Slicer

This model simulates an electrically operated 1-DOF bread slicing mechanism. The electrical circuit and motor are modeled in Simscape™ while the bread slicing mechanism is modeled in Simscape Multibody™. The model uses a Rotational Multibody Interface block to connect the Simscape components to a Revolute Joint block in Multibody.



Translational Interface : Radial Engine with Gas Force Model

This model simulates a five cylinder radial engine. The pressure dynamics inside the cylinders are modelled using the Simscape™ Foundation Library gas and mechanical translational domains. The 3D mechanical components are modelled using Simscape Multibody™. See inside any of the blocks called "Force Model" to see how the 1D Simscape and 3D Multibody parts of the model are interfaced. The pressure model is an ideal pressure source which applies pressure based on the crank angle. This model can be replaced with a more realistic pressure model of the cylinder chamber. The cylinders fire in the sequence - A C E B D providing a power stroke every 144 deg of crank rotation.



Translational Interface : Radial Engine with Gas Force Model

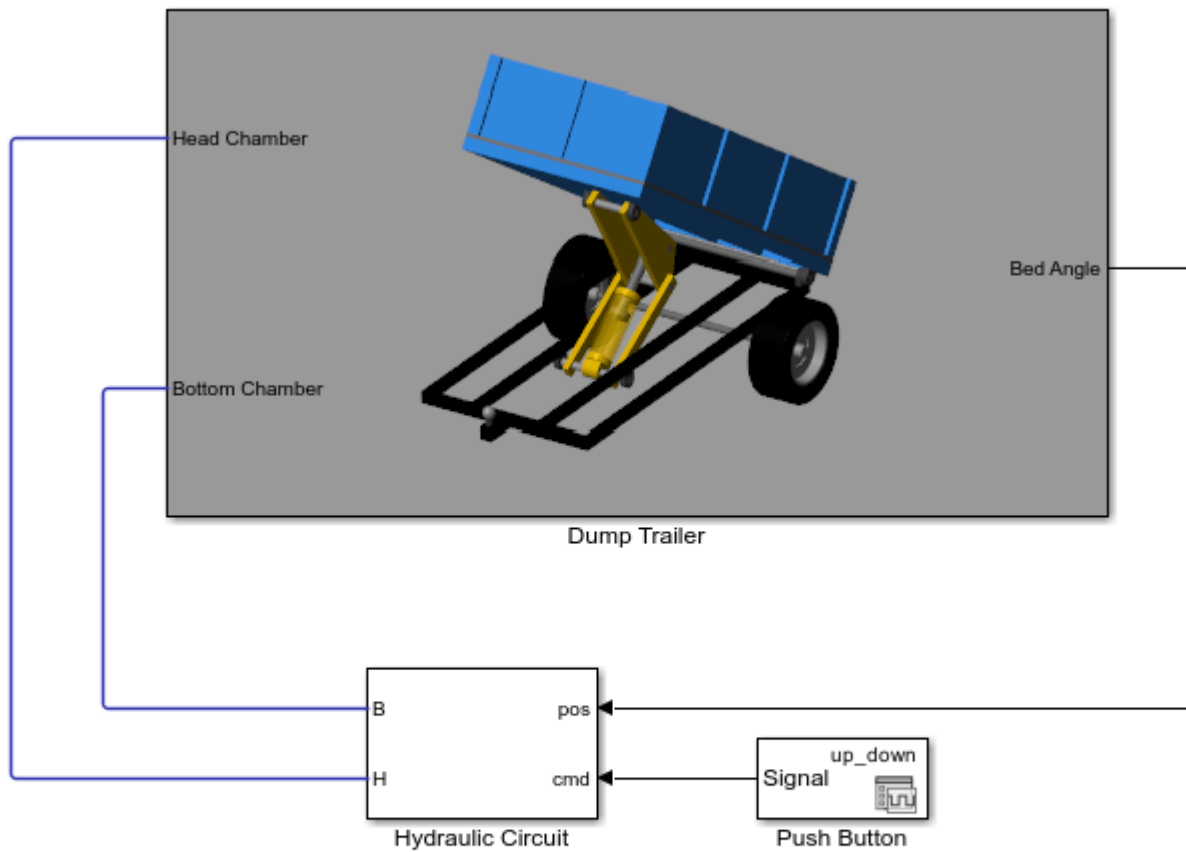
This model simulates a five cylinder radial engine. The pressure dynamics inside the cylinders are modelled using the Simscape(TM) Foundation Library gas and mechanical translational domains. The 3D mechanical components are modelled using Simscape Multibody(TM). See inside any of the blocks called "Force Model" to see how the 1D Simscape and 3D Multibody parts of the model are interfaced. The pressure model is an ideal pressure source which applies pressure based on the crank angle. This model can be replaced with a more realistic pressure model of the cylinder chamber. The cylinders fire in the sequence - A C E B D providing a power stroke every 144 deg of crank rotation.

Hydraulic Interface - Dump Trailer with Hydraulic Cylinder

This example shows a dump trailer powered by a double-acting hydraulic cylinder. The cylinder actuates a scissor hoist mechanism that raises and lowers the dump bed. The model provides an example of how to interface Simscape™ Multibody™ joints with Simscape components that have mechanical domain ports.

The Double-Acting Hydraulic Cylinder block in the scissor hoist contains mechanical and hydraulic subsystems. The mechanical subsystem models the piston and barrel. A Prismatic Joint block provides the translational degree of freedom needed for piston extension and retraction. The piston's range of motion is constrained using joint limits. The hydraulic subsystem contains two Translational Mechanical Converter blocks from the Isothermal Liquid (IL) foundation library in Simscape. These blocks exchange velocity and force information with the prismatic joint via a Translational Multibody Interface block. In addition, both of the converter blocks are configured to accept a physical position signal from the prismatic joint. These connections ensure that the state of the hydraulic chambers in the converter blocks are always consistent with the state of the prismatic joint.

The Double-Acting Hydraulic Cylinder block is located in the `sm_interface_elements_lib` supporting library.

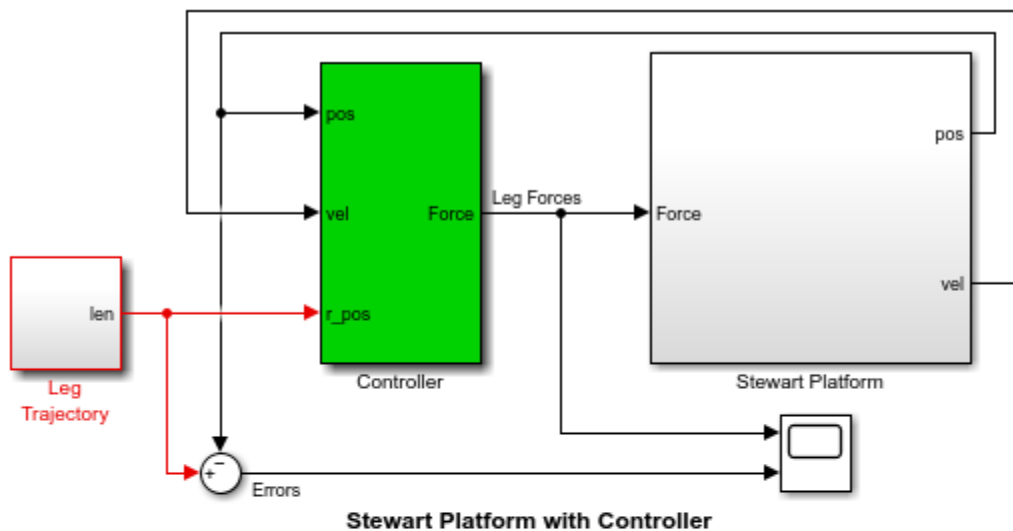


Hydraulic Interface - Dump Trailer with Hydraulic Cylinder

1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Learn more](#) about this example
3. Learn more about [multibody modeling](#)
4. Open the [Hydraulic Interface library](#)

Stewart Platform with Controller

This model illustrates the CAD import workflow in Simscape™ Multibody™. 1. Export CAD model into a Simscape Multibody Import XML (stewart_platform.xml). 2. Import the Simscape Multibody Import XML into a Simscape Multibody model using the smimport command. 3. Enclose the imported model into a subsystem (Stewart Platform). Interface layer is created to isolate the imported model from the blocks used to interface it to the rest of the model. The subsystem "Stewart Platform/Imported Stewart Platform" contains only the imported model. 4. Export input (Force) and output (pos and vel) ports that correspond to the control inputs and sensor outputs. These form the interface of the Stewart Platform model to its controller. 5. Within the subsystem, connect the force input and sensor output ports to appropriate blocks of the Stewart Platform model. Example: see block "Stewart Platform/Imported Stewart Platform/ActuatorAssm1_2/Cylindrical". 6. Connect the controller to the Stewart Platform model inputs and outputs. 7. Design the controller gains using control design tools. 8. If there are any changes to the CAD model, then it can be re-exported and the contents of the "Stewart Platform/Imported Stewart Platform" subsystem can be replaced with the newly imported model without changing its interface (ports) to the controller. The controller gains can be re-tuned.

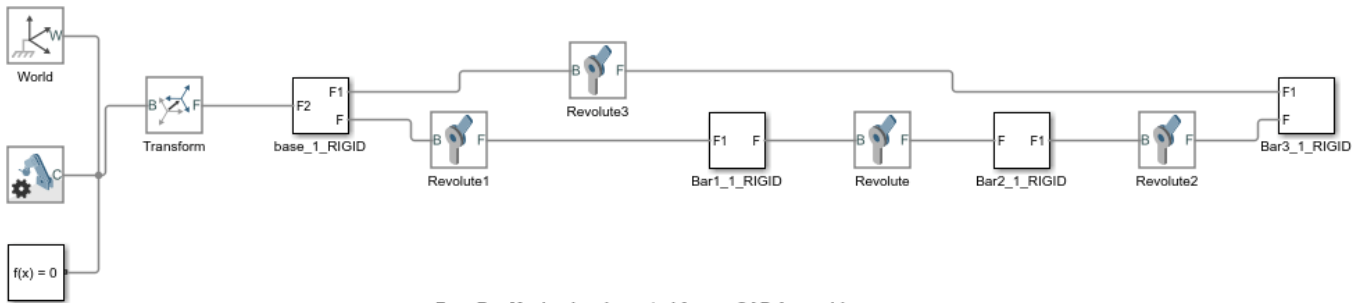


This model illustrates the CAD import workflow in Simscape Multibody(TM).

1. Export CAD model into a SimMechanics Import XML (stewart_platform.xml)
2. Import the Simscape Multibody Import XML into a Simscape Multibody model using the smimport command
3. Enclose the imported model into a subsystem (Stewart Platform). Interface layer is created to isolate the imported model from the blocks used to interface it to the rest of the model. The subsystem "Stewart Platform/Imported Stewart Platform" contains only the imported model.
4. Export input (Force) and output (pos and vel) ports that correspond to the control inputs and sensor outputs. These form the interface of the Stewart Platform model to its controller.
5. Within the subsystem, connect the force input and sensor output ports to appropriate blocks of the Stewart Platform model.
Example: see block "Stewart Platform/Imported Stewart Platform/ActuatorAssm1_2/Cylindrical"
6. Connect the controller to the Stewart Platform model inputs and outputs.
7. Design the controller gains using control design tools.
8. If there are any changes to the CAD model, then it can be re-exported and the contents of the "Stewart Platform/Imported Stewart Platform" subsystem can be replaced with the newly imported model without changing its interface (ports) to the controller. The controller gains can be re-tuned.

Four Bar Mechanism Imported from a CAD Assembly

This example has been imported from a CAD assembly designed in SolidWorks® using the smimport command. The XML file "fourbar.xml" and the STL files obtained during export of the CAD assembly have been used to create this example.

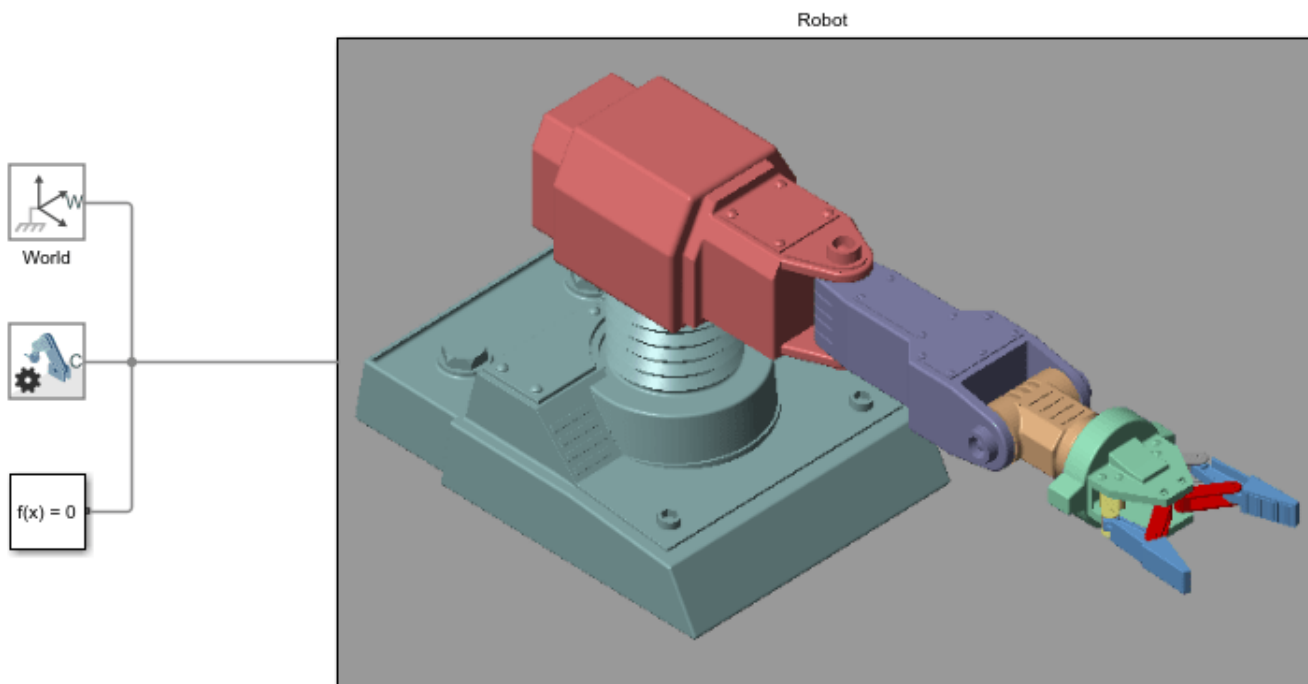


Four Bar Mechanism Imported from a CAD Assembly

This example has been imported from a CAD assembly designed in SolidWorks(R) using the smimport command.

Modeling A Robot Using STEP Files

This example shows how to import geometry and inertia data using STEP files. The STEP file is a standard format used commonly for data exchange between CAD applications. The format can capture a parts complete geometry information. Given the mass of the part and the volume distribution, Simscape™ Multibody™ can automatically compute the inertia properties of the part. Open the Solid block dialogs in the above model to see how the geometry and inertia parameters are configured to utilize STEP files exported from another application (CAD for example). This method of importing geometry and inertia information can be used for CAD systems that are not supported by Simscape Multibody Link. A valid STEP file from any source application can be imported by this method.



Modeling A Robot Using STEP Files

This example shows how to import geometry and inertia data using STEP files. The STEP file is a standard format used commonly for data exchange between CAD applications. The format can capture a parts complete geometry information. Given the mass of the part and the volume distribution Simscape Multibody can automatically compute the inertia properties of the part. Open the Solid block dialogs in the above model to see how the geometry and inertia parameters are configured to utilize STEP files exported from a CAD application. This method of importing geometry and inertia information can be used for CAD systems that are not supported by Simscape Multibody Link. A valid STEP file from any source application can be imported by this method.

See Also

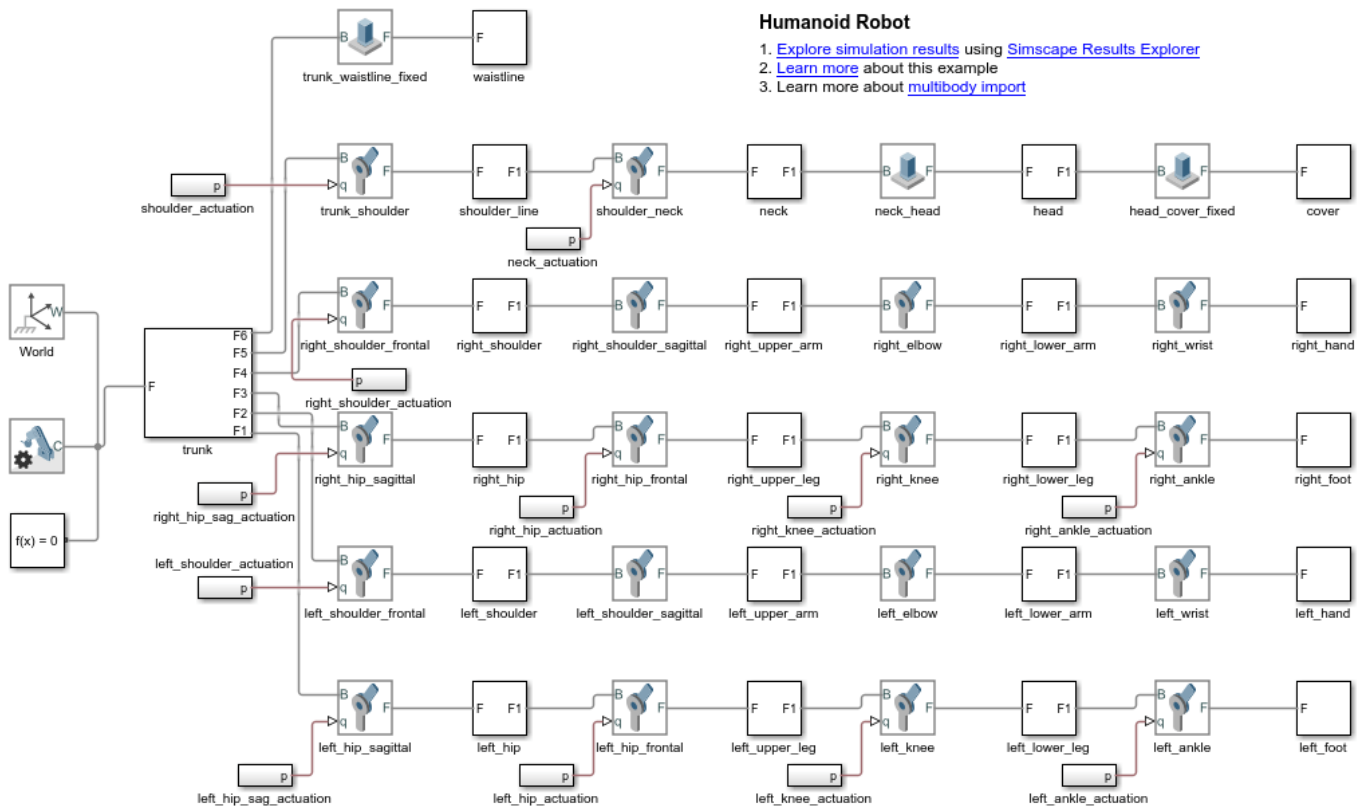
File Solid

More About

- “Specifying Custom Inertias” on page 1-68

Humanoid Robot

This example has been imported from a URDF file using the `smimport` command. The URDF file "sm_humanoid.urdf" and the STEP files that visualize the robot parts were used to create this example. Motion actuation of the joints was manually added to the imported model to make the robot perform interesting movements.



See Also

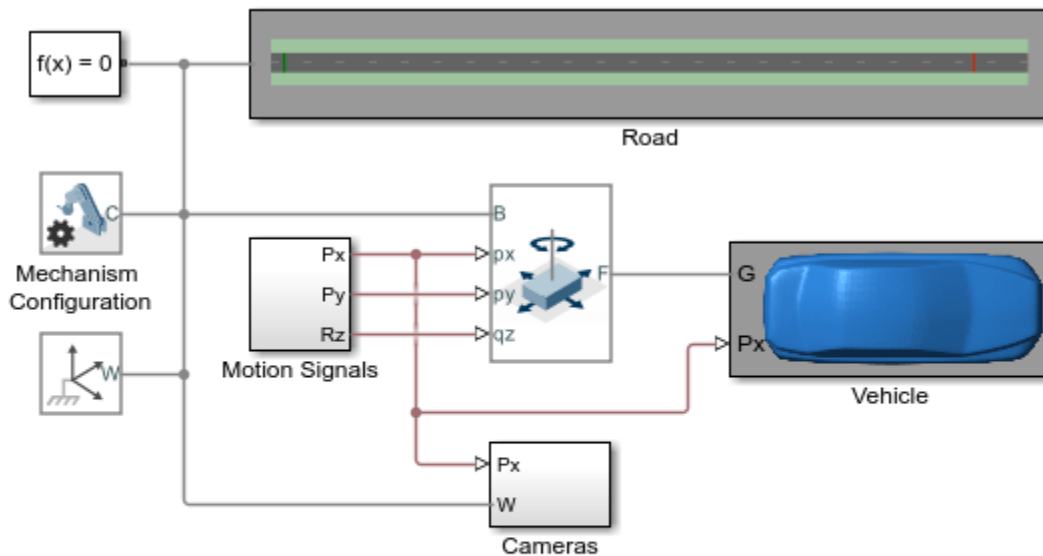
`smimport`

More About

- "Import a URDF Humanoid Model" on page 6-40
- "Import URDF Models" on page 6-33
- "Specifying Joint Actuation Inputs" on page 3-19

Configuring Dynamic Cameras - Vehicle Slalom

This example illustrates the use of dynamic cameras to view a vehicle traversing a slalom course. The vehicle accelerates from stationary to a fixed speed and then enters the slalom course. Upon exiting, the vehicle decelerates back to stationary before the end of the road. The model has four dynamic cameras: three with tracking parameterization and one with keyframes. The tracking cameras show the front and side views of the vehicle from a fixed offset relative to the vehicle, along with the view from the driver's perspective. The keyframed camera shows a sweeping view of the vehicle maneuver.



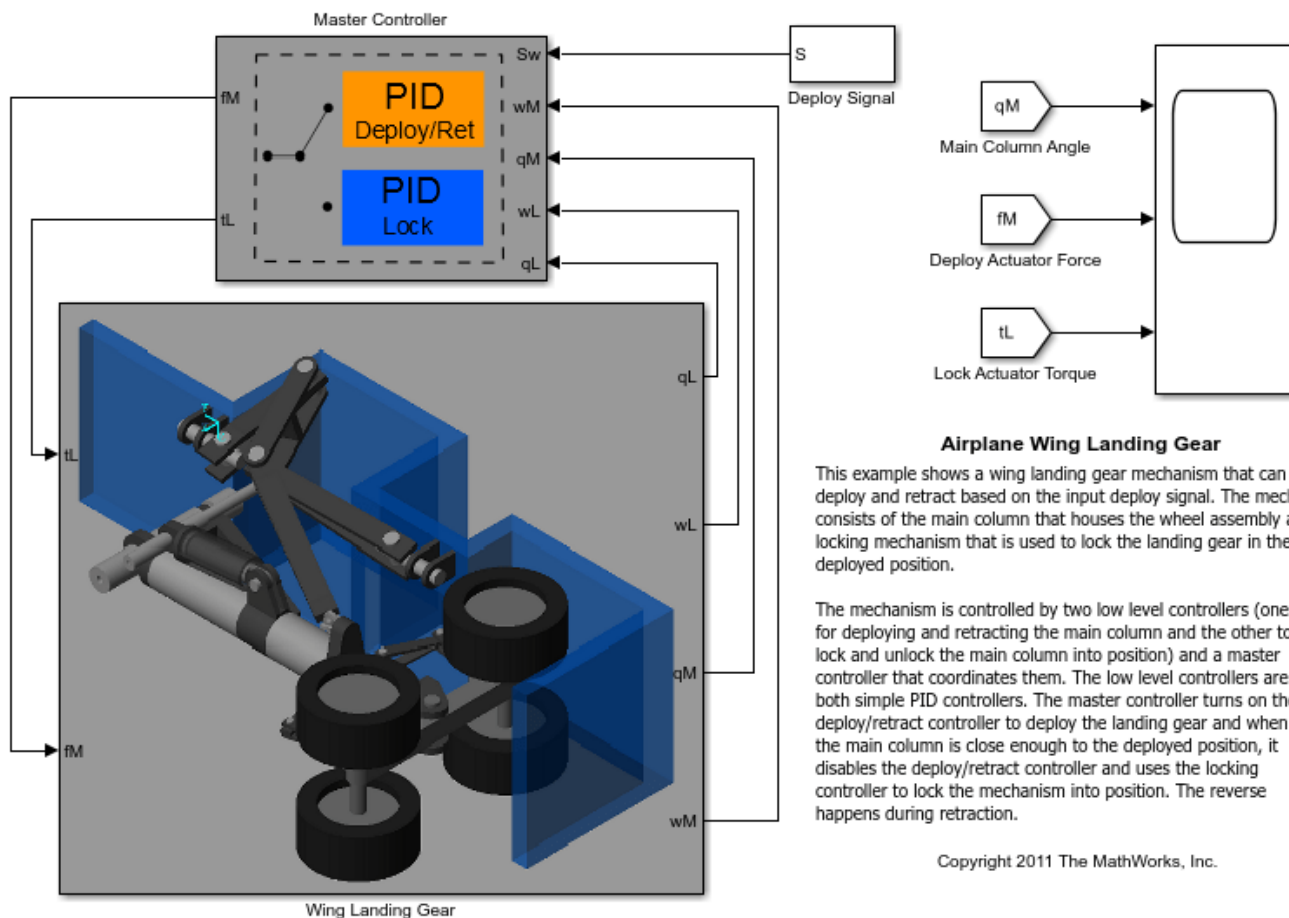
Configuring Dynamic Cameras - Vehicle Slalom

1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Learn more](#) about this example
3. Learn more about [dynamic cameras](#)
4. Learn more about [multibody modeling](#)

Airplane Wing Landing Gear

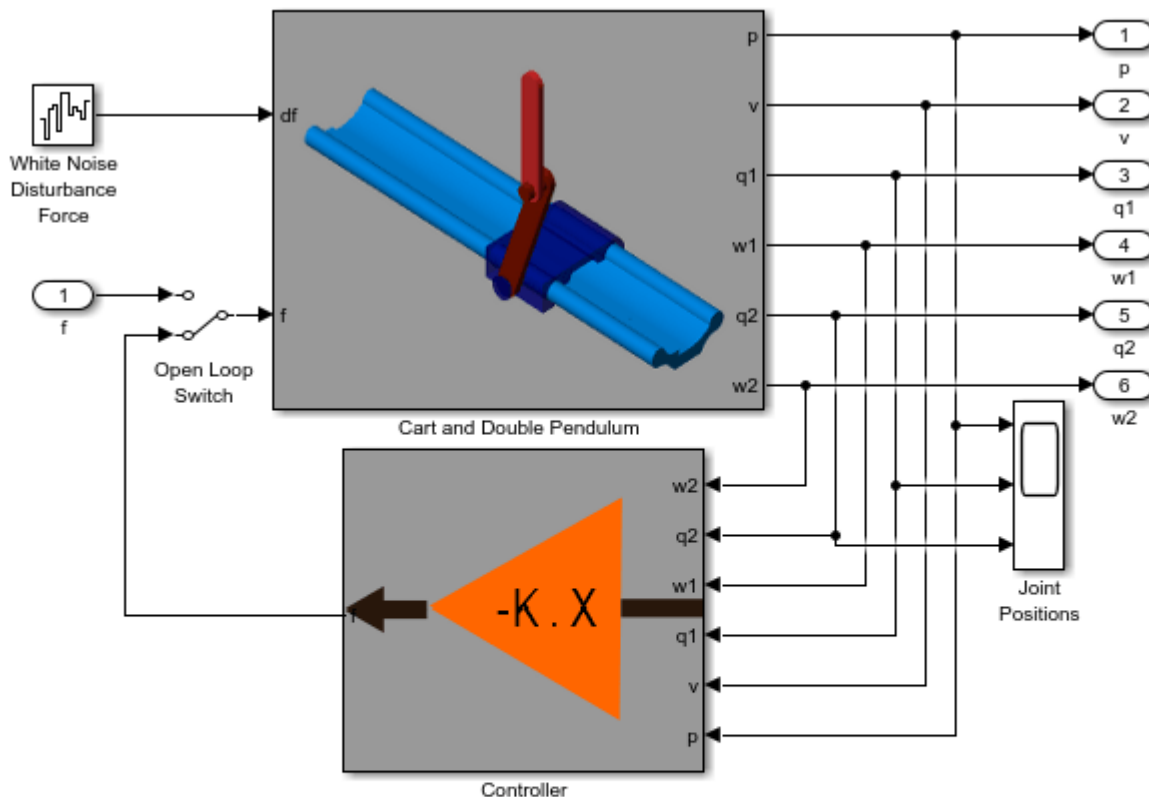
This model shows a wing landing gear mechanism that can deploy and retract based on the input deploy signal. The mechanism consists of the main column that houses the wheel assembly and the locking mechanism that is used to lock the landing gear in the deployed position.

The mechanism is controlled by two low level controllers (one for deploying and retracting the main column and the other to lock and unlock the main column into position) and a master controller that coordinates them. The low level controllers are both simple PID controllers. The master controller turns on the deploy/retract controller to deploy the landing gear and when the main column is close enough to the deployed position, it disables the deploy/retract controller and uses the locking controller to lock the mechanism into position. The reverse happens during retraction.



Inverted Double Pendulum on a Sliding Cart

This example shows how to model an inverted double pendulum mounted on a sliding cart using Simscape™ Multibody™. It also illustrates the use of a controller to balance the pendulum in the upright position. Make any changes to the system and click on the blue box to generate linearized model for the system before running the simulation. The control gains are computed using the linearized model. The pole placement technique is used to compute the control gains from the linearized model. The controller keeps the double pendulum vertical in the presence of a random disturbance force. See the files `sm_cart_dpen_linearize.m` and `sm_cart_dpen_control_gains.m` for details.



Inverted Double Pendulum on a Sliding Cart

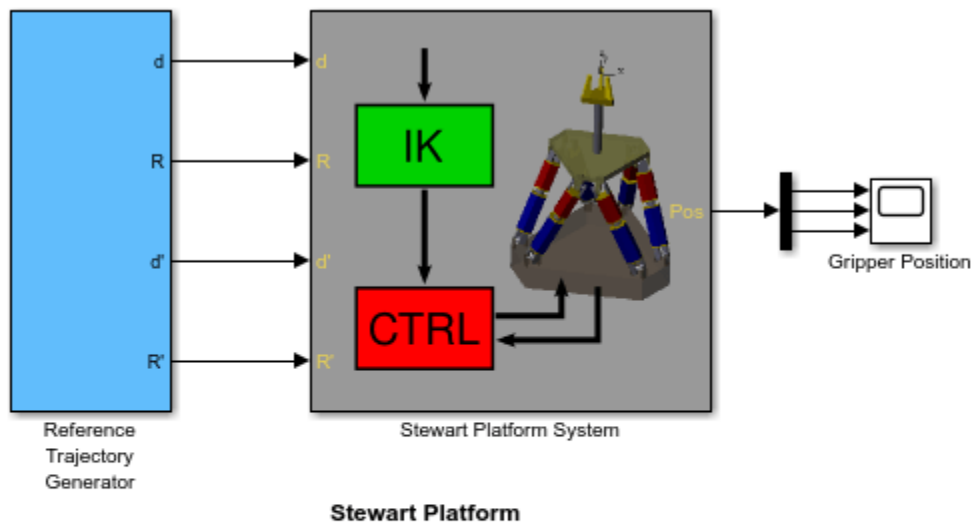
This example shows how to model an inverted double pendulum mounted on a sliding cart using Simscape Multibody(TM). It also illustrates the use of a controller to balance the pendulum in the upright position. Make any changes to the system and click on the blue box to generate linearized model for the system before running the simulation. The control gains are computed using the linearized model. The pole placement technique is used to compute the control gains from the linearized model. The controller keeps the double pendulum vertical in the presence of a random disturbance force. See the files `sm_cart_dpen_linearize.m` and `sm_cart_dpen_control_gains.m` for details.

Double click to generate linearized system after changes to plant

Stewart Platform

This model shows a Stewart platform manipulator that can track a parameterized reference trajectory. The shape, size, and kinematics of the manipulator are highly configurable.

The reference trajectory is specified in 6-D pose space, and an inverse kinematics module converts it into one through 6-D leg position space. A generic PID controller attempts to drive the manipulator along the desired trajectory.



This example shows a Stewart platform manipulator that can track a parameterized reference trajectory. The shape, size, and kinematics of the manipulator are highly configurable.

The reference trajectory is specified in 6-D pose space, and an inverse kinematics module converts it into one through 6-D leg position space. A generic PID controller attempts to drive the manipulator along the desired trajectory.

Copyright 2011 The MathWorks, Inc.

See Also

Cylindrical Joint | Transform Sensor

More About

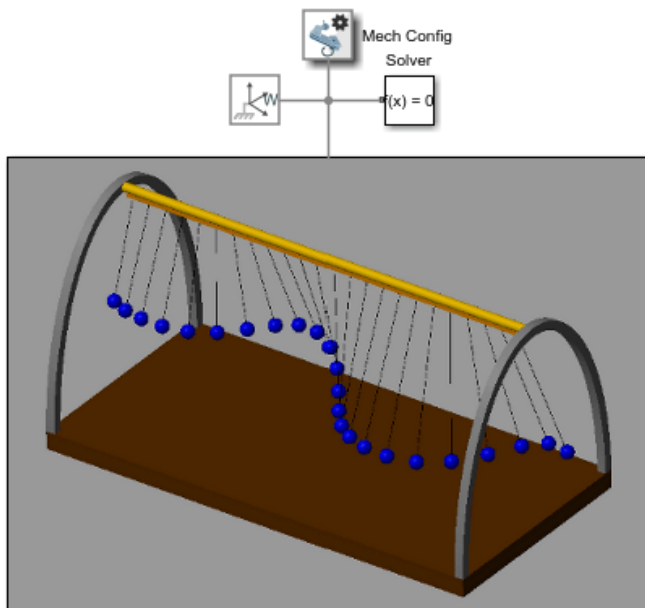
- “Stewart Platform with Controller” on page 8-60

Pendulum Waves

This example shows interesting wave patterns that emerge among an array of simple pendulums with carefully chosen lengths. It is based on the physical system that can be viewed at www.youtube.com/watch?v=yVkdFJ9PkRQ

Under a common approximation of pendulum motion, the period of a pendulum depends only upon the length of the pendulum and the acceleration of gravity. The pendulum system in this example comprises 24 pendulums with lengths chosen so that each pendulum completes a different integral number of oscillations during a 60 second "system period". The number of oscillations of the pendulums during the system period lie among the 24 integers in the range [68, 91]: the longest pendulum completes 68 oscillations while the shortest one completes 91. When the pendulums are released from rest at a common initial angle, wave patterns emerge in their motion. These patterns are most evident when the system is viewed from the side or top.

The entire simulation runs for two minutes, or two system periods; at the one minute mark, the pendulums return to their initial state, and the pattern repeats. The formula for computing pendulum periods assumes a pendulum executes simple harmonic motion. In fact, this is an approximation that becomes less accurate as the angular amplitude of the swing increases. As a result, slight discrepancies emerge in the motion patterns, and it is evident that after two system periods, the system has not quite returned to its initial state. (The true period of a simple pendulum is longer than the one computed under the harmonic motion assumption.) Reducing the initial angle improves the repeatability of the wave patterns.



Pendulum System

Pendulum Waves

This example shows interesting wave patterns that emerge among an array of simple pendulums with carefully chosen lengths. It is based on the physical system that can be viewed at www.youtube.com/watch?v=yVkdFJ9PkRQ

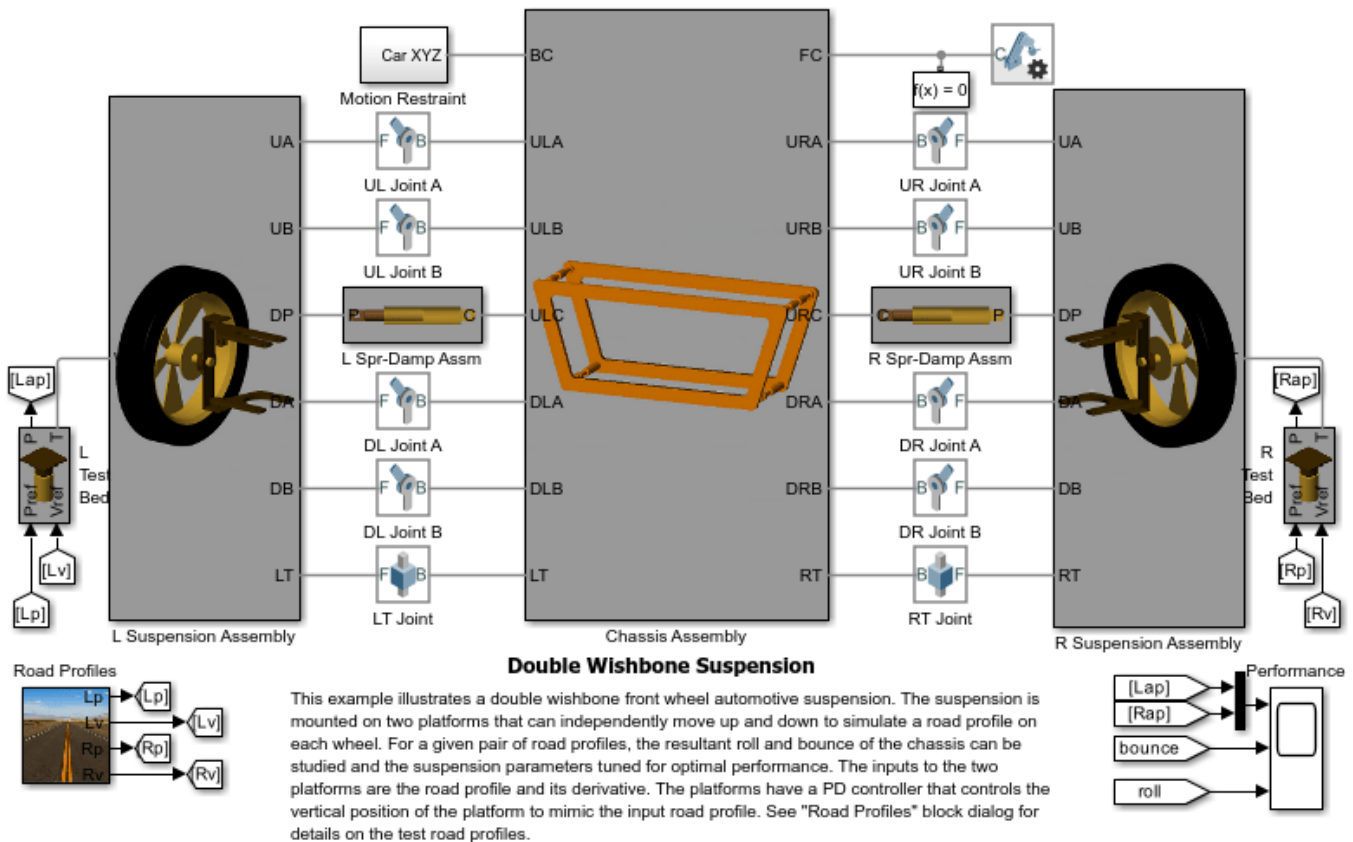
Under a common approximation of pendulum motion, the period of a pendulum depends only upon the length of the pendulum and the acceleration of gravity. The pendulum system in this example comprises 24 pendulums with lengths chosen so that each pendulum completes a different integral number of oscillations during a 60 second "system period". The number of oscillations of the pendulums during the system period lie among the 24 integers in the range [68, 91]: the longest pendulum completes 68 oscillations while the shortest one completes 91. When the pendulums are released from rest at a common initial angle, wave patterns emerge in their motion. These patterns are most evident when the system is viewed from the side or top.

The entire simulation runs for two minutes, or two system periods; at the one minute mark, the pendulums return to their initial state, and the pattern repeats. The formula for computing pendulum periods assumes a pendulum executes simple harmonic motion. In fact, this is an approximation that becomes less accurate as the angular amplitude of the swing increases. As a result, slight discrepancies emerge in the motion patterns, and it is evident that after two system periods, the system has not quite returned to its initial state. (The true period of a simple pendulum is longer than the one computed under the harmonic motion assumption.) Reducing the initial angle improves the repeatability of the wave patterns.

Copyright 2011 The MathWorks, Inc.

Double Wishbone Suspension

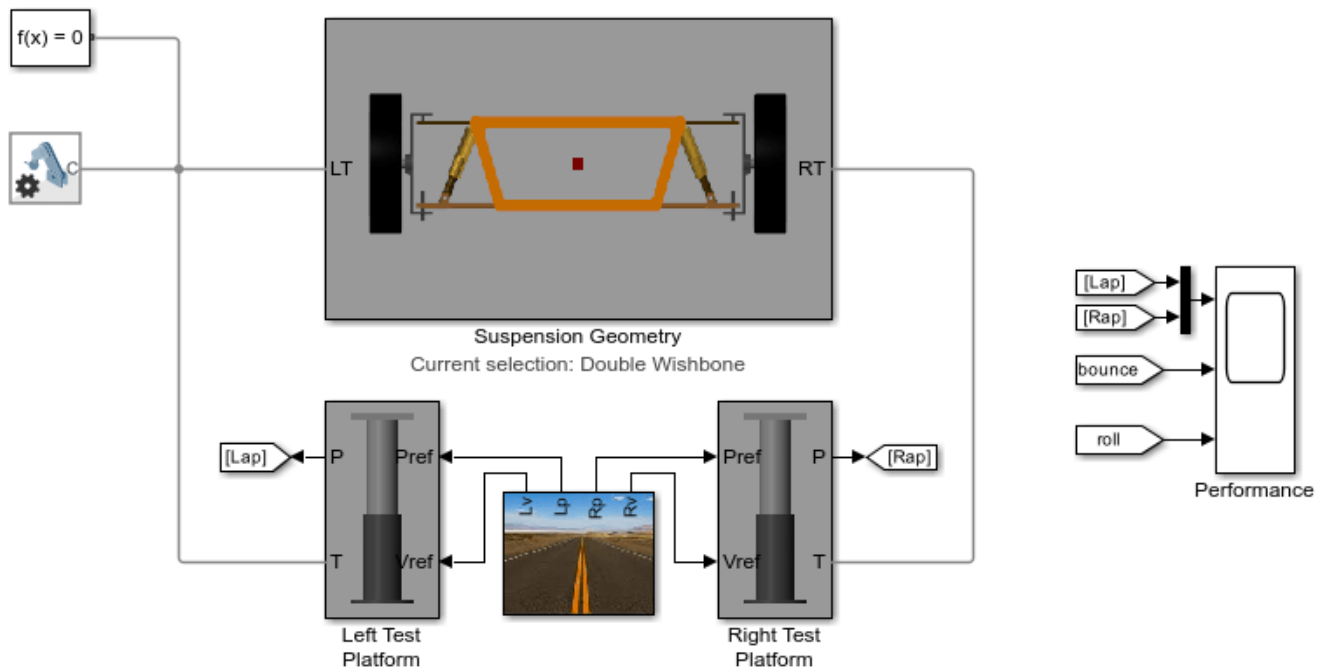
This example illustrates a double wishbone front wheel automotive suspension. The suspension is mounted on two platforms that can independently move up and down to simulate a road profile on each wheel. For a given pair of road profiles, the resultant roll and bounce of the chassis can be studied and the suspension parameters tuned for optimal performance. The inputs to the two platforms are the road profile and its derivative. The platforms have a PD controller that controls the vertical position of the platform to mimic the input road profile. See "Road Profiles Generator" block dialog for details on the test road profiles.



Copyright 2012-2013 The MathWorks, Inc.

Independent Suspension System Templates

This example includes templates for three common types of automotive independent suspension systems: double wishbone, MacPherson, and pushrod. Tires attached to the suspension systems are mounted on platforms that can independently move up and down. Each platform has a PD controller that allows it to simulate a desired road profile. For a given pair of road profiles, the resultant roll and bounce of the chassis can be studied and the suspension parameters can be tuned for optimal performance.



Independent Suspension System Templates

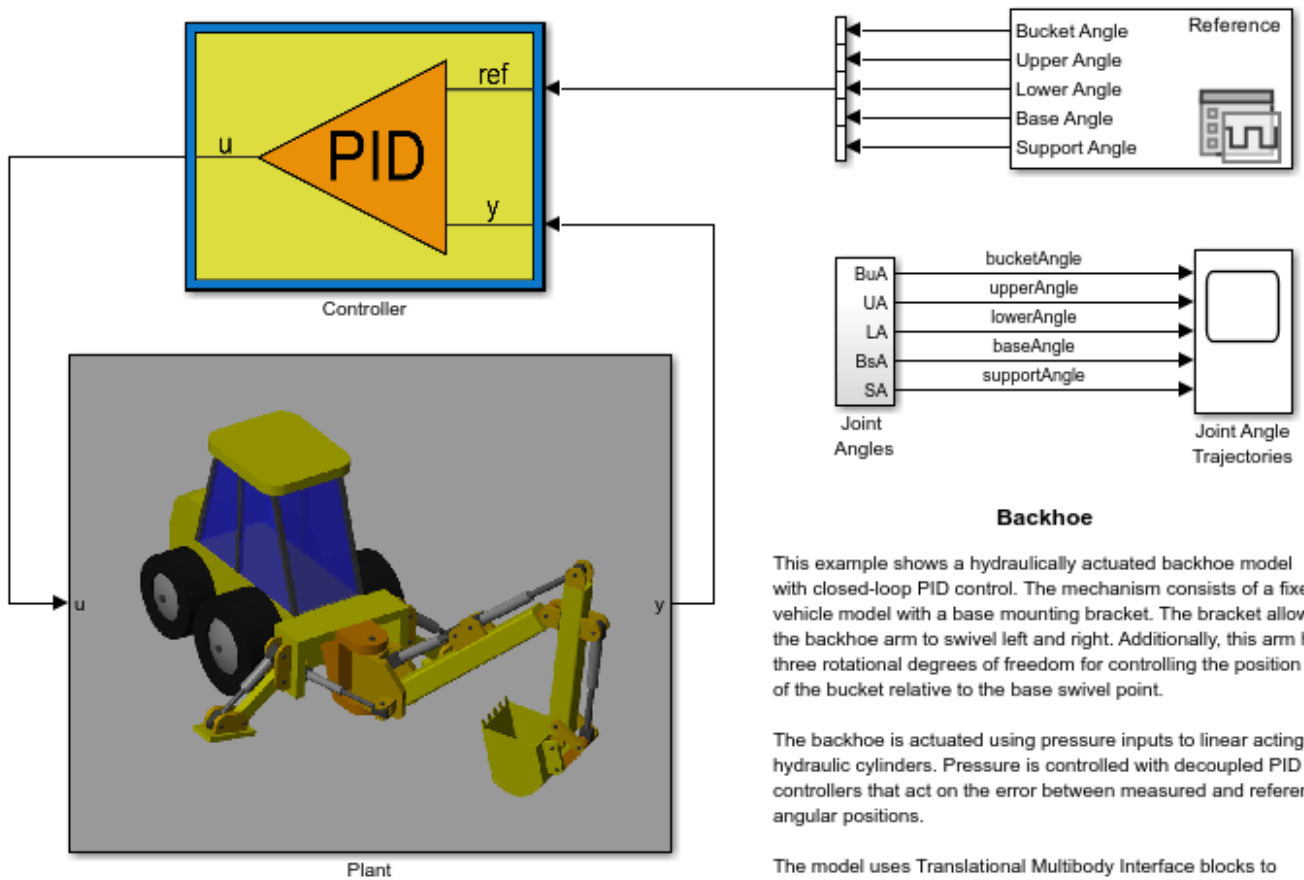
1. Double-click on the [Suspension Geometry](#) block to change suspension geometry to [double wishbone](#), [MacPherson](#), or [pushrod](#)
2. Double-click on the [Road Profiles](#) block to modify the motion of the test platforms
3. [Explore simulation results](#) using [Simscape Results Explorer](#)
4. [Learn more](#) about this example
5. Learn more about [multibody modeling](#)

Backhoe

This example shows a hydraulically actuated backhoe model with closed-loop PID control. The mechanism consists of a fixed vehicle model with a base mounting bracket. The bracket allows the backhoe arm to swivel left and right. Additionally, this arm has three rotational degrees of freedom for controlling the position of the bucket relative to the base swivel point.

The backhoe is actuated using pressure inputs to linear acting hydraulic cylinders. Pressure is controlled with decoupled PID controllers that act on the error between measured and reference angular positions.

The model uses Translational Multibody Interface blocks to connect 1D mechanical translational domain lines in Simscape (used by Translational Mechanical Converter blocks to model the hydraulic cylinders) with Prismatic Joint blocks in Simscape Multibody (used to model the backhoe mechanism). Additionally, position signals from the Prismatic Joint blocks are fed directly into the Translational Mechanical Converter blocks to ensure that Simscape and Simscape Multibody always agree on the value of each cylinder's displacement.



See Also

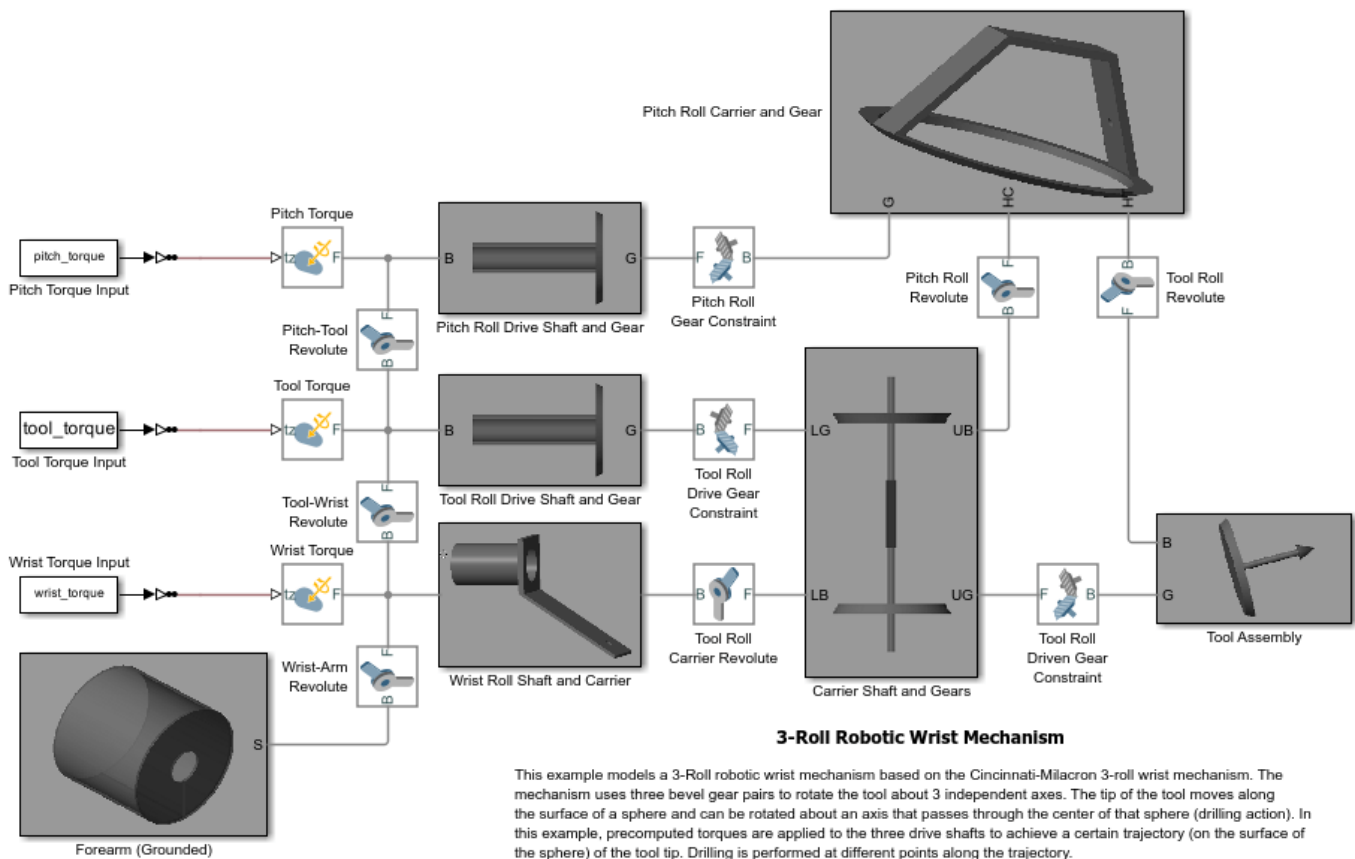
Prismatic Joint | Translational Multibody Interface

More About

- “Connecting Simscape Networks to Simscape Multibody Joints”
- “Hydraulic Interface - Dump Trailer with Hydraulic Cylinder” on page 8-58
- “Modeling a Double-Acting Actuator”

3-Roll Robotic Wrist Mechanism

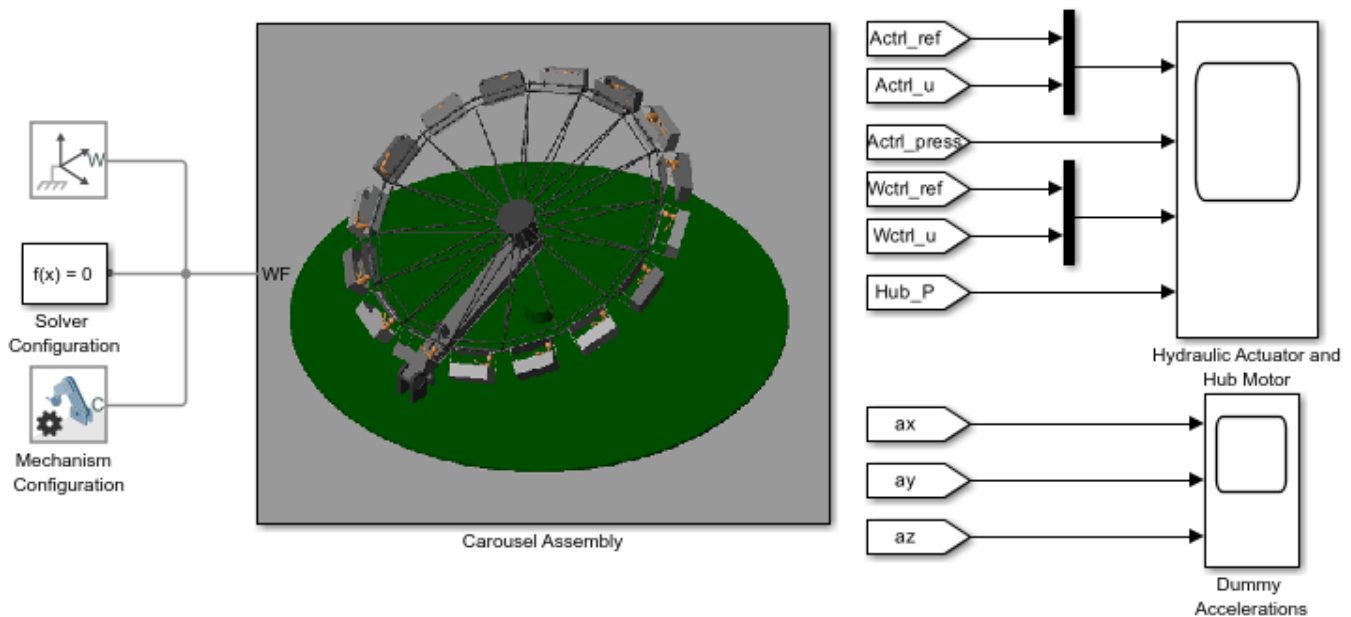
This example models a 3-Roll robotic wrist mechanism based on the Cincinnati-Milacron 3-roll wrist mechanism. The mechanism uses three bevel gear pairs to rotate the tool about 3 independent axes. The tip of the tool moves along the surface of a sphere and can be rotated about an axis that passes through the center of that sphere (drilling action). In this example, precomputed torques are applied to the three drive shafts to achieve a certain trajectory (on the surface of the sphere) of the tool tip. Drilling is performed at different points along the trajectory.



Fairground Carousel Ride

This example shows a fairground carousel ride. A torque applied to the wheel causes the carousel to rotate and a hydraulic actuator provides the force to lift the arm. The cabs are free to rotate about an axis approximately tangential to the wheel radius. When the wheel is near vertical, the centrifugal acceleration acting on the cabs (caused by the rotation of the wheel) ensures that the cabs are close to a near vertical position. Consequently, the riders are close to 'up-side-down' at the top of the rotation.

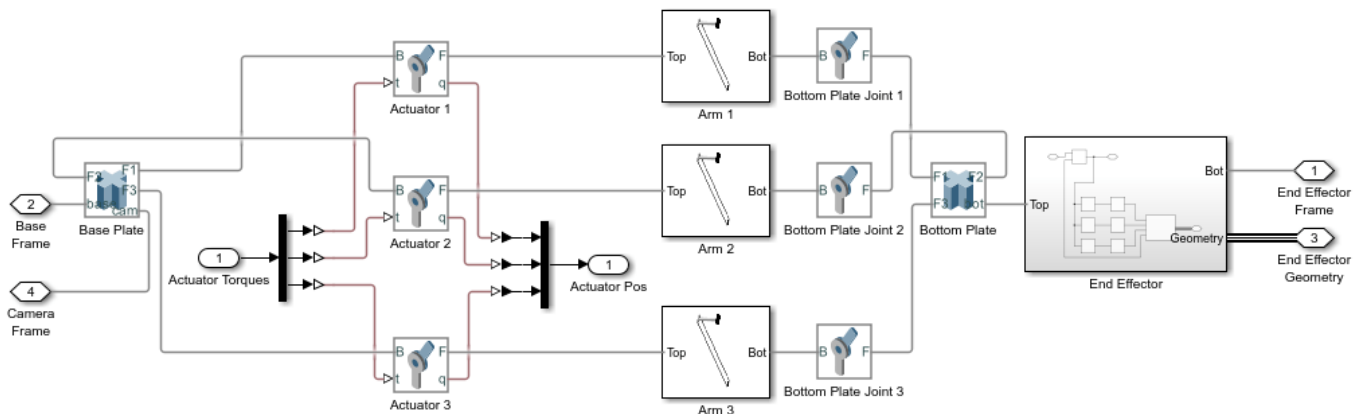
The acceleration at the center of the torso of a dummy is measured to show whether the ride is 'fun' and 'safe'. In addition, the hub motor power and hydraulic pressure are measured to validate the design. The control signals (reference and actual) for the hub speed and lift-arm angle are plotted to show the performance of the controllers.



Fairground Carousel Ride

This example shows a fairground carousel ride. A torque applied to the wheel causes the carousel to rotate and a hydraulic actuator provides the force to lift the arm. The cabs are free to rotate about an axis approximately tangential to the wheel radius. When the wheel is near vertical, the centrifugal acceleration acting on the cabs (caused by the rotation of the wheel) ensures that the cabs are close to a near vertical position. Consequently, the riders are close to 'up-side-down' at the top of the rotation.

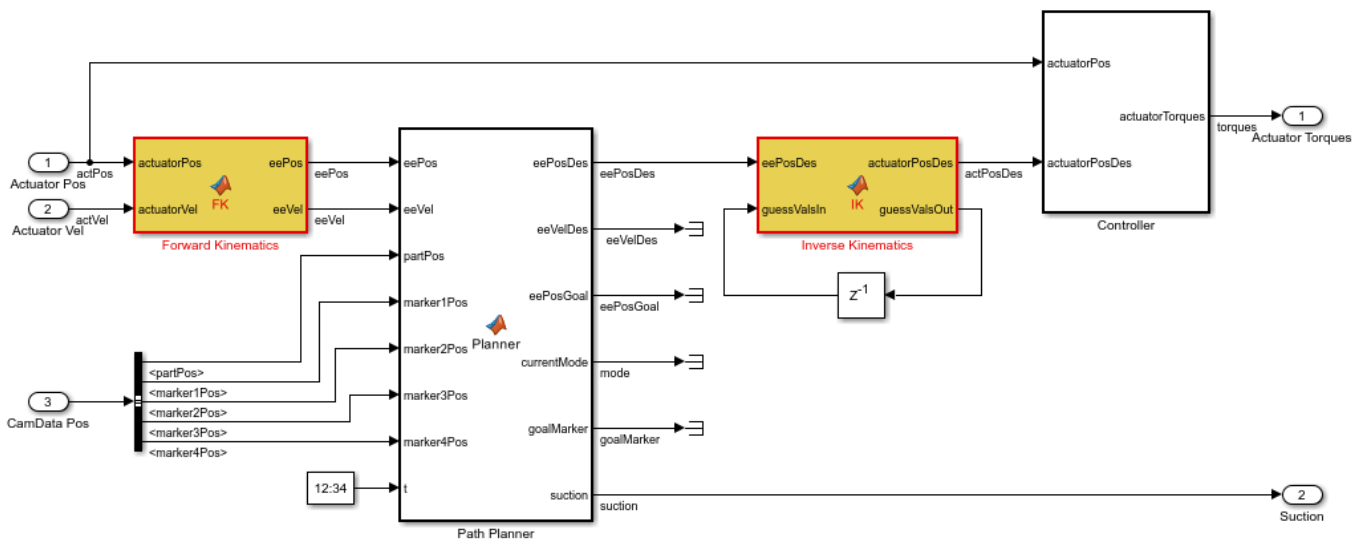
The acceleration at the center of the torso of a dummy is measured to show whether the ride is 'fun' and 'safe'. In addition, the hub motor power and hydraulic pressure are measured to validate the design. The control signals (reference and actual) for the hub speed and lift-arm angle are plotted to show the performance of the controllers.



Planning and Control Subsystem: Forward and Inverse Kinematics

Because trajectory planning for the end effector is done with respect to the xyz coordinates of the robot's camera frame, a forward kinematics map is needed to transform the positions and velocities of the actuators to the position and velocity of the end effector. Similarly, an inverse kinematics map is needed to transform the desired position of the end effector computed by the planner to the corresponding positions of the three actuators. These forward and inverse kinematics computations are done using `KinematicsSolver` objects. The objects are defined as persistent variables in the functions `sm_pick_and_place_robot_fk` and `sm_pick_and_place_robot_ik`. These functions are called by the MATLAB function blocks Planning and Control/Forward Kinematics and Planning and Control/Inverse Kinematics highlighted below. To speed up computation and help ensure the `KinematicsSolver` object for the inverse kinematics problem finds the desired solutions, the previous solution is used as the initial guess for the current problem. Whenever the parameters of the Delta Robot subsystem change, the `sm_pick_and_place_robot_fk` and `sm_pick_and_place_robot_ik` functions are cleared from memory so that the `KinematicsSolver` objects are regenerated at the beginning of the next simulation. This ensures that the `KinematicsSolver` objects and the model stay in sync.

Open Planning and Control Subsystem



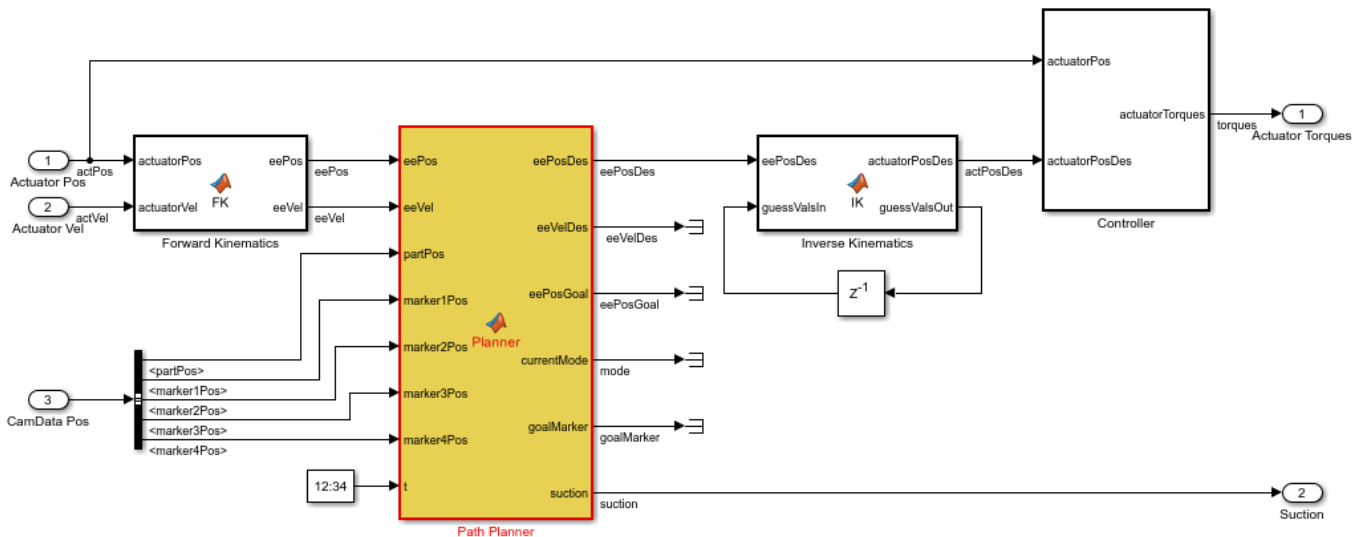
Planning and Control Subsystem: Path Planner

Planning occurs in the MATLAB Function block Planning and Control/Path Planner highlighted below. The planner transitions the robot between three different modes:

- go to location directly above part
- grasp part and move to goal location
- go home

Whenever a mode begins, a trajectory is computed that takes the end effector from its current position to the mode's goal position in a fixed amount of time. The trajectory is generated in two stages: first, a third-order polynomial is computed corresponding to the path of the end effector in xyz camera coordinates from its current position to the goal position; second, a fifth-order polynomial is computed which is used to scale the time along the path such that the initial and final velocities and accelerations are all zero. A mode transition occurs when the position and velocity of the end effector are sufficiently close to the goal values. Given the current time, the planner returns the desired position and velocity of the end effector along the trajectory as well as the desired state of the vacuum.

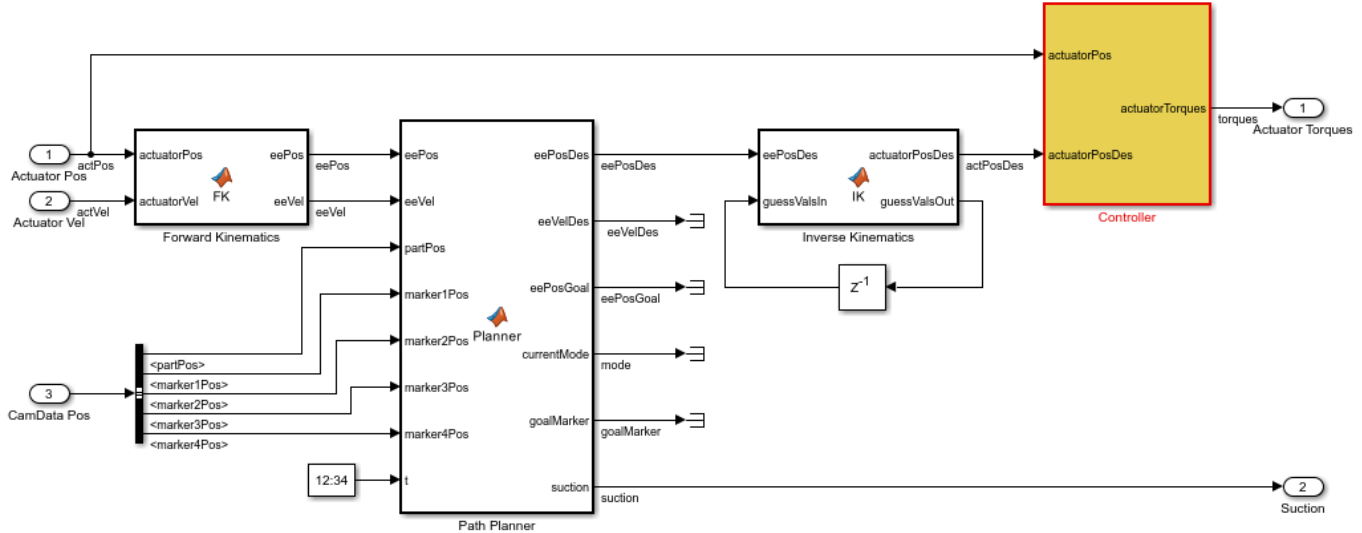
Open Planning and Control Subsystem



Planning and Control Subsystem: Controller

The Planning and Control/Controller subsystem highlighted below contains a discrete time PID controller that drives the actual positions of the actuators to their desired values.

Open Planning and Control Subsystem



Vacuum Subsystem

To grasp the part, a simple vacuum is modeled between the part and the robot's end effector. Whenever the planner commands suction, the vacuum applies a constant force between the center of mass of the part and the tip of the end effector.

Open Vacuum Subsystem

Contact Subsystems

Spatial Contact Force blocks inside the End Effector-Part Contact Forces and Part-Table Contact Forces subsystems are used to model contact. To speed up the simulation, three contact points equally spaced around the tip of the end effector are used as proxies for full cylindrical geometry when it is in contact with the part. Similarly, three equally spaced contact points around the bottom edge of the part are used as contact proxies for when it makes contact with the table. The vacuum force keeps the part in contact with the end effector, and friction prevents it from slipping during transport.

Open End Effector-Part Contact Forces Subsystem

Open Part-Table Contact Forces Subsystem

Image Processor

The Image Processor subsystem uses Transform Sensor blocks to simulate the processing of camera data to track the locations of the part and the markers on the table.

Open Image Processor Subsystem

See Also

[KinematicsSolver](#) | [solve](#)

More About

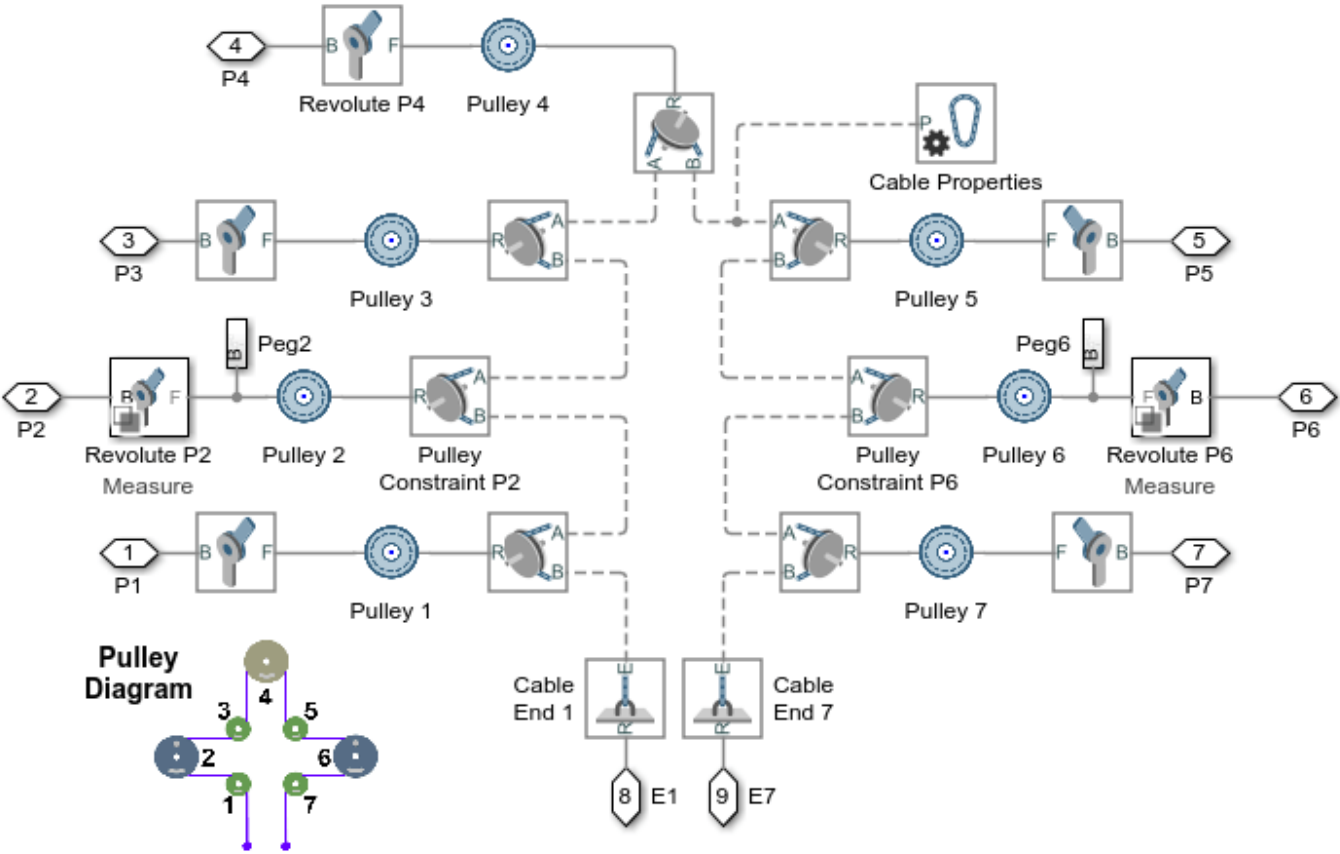
- “Perform Forward and Inverse Kinematics on a Five-Bar Robot” on page 8-162

Cable-Driven XY Table with Cross Base

This example models an XY positioning table that uses a cable-driven mechanism. A single cable wraps around seven different pulleys and converts the rotational angle of the two input pulleys to the x-y position of the table.

Inverse kinematics can be used to map table position to pulley angle. The model allows you to specify the motion of the table in x-y coordinates and determine the required pulley rotation to produce that movement. Inverse dynamics can be used to calculate the torque required to produce that motion.

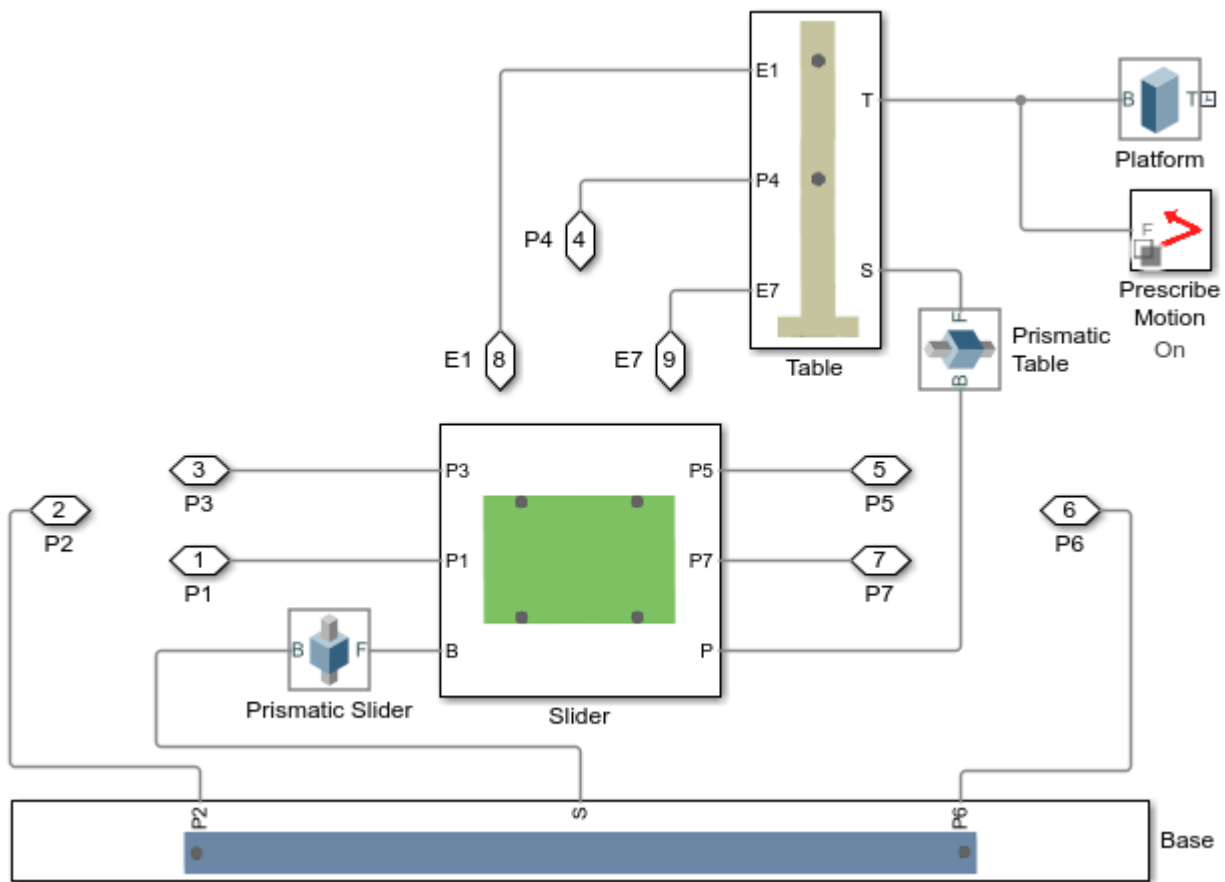
Model



Platform Subsystem

This subsystem models the platform that has two degrees of freedom. The slider and the table are constrained by two prismatic joints which permit movement along two perpendicular axes. The mounting points for all seven pulleys are defined in this subsystem.

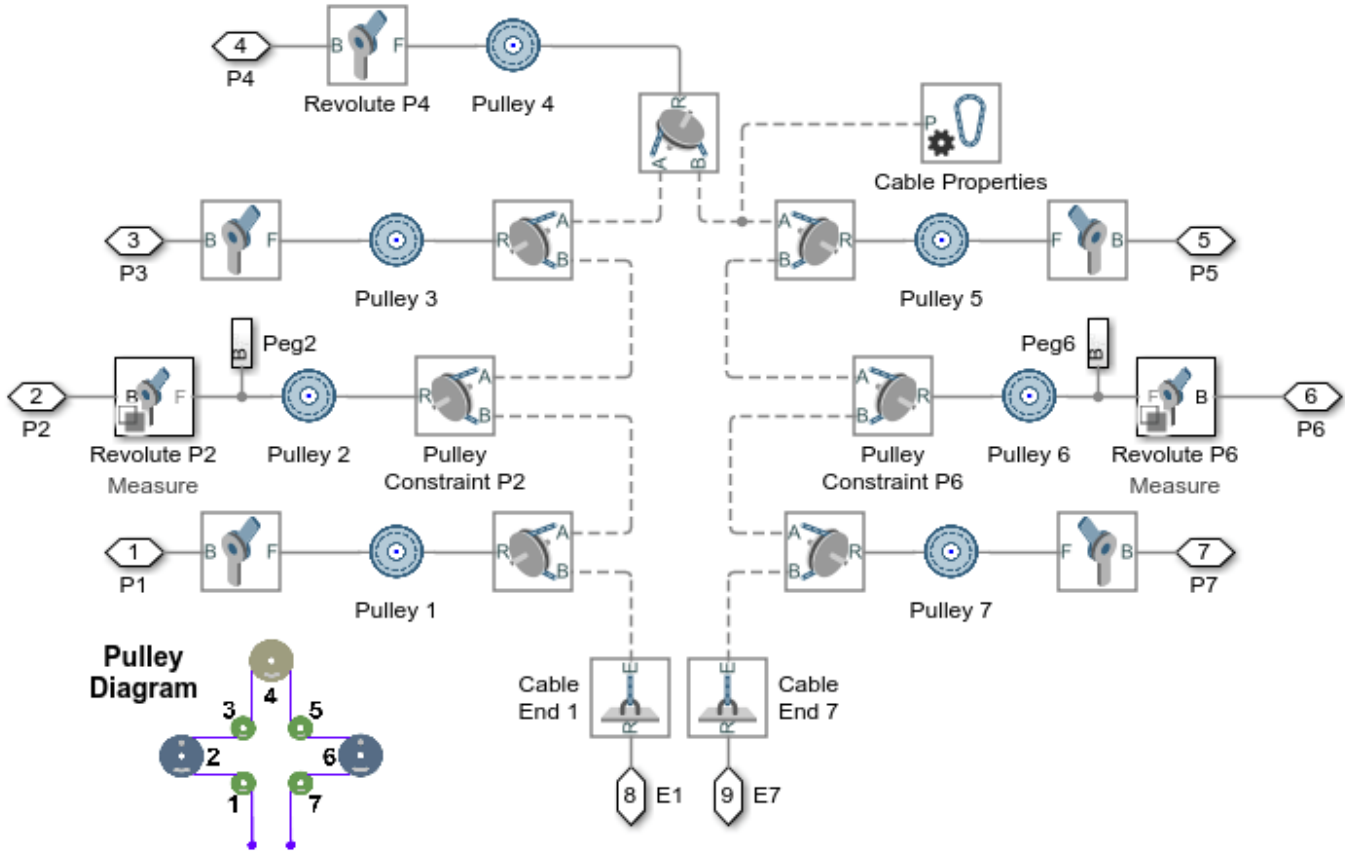
Open Subsystem



Pulleys Subsystem

This subsystem models the seven pulleys which are connected by a single cable. The pulley constraints and the cable connections ensure that the rotation of the individual pulleys follows the kinematic behavior as specified in the diagram. The cable ends attach to points on the upper part of the platform.

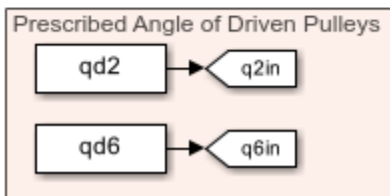
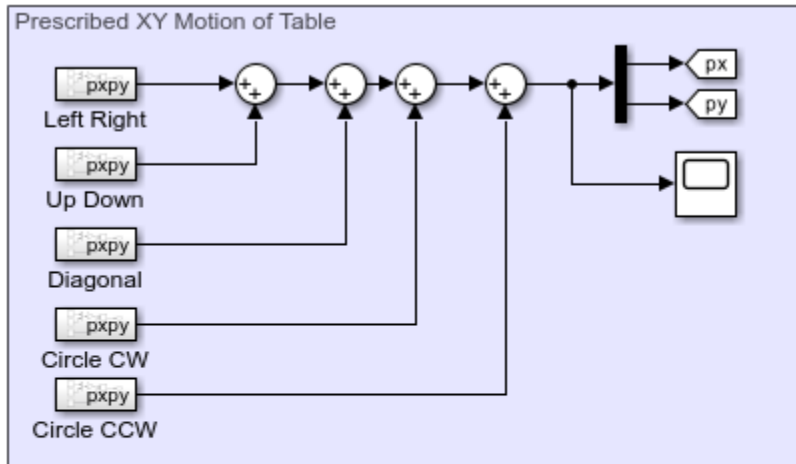
Open Subsystem



Motion Subsystem

This subsystem shows the inputs that can be used to prescribe the motion of the table. The upper set of inputs prescribes the motion of the table in x-y coordinates, and an inverse kinematic simulation can determine the required rotations of pulleys 2 and 6 to achieve that movement. The lower set of inputs prescribe the angles of pulleys 2 and 6. This data was recorded from the inverse kinematic simulation.

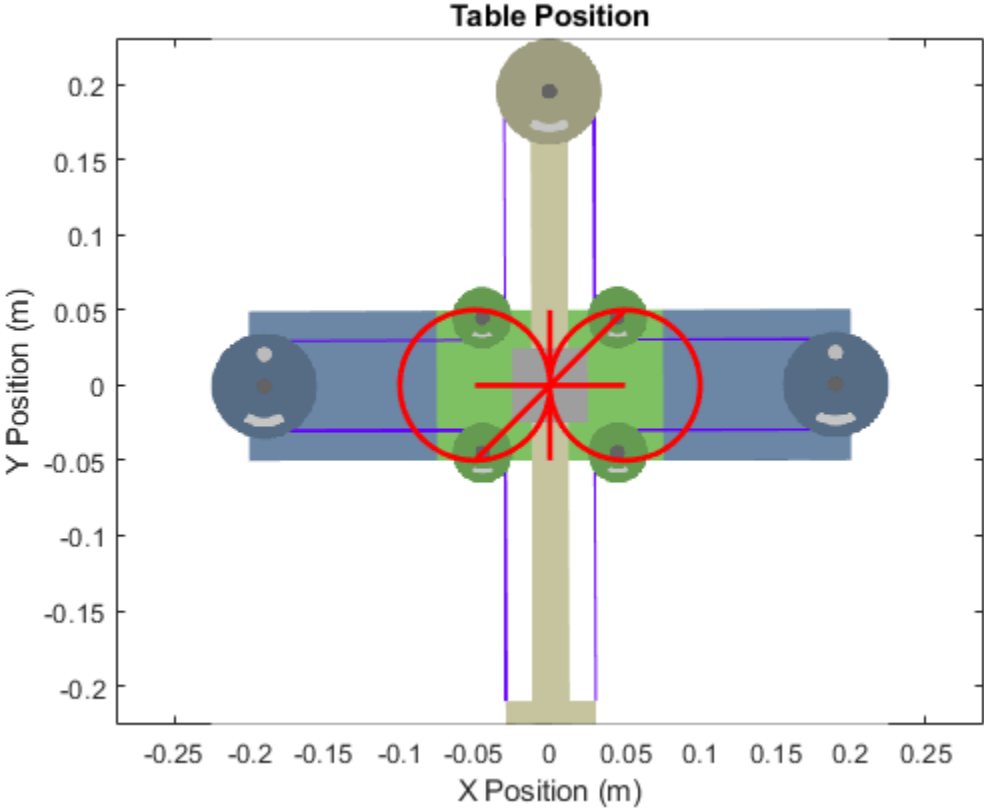
Open Subsystem



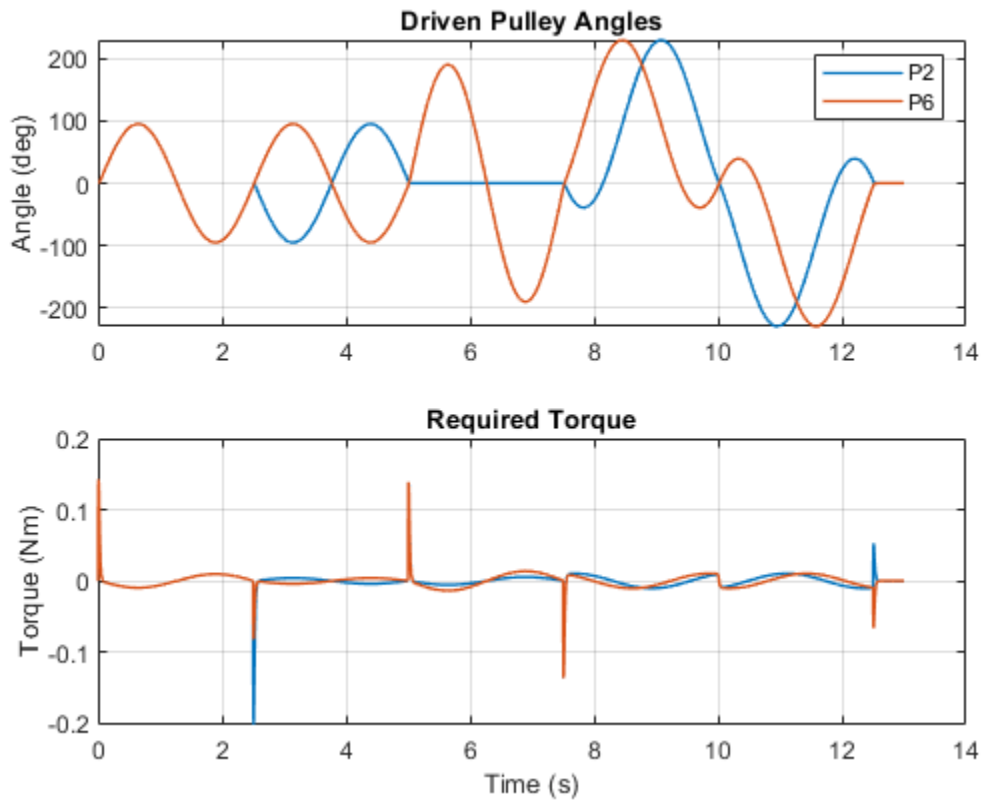
Only one of these is used at a time. Variant subsystems are used to control which inputs are used.

Simulation Results from Simscape Logging

This plot shows the XY position of the table.



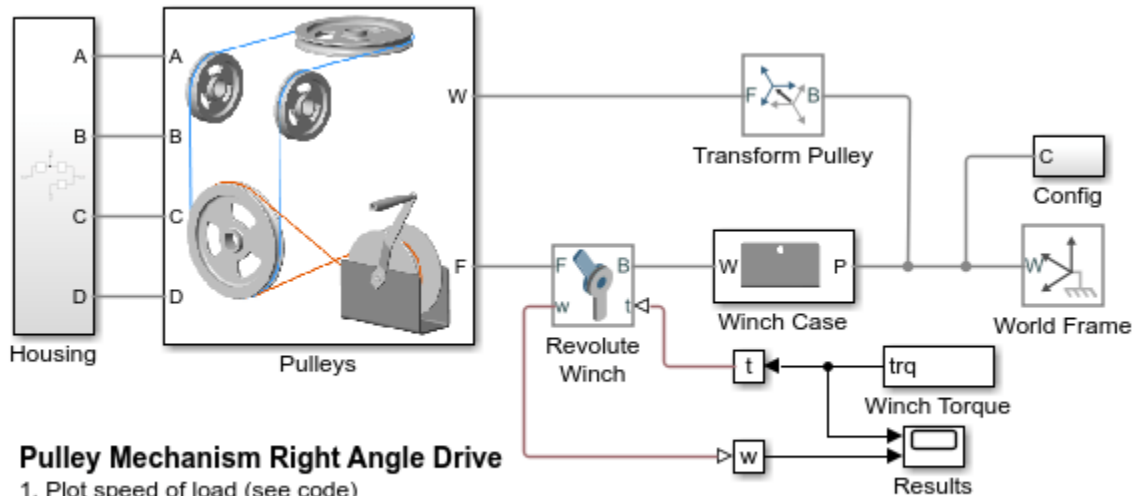
The plots below show the required motion and torques for pulley 2 and pulley 6 to produce the desired motion of the table.



Pulley Mechanism Right Angle Drive

This example models a pulley mechanism that takes a torque applied to a winch and transmits it to a pulley rotated at 90 degrees to that winch. This example uses blocks from the Simscape Multibody Belts and Cables library to model a pulley mechanism that is not all in a single plane.

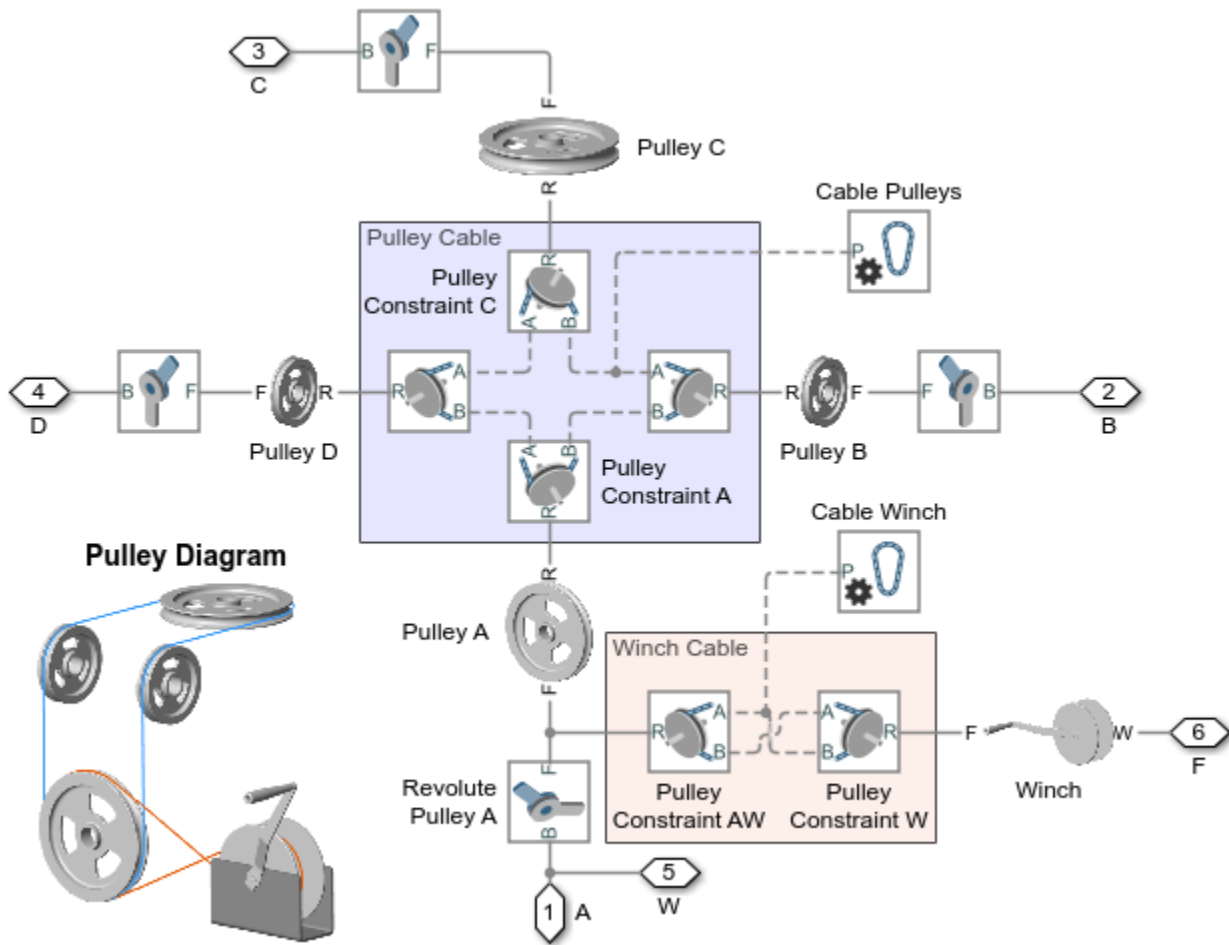
Model



1. Plot speed of load (see code)
2. Explore simulation results using Simscape Results Explorer
3. Learn more about this example

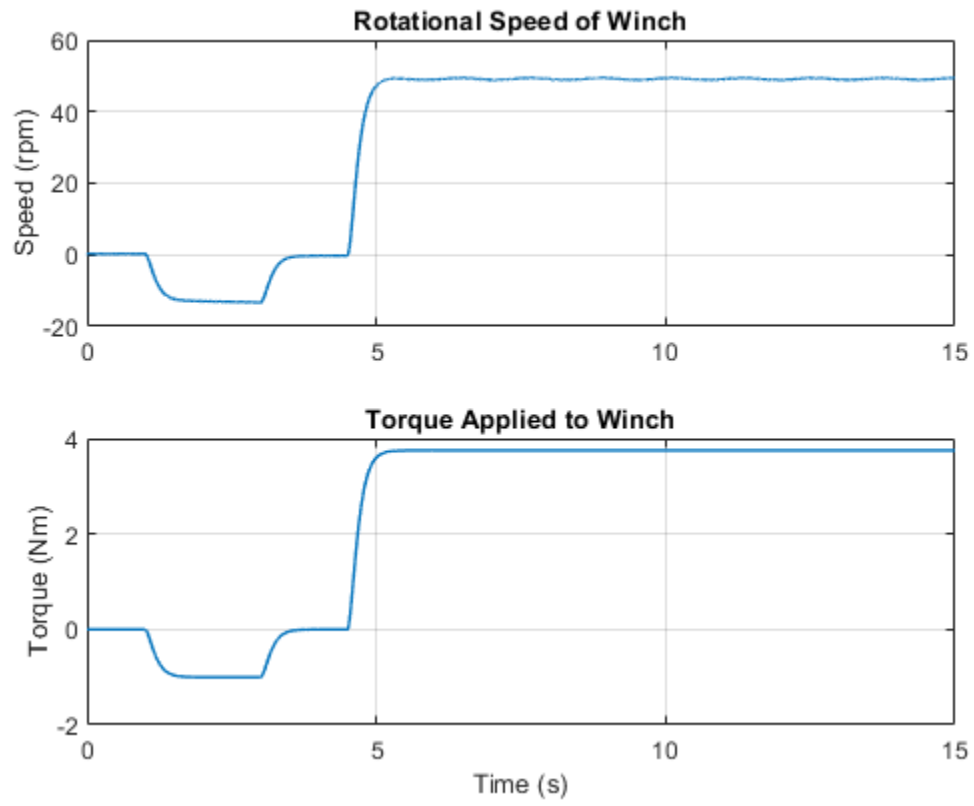
Pulleys Subsystem

Open Subsystem



Simulation Results from Simscape Logging

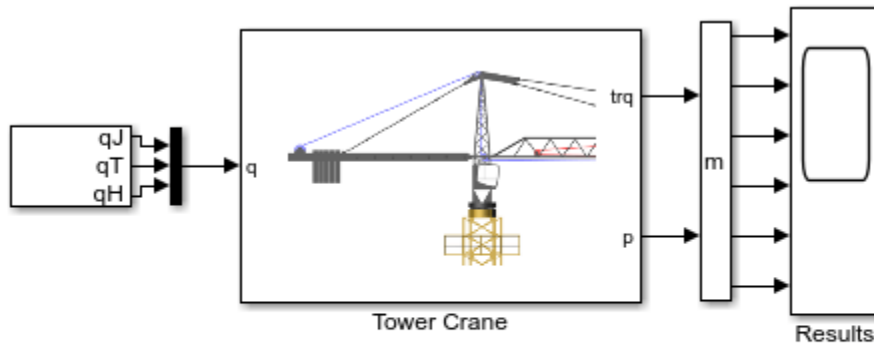
The plot below shows the torque applied to the winch and the speed of the winch. Note that for the first second, no torque is applied to the winch, but the load still moves due to gravity.



Tower Crane With Trolley and Hoist

This example models a tower crane with a trolley and a hoist. The hoist can raise and lower a load, and the trolley moves the load towards and away from the tower. Blocks from the belts and cables library are used to model the pulleys that control lifting the load and moving the trolley.

Model

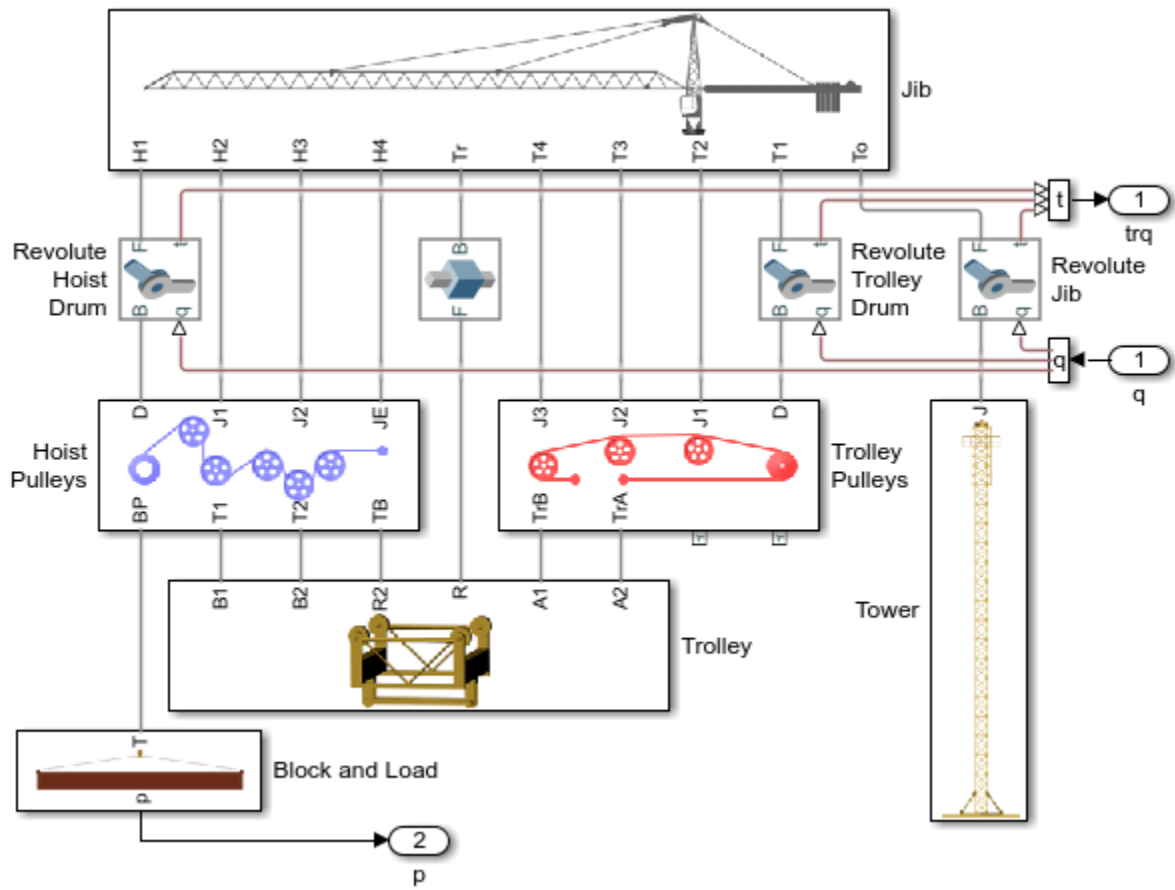


Tower Crane With Trolley and Hoist

1. Plot torque applied to hoist drum (see code)
2. Explore simulation results using Simscape Results Explorer
3. Learn more about this example

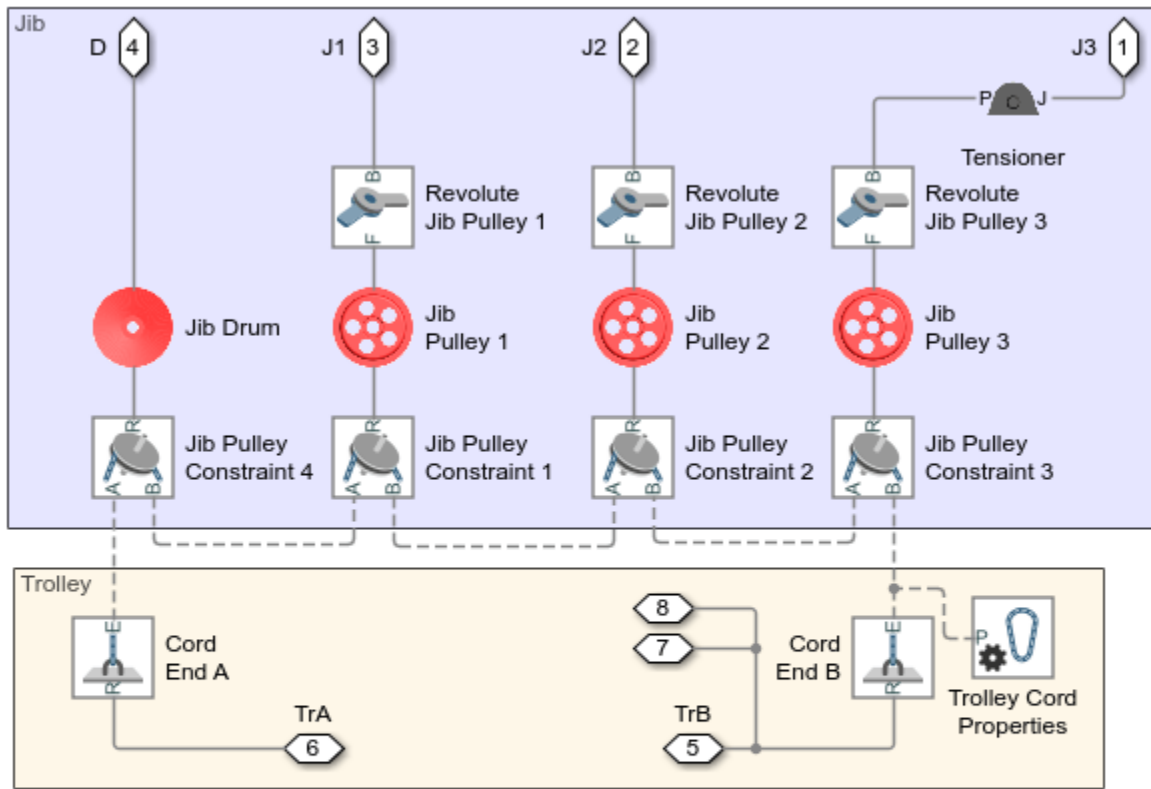
Tower Crane Subsystem

Open Subsystem



Trolley Pulleys Subsystem

Open Subsystem

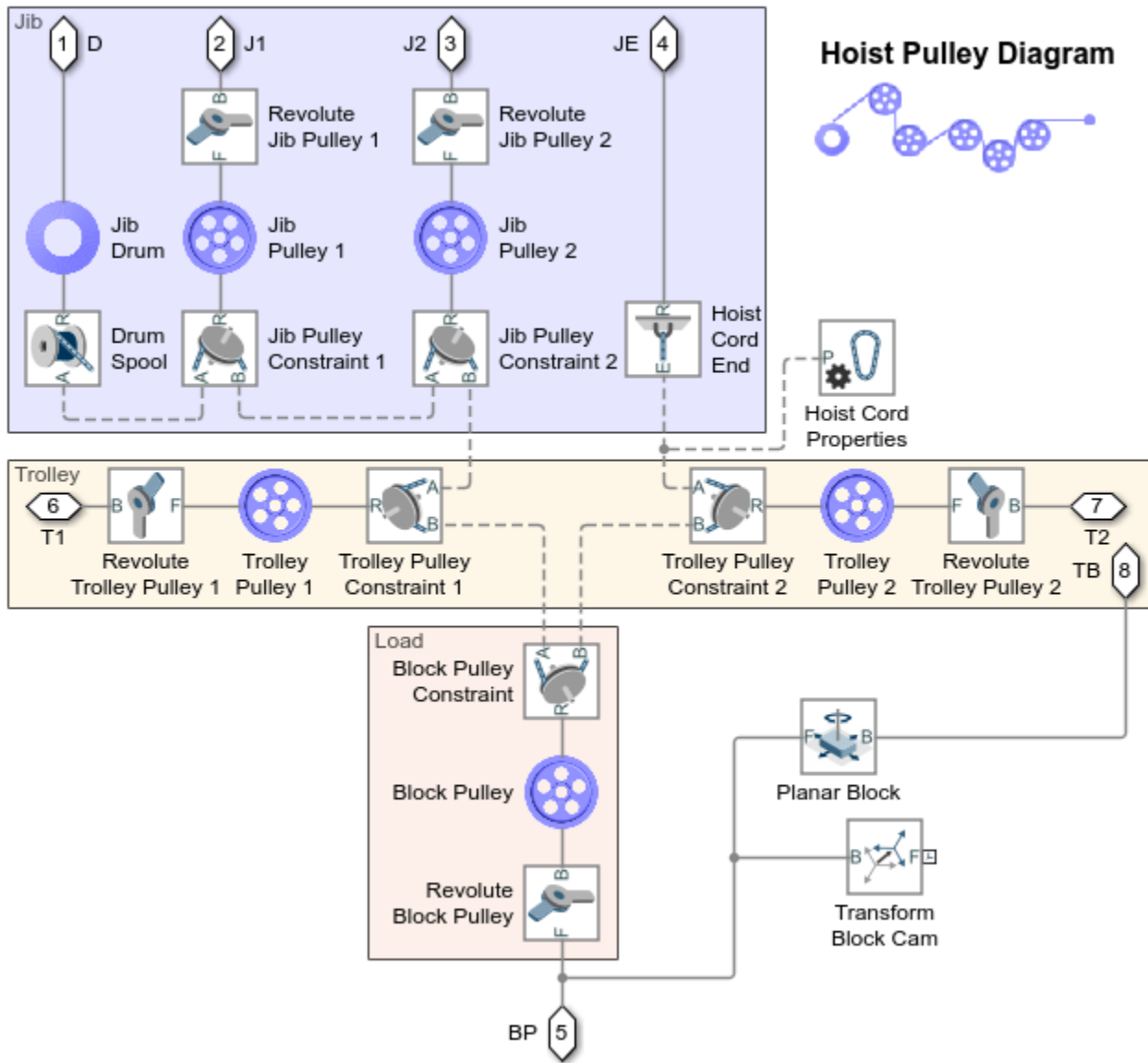


Trolley Pulley Diagram



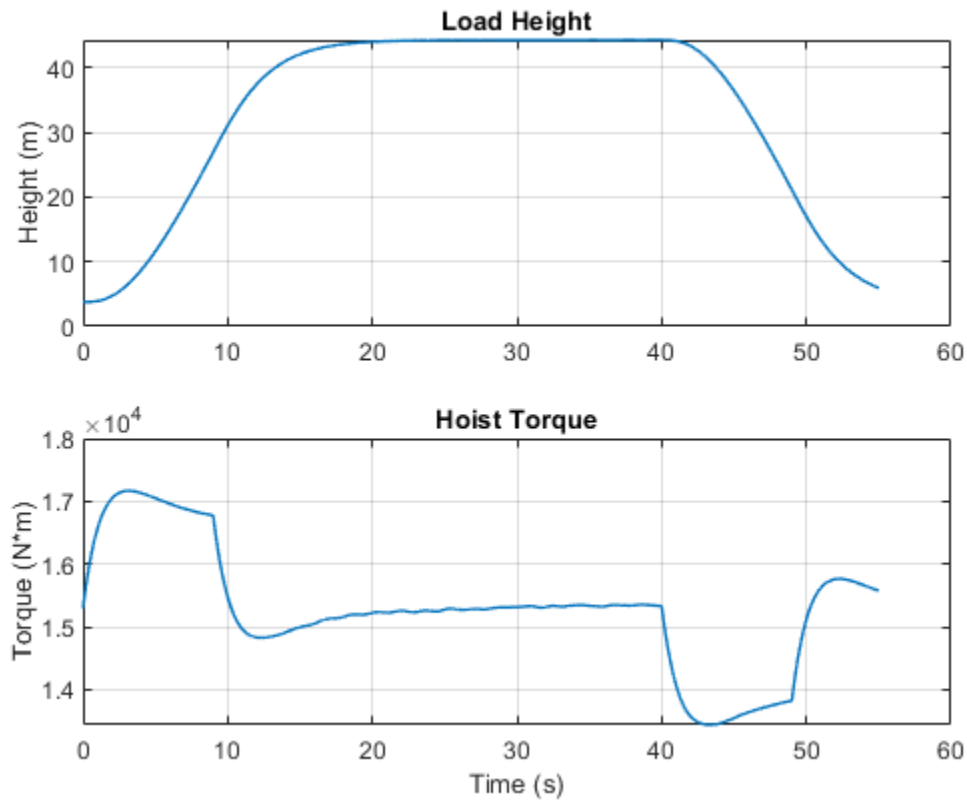
Hoist Pulleys Subsystem

Open Subsystem



Simulation Results from Simscape Logging

The plot below shows the load height and the torque applied to the hoist drum.



See Also

[Belt-Cable End](#) | [Belt-Cable Properties](#) | [Belt-Cable Spool](#) | [Bushing Joint](#) | [Prismatic Joint](#) | [Pulley](#) | [Revolute Joint](#) | [Spatial Contact Force](#) | [Spherical Joint](#)

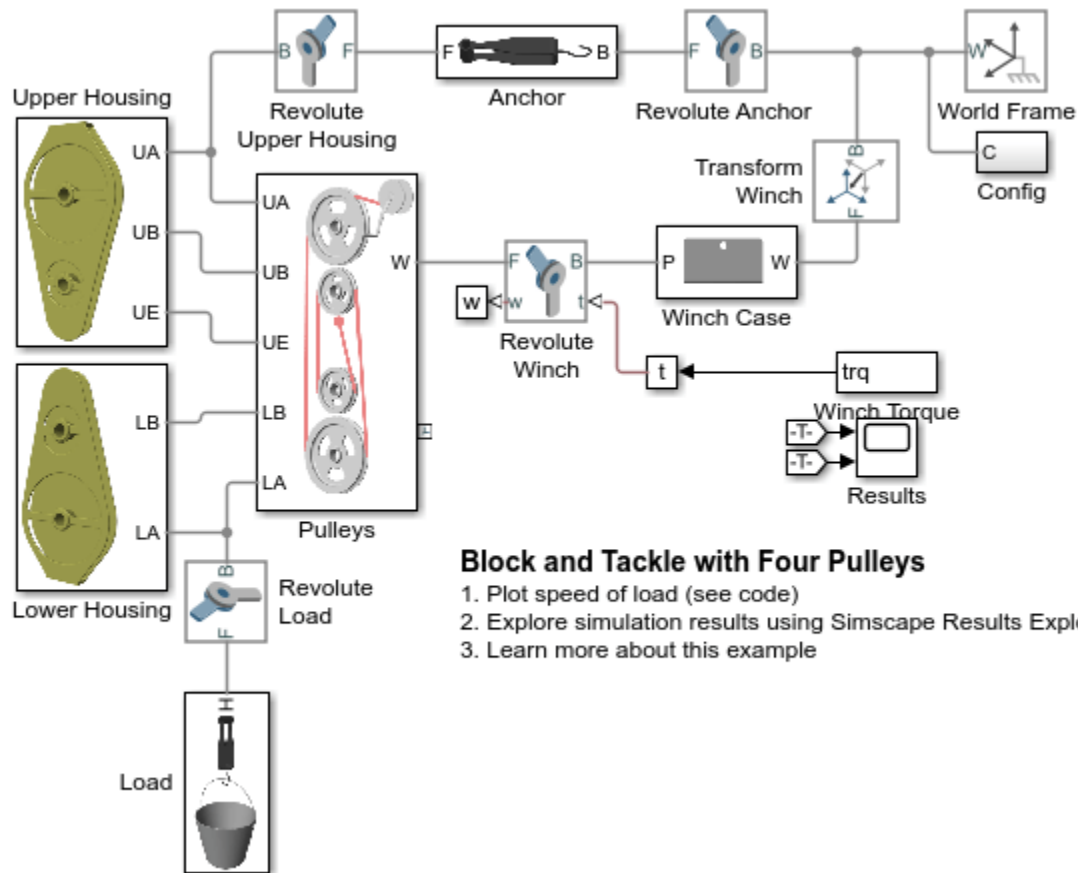
More About

- "Cable Driven Space Manipulator" on page 8-5
- "Cable Robot" on page 8-80
- "Block and Tackle with Four Pulleys" on page 8-95

Block and Tackle with Four Pulleys

This example models a block and tackle with four pulleys. Torque is applied to a winch which acts through the pulley mechanism to lift a load. Blocks from the Simscape Multibody Belts and Cables library are used to model the block and tackle.

Model

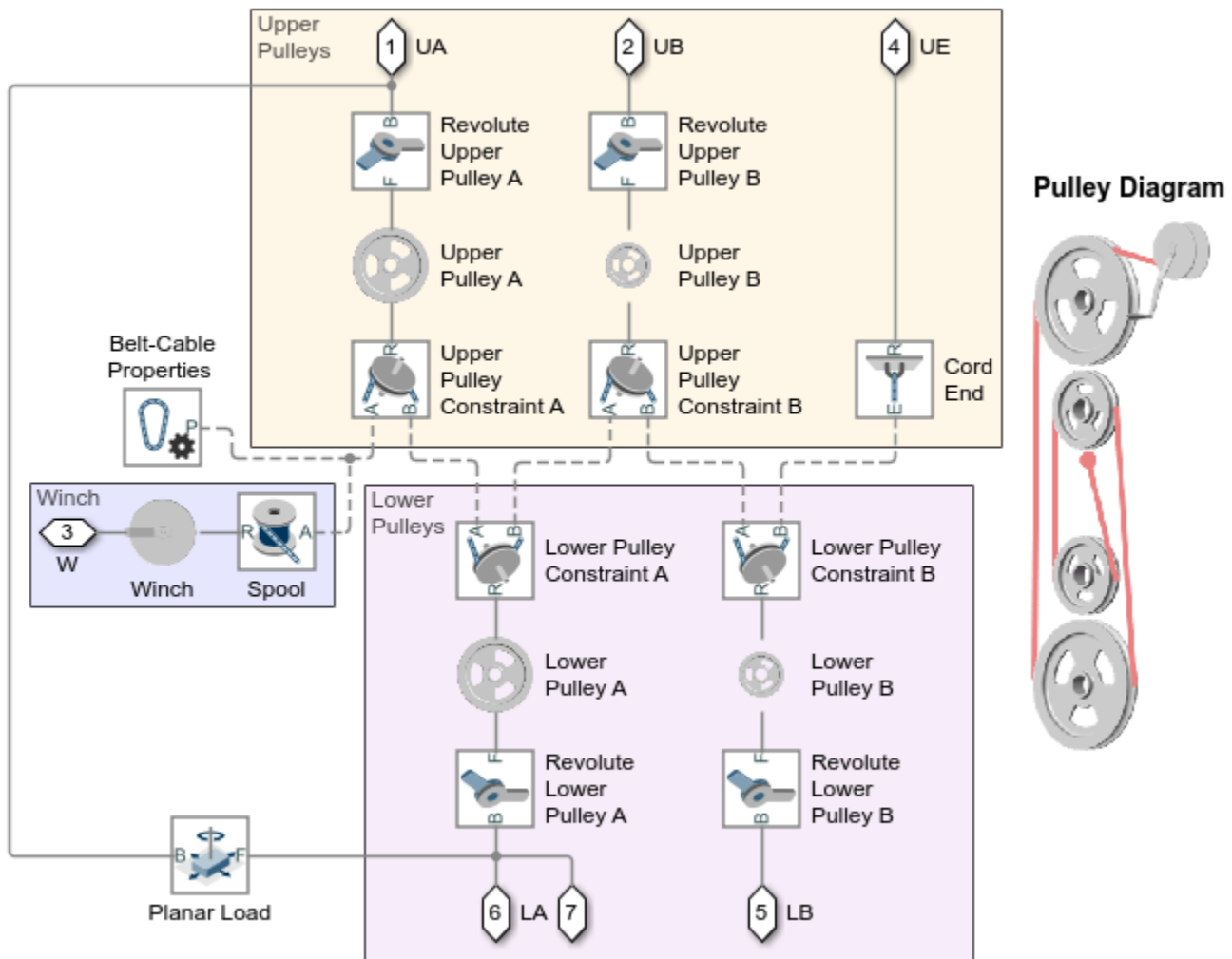


Block and Tackle with Four Pulleys

1. Plot speed of load (see code)
2. Explore simulation results using Simscape Results Explorer
3. Learn more about this example

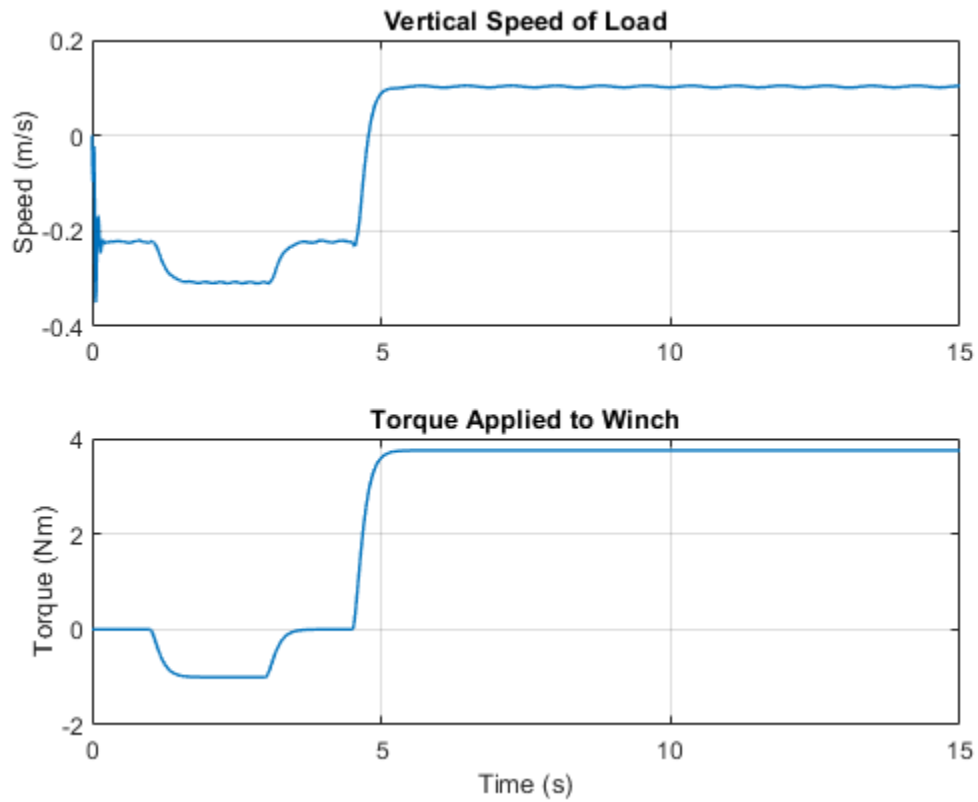
Pulleys Subsystem

Open Subsystem



Simulation Results from Simscape Logging

The plot below shows the torque applied to the winch and the vertical speed of the load. Note that for the first second, no torque is applied to the winch, but the load still moves due to gravity.



See Also

[Belt-Cable End](#) | [Belt-Cable Properties](#) | [Belt-Cable Spool](#) | [Planar Joint](#) | [Pulley](#) | [Revolute Joint](#)

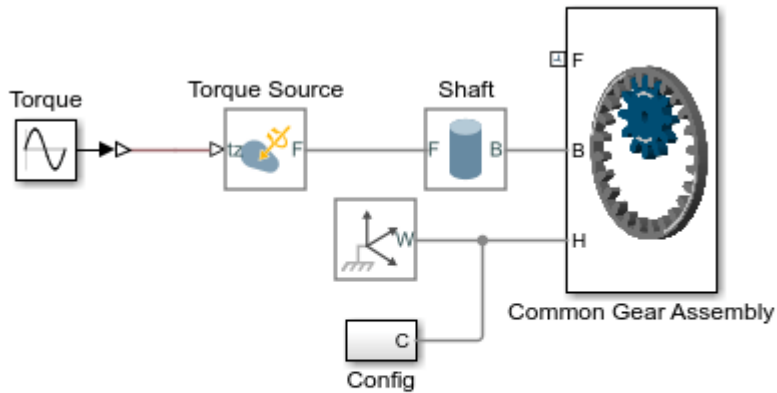
More About

- "Cable Robot" on page 8-80
- "Elevator" on page 8-2

Using the Common Gear Block

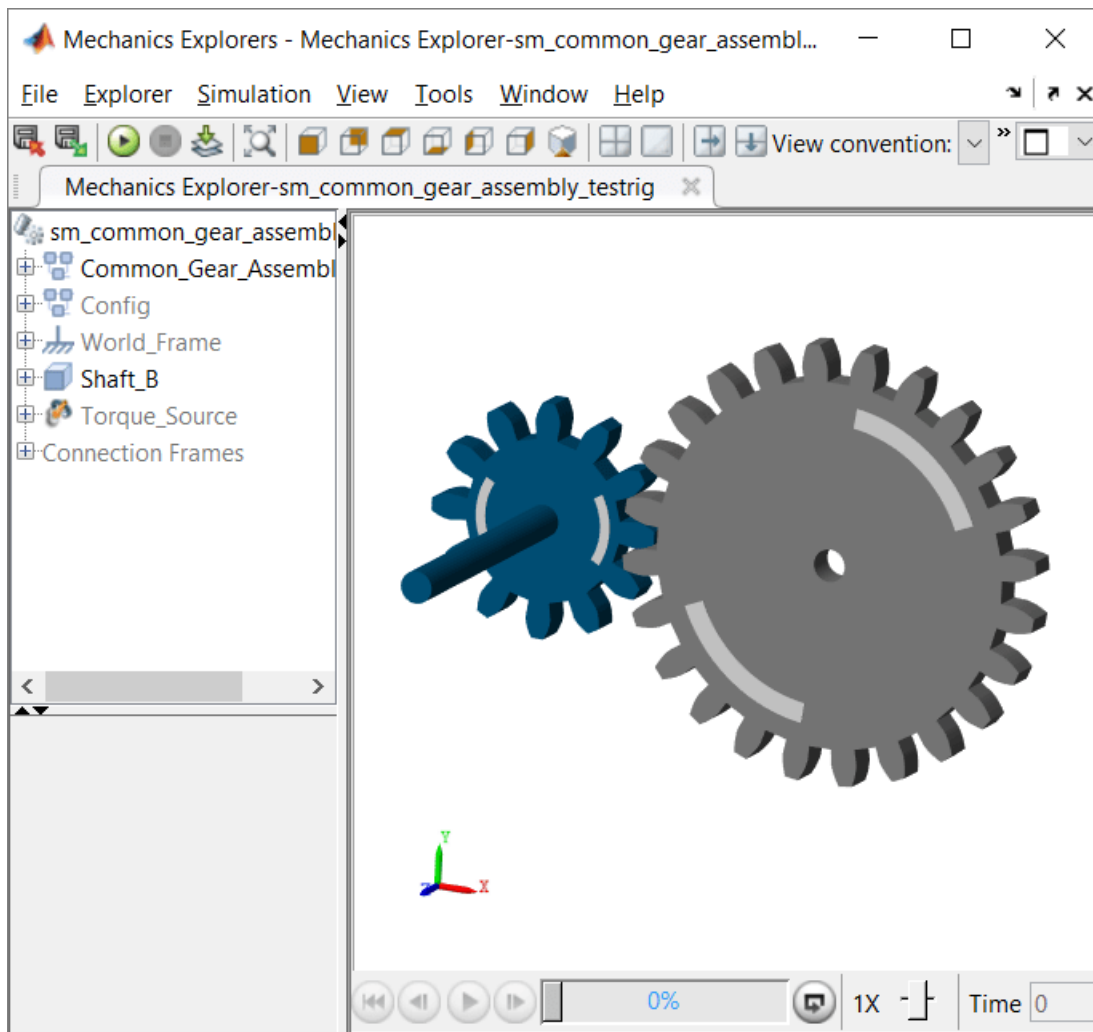
This example models a pair of gears with parallel axes. The assembly contains the two gears and all constraints required for the gear set. The assembly can be configured to model an external or internal gear set by adjusting parameters in the mask.

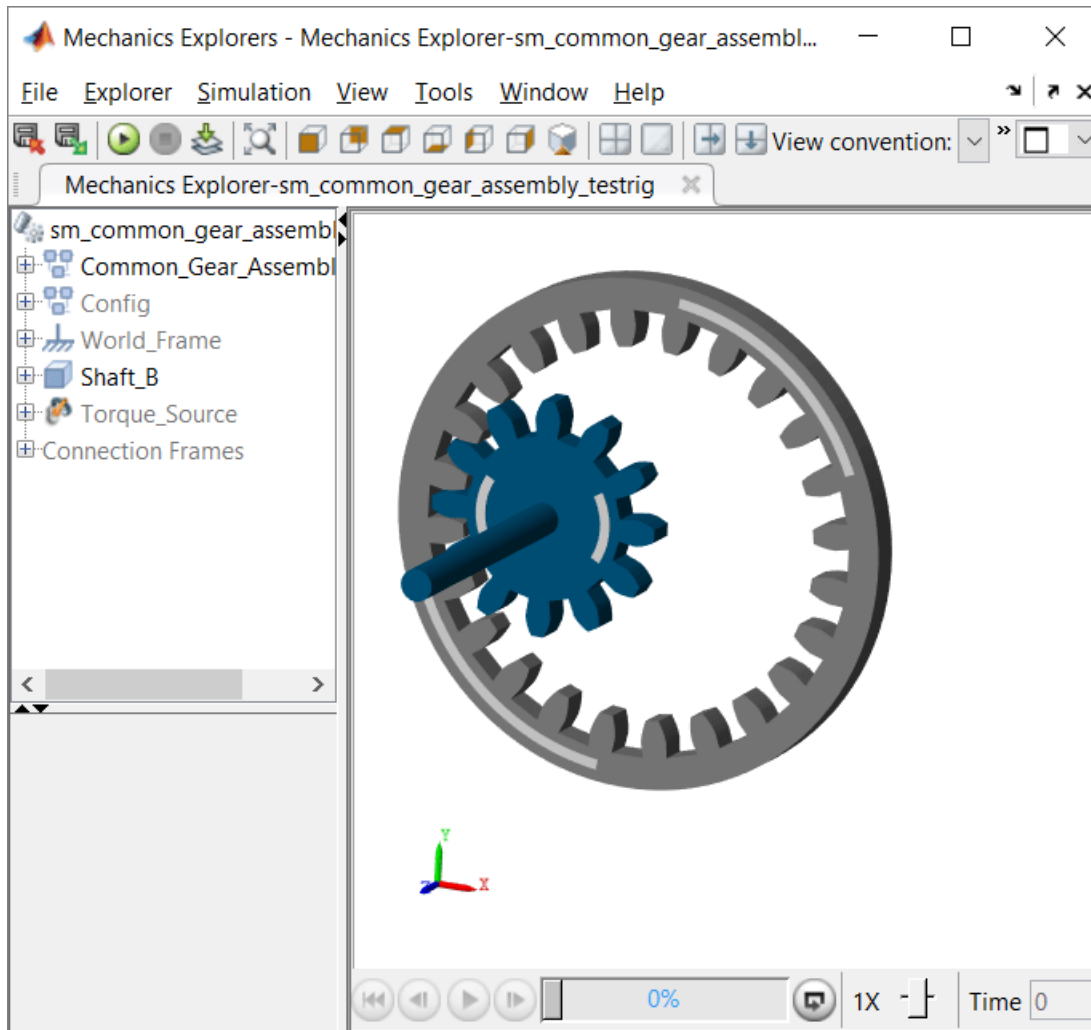
Model



Using the Common Gear Block

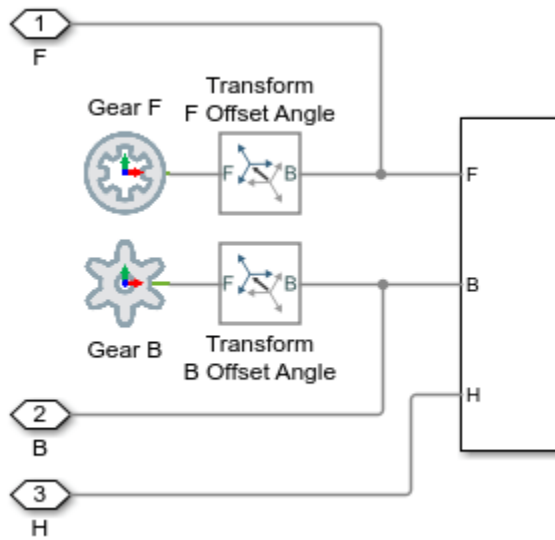
1. Plot speed of gears (see code)
2. Configure gear assembly: External, Internal (see code)
3. Explore simulation results using Simscape Results Explorer
4. Learn more about this example





Common Gear Assembly Subsystem

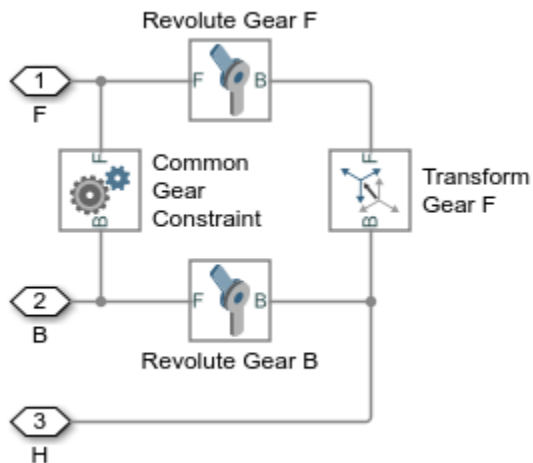
Open Subsystem



Common Gear Assembly Constraints Subsystem

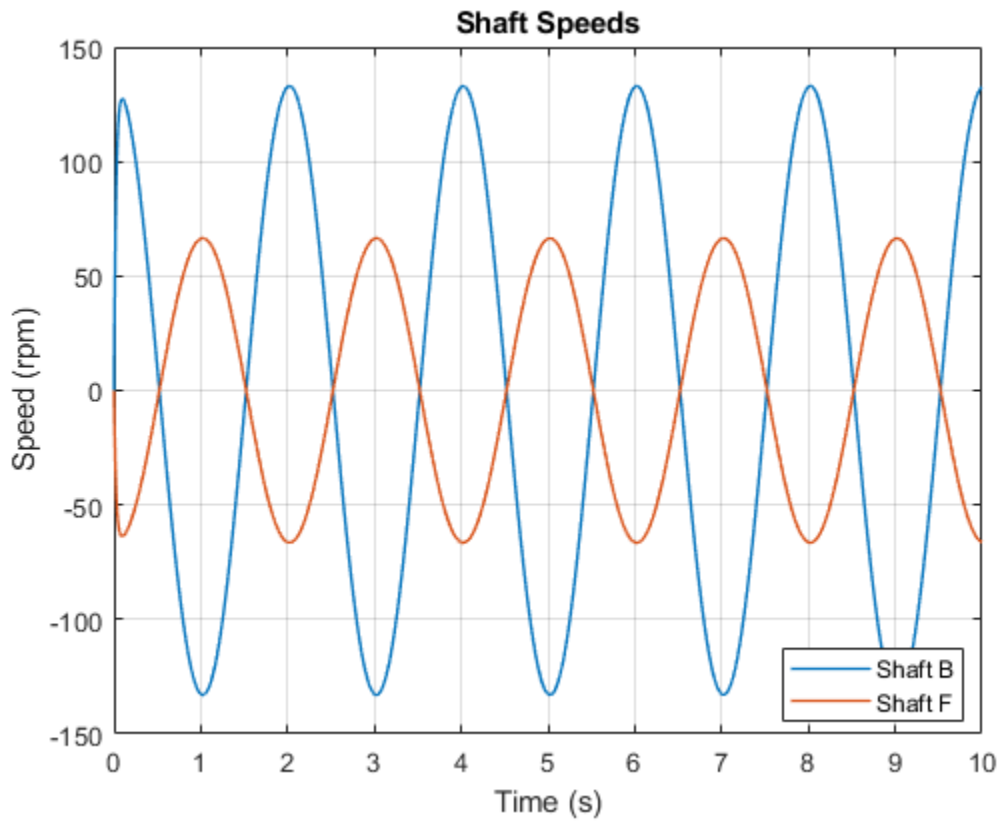
The Common Gear Constraint requires that the rest of the mechanism hold the two frames to which it is connected in alignment. This subsystem has the necessary constraints and parameterized Rigid Transform to hold the frames in the right position and orientation.

Open Subsystem

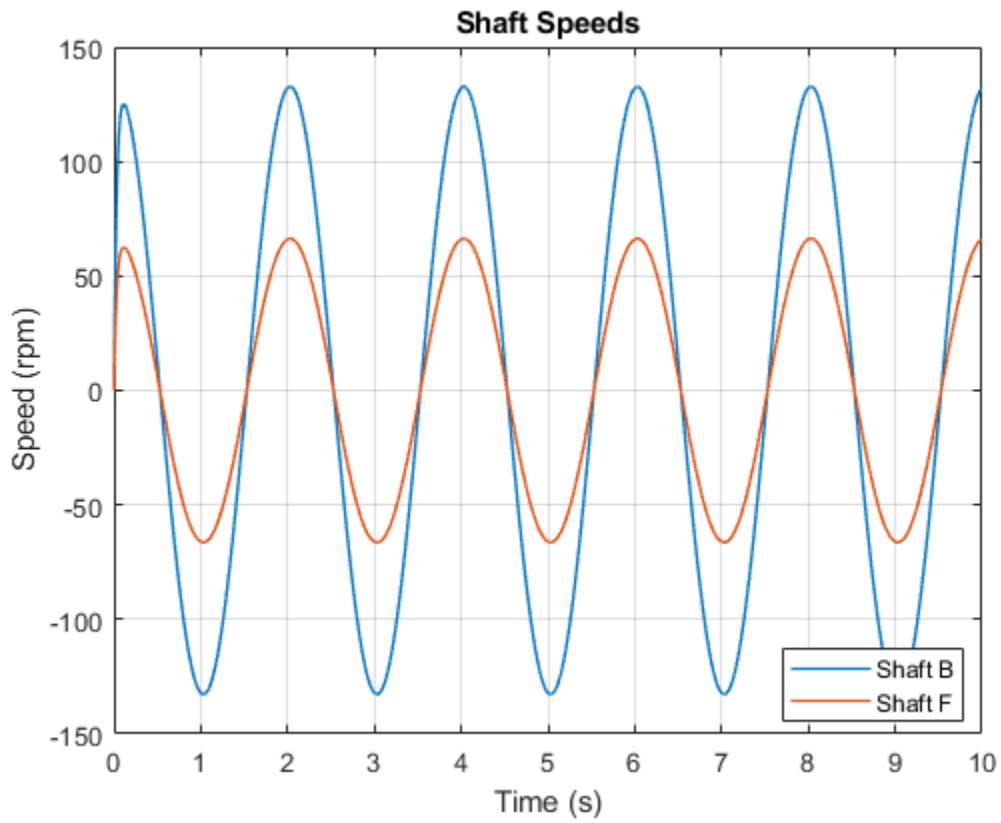


Simulation Results from Simscape Logging

The plot below shows the speeds of the two shafts connected by the common gear assembly in an external configuration.



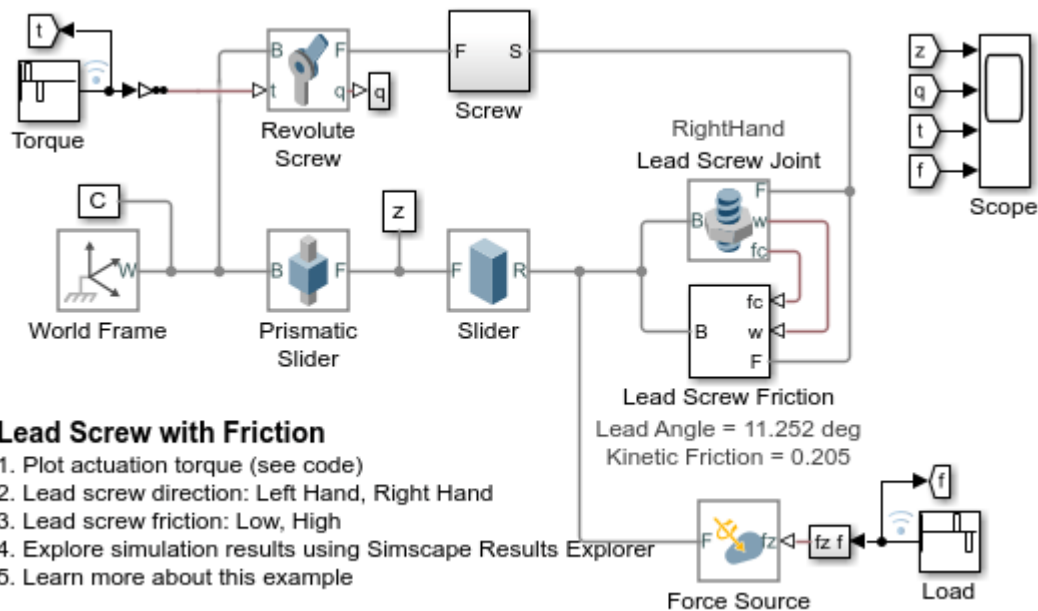
The plot below shows the speeds of the two shafts connected by the common gear assembly in an internal configuration.

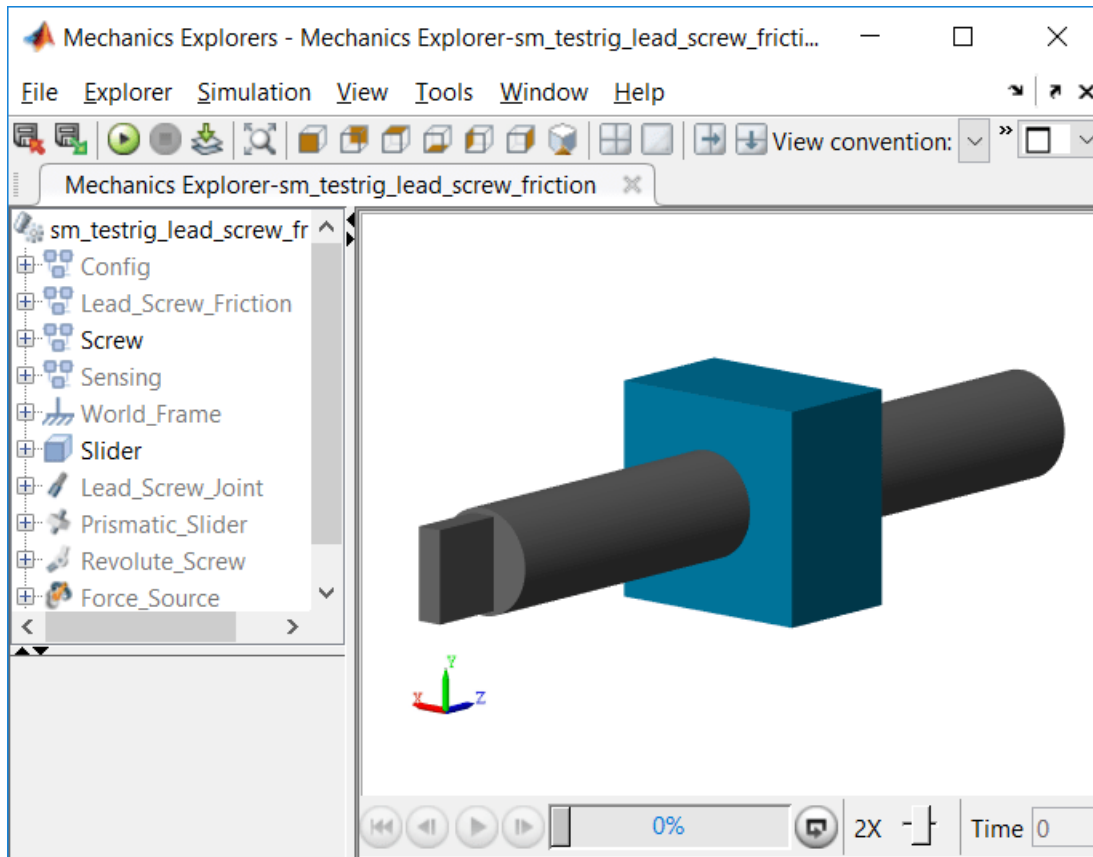


Lead Screw with Friction

This example models a lead screw with friction. The constraint force in the lead screw is measured and used to calculate the friction torque within the lead screw. A continuous stick-slip friction model is used to determine the coefficient of friction based on the relative rotational speed of the two parts connected by the lead screw.

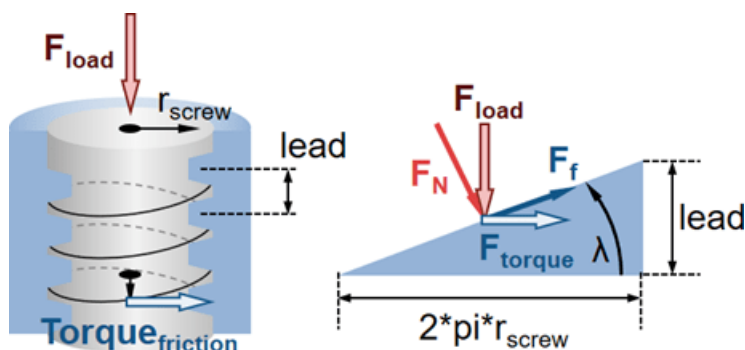
Model





Lead Screw Friction Subsystem

This subsystem calculates and applies the friction torque to the two parts connected by the lead screw joint. The following free-body diagram shows the relevant parameters and forces acting on the system.

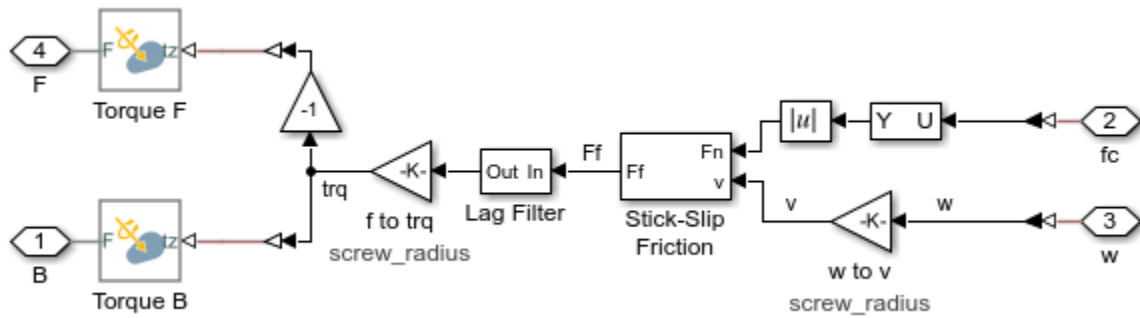


The friction equation is:

$$Torque_{friction} = F_{load} \cdot r_{screw} \cdot \mu$$

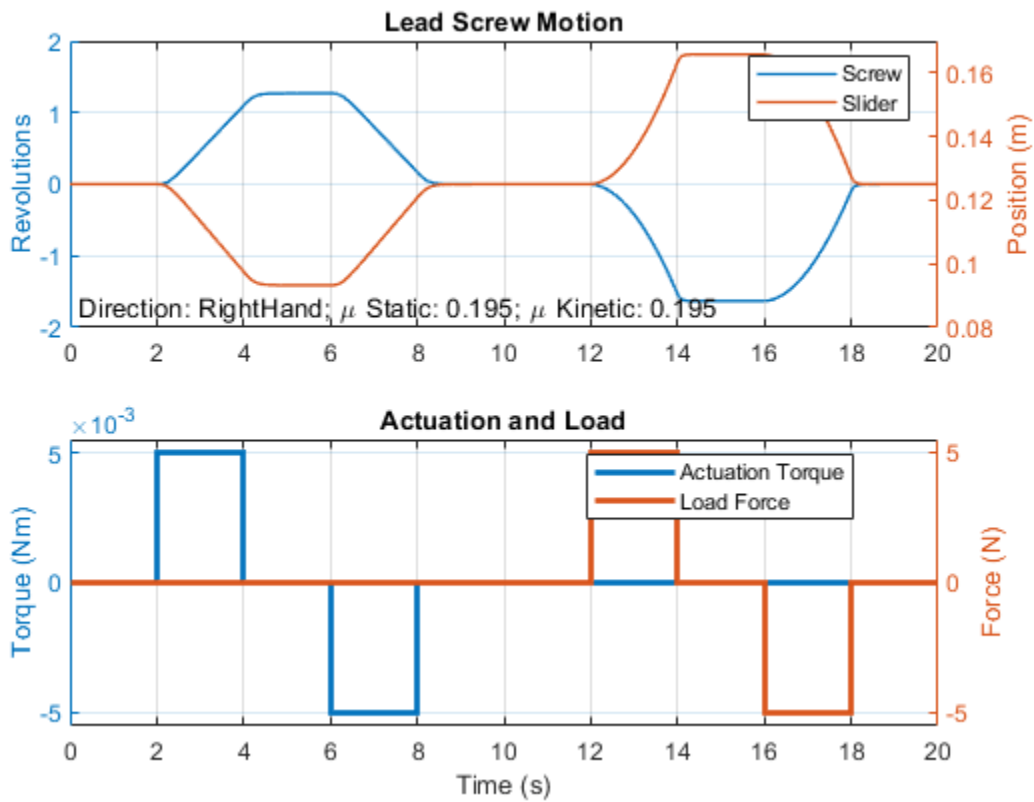
If $\mu > \tan(\lambda)$, the lead screw is non-backdriveable. Applying an axial load force will not be sufficient to permit the lead screw to move.

Open Subsystem

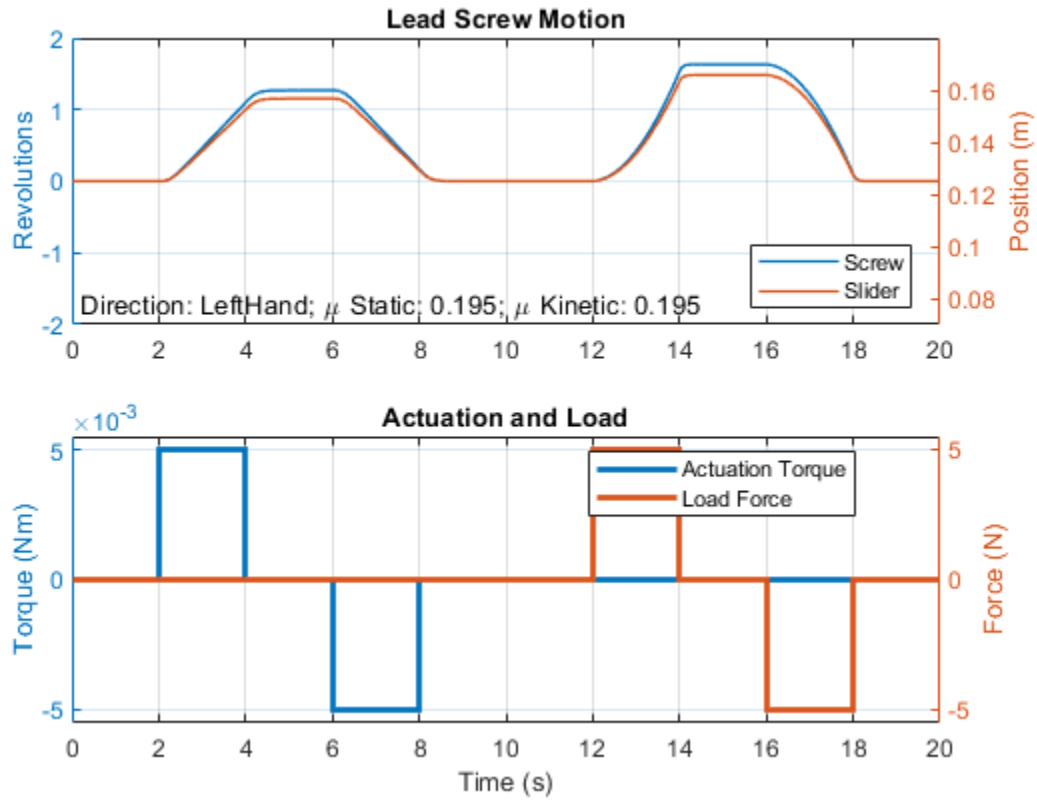


Simulation Results from Simscape Logging

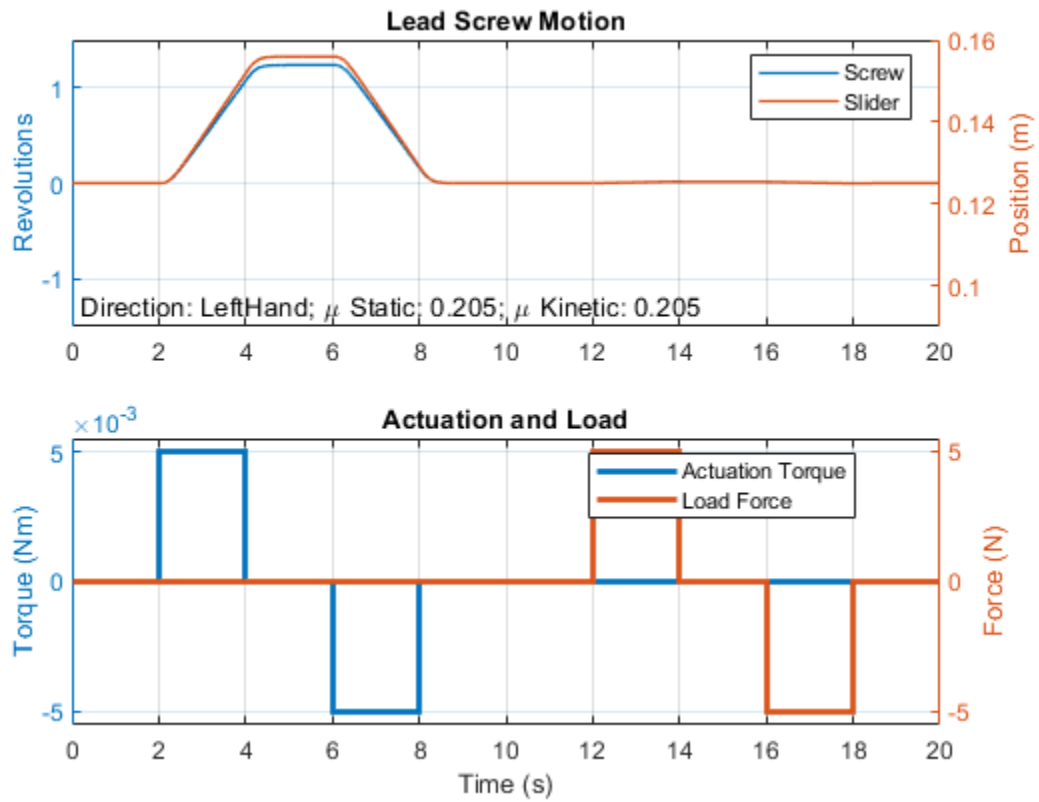
The plot below shows the actuation torque of the lead screw. In this test, the coefficient of friction is low enough that the load force can backdrive the lead screw.



The Lead Screw Joint can be configured such that positive rotation leads to positive translation.



Increasing the coefficient of friction higher than the tangent of the lead angle will make the lead screw non-backdriveable. Applying an axial load force will not be sufficient for the screw to move.



See Also

Prismatic Joint | Revolute Joint | Lead Screw Joint | External Force and Torque

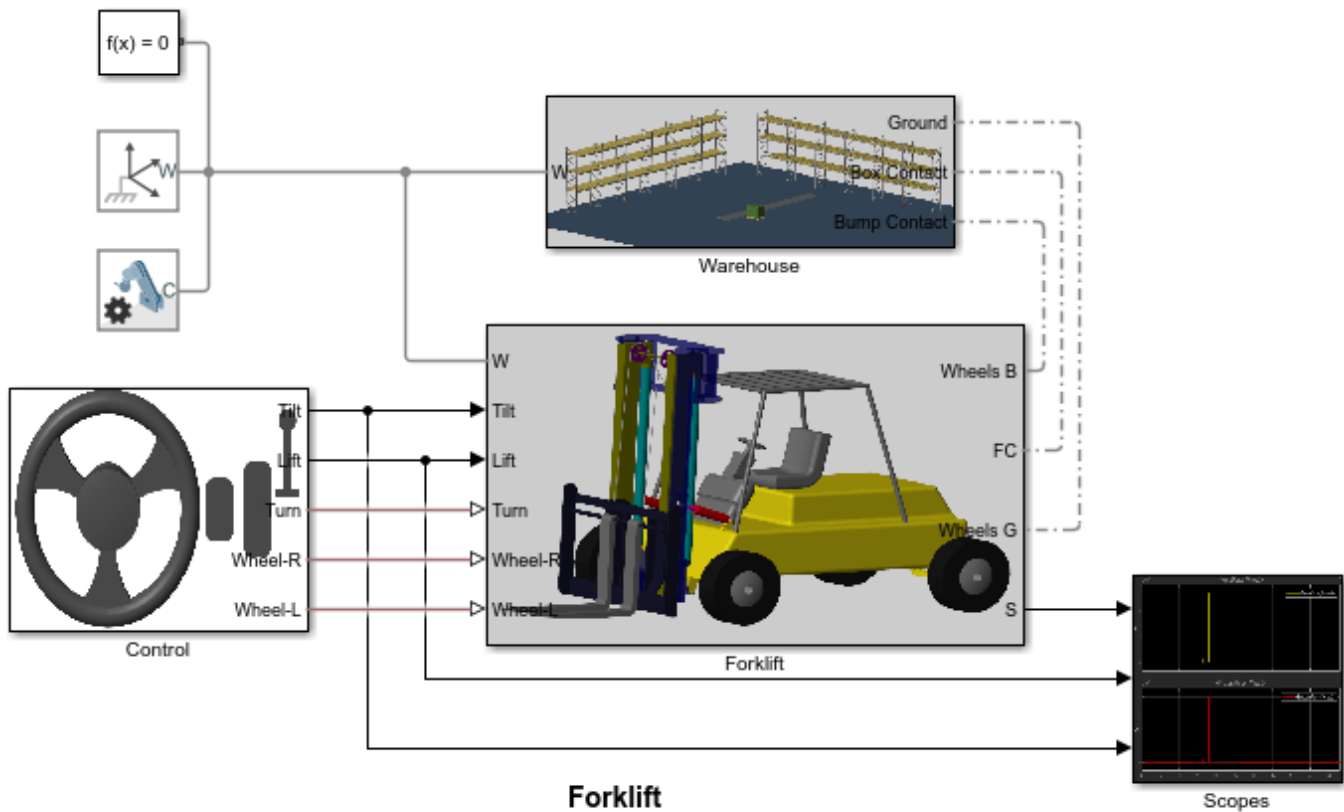
More About

- "Actuating and Sensing with Physical Signals" on page 3-28

Forklift

This example models a forklift which uses the hydraulic and pulley mechanisms to perform the lift action. The tilting of masts is also controlled by hydraulic cylinders. The forklift comprises of 3 masts, namely main mast, top mast and fork mast. The main mast is connected to the chassis by revolute joints and its tilting is governed by the hydraulic tilt cylinders. The top mast slides over the main mast and its motion is governed by the hydraulic lift cylinders. The fork mast slides on the top mast and hangs through belt-cable circuits which drives the movement of the fork mast. A common warehouse application is shown in this example where the objective of the forklift is to grab a box, pass over a bump and place the box in the racks. Spatial Contact Force blocks are used at all contact locations to model the contact between the bodies. The contact between the ground surface and the wheels are modeled using infinite plane block and the contact between the forks and the box are modeled using the point blocks.

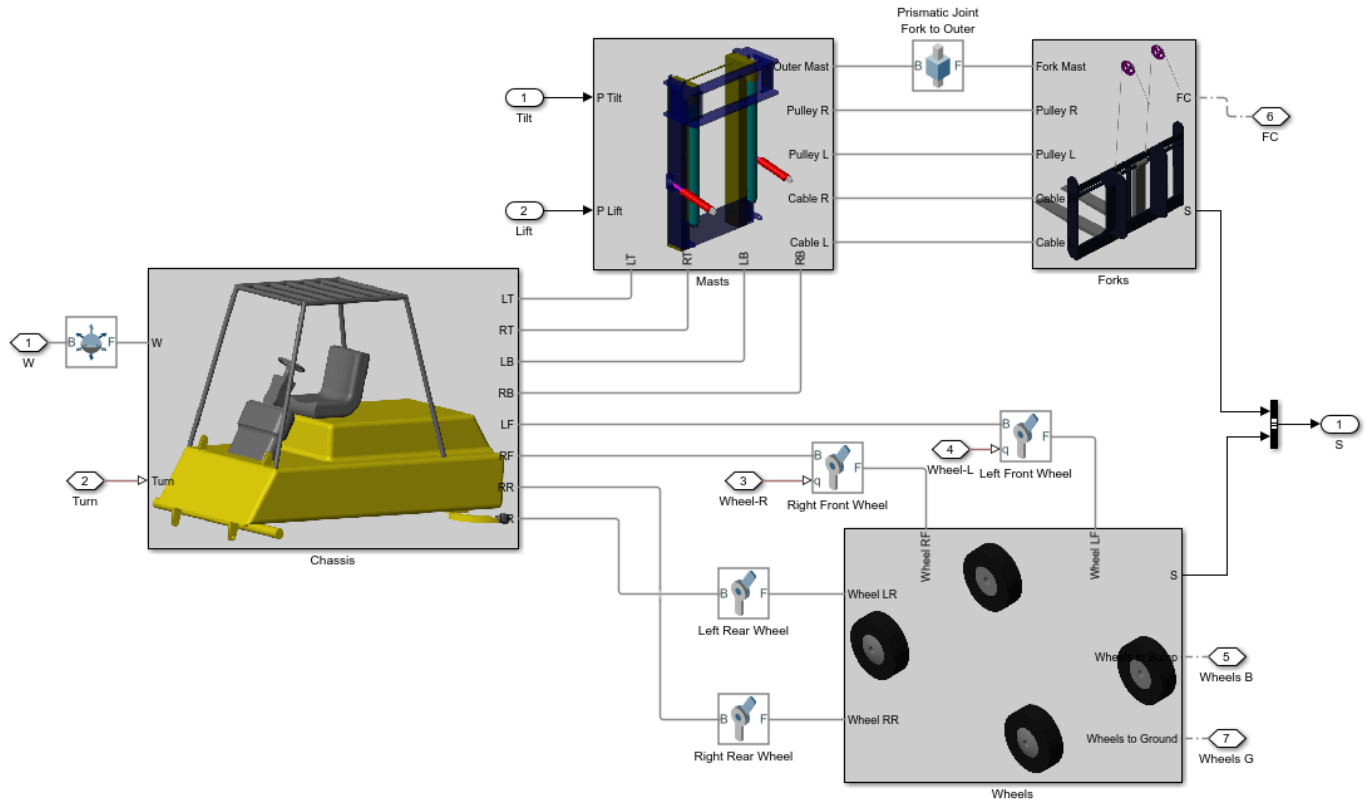
Model



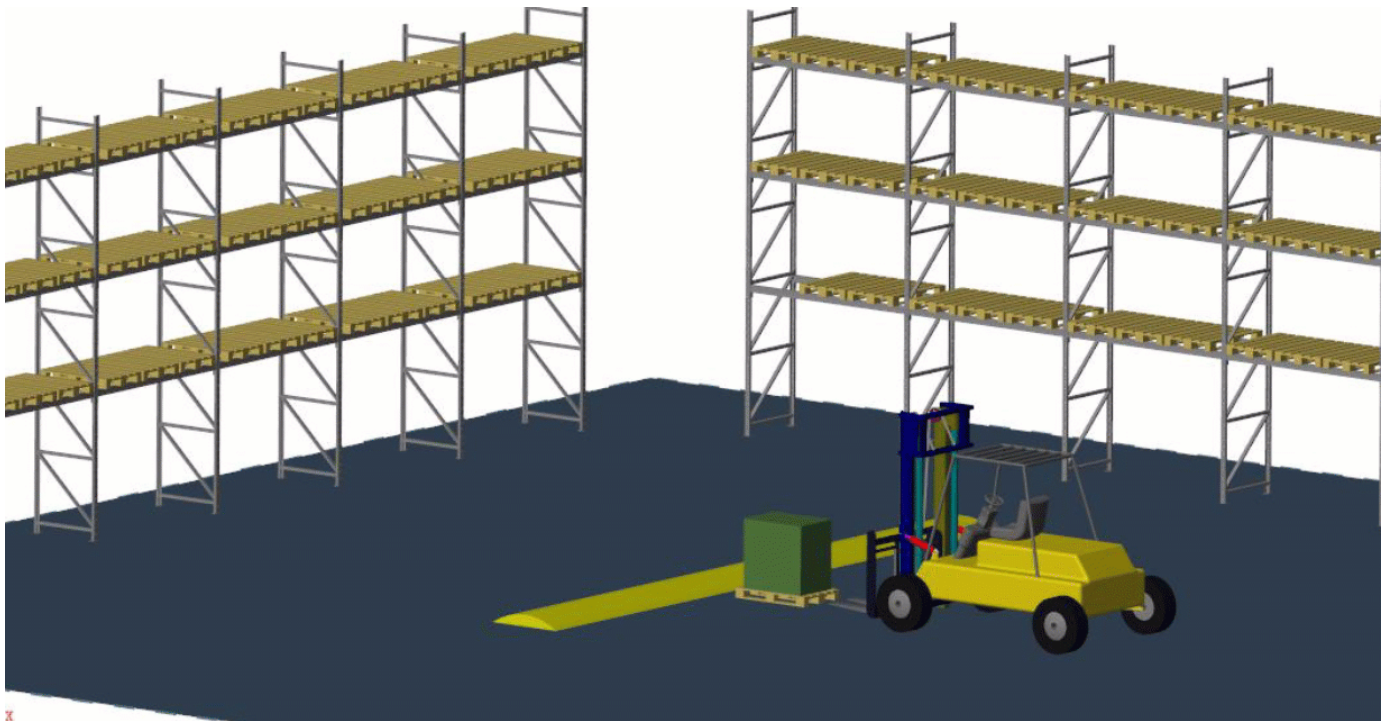
1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Analyze the contact forces](#) at the wheels
3. [Open](#) the subsystem containing the Spatial Contact Force blocks between the forks and the box
4. [Learn more](#) about this example
5. Learn more about the [Spatial Contact Force Block](#)
6. Learn more about [multibody modeling](#)

Assembly of Forklift

Open Subsystem

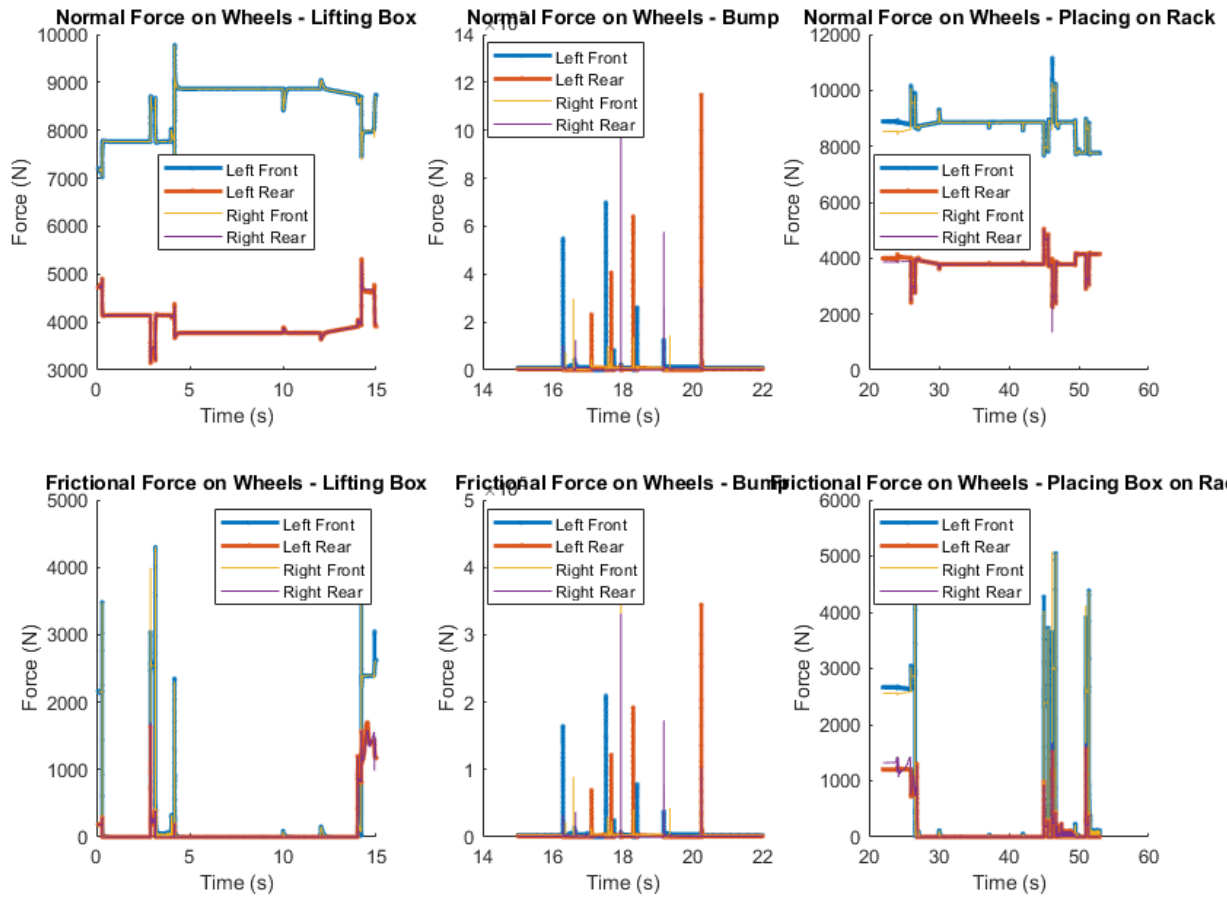


Mechanics Explorer Animation



Simulation Results from Simscape Logging

The plot below shows the Normal force and Frictional force at each wheels during the lift, bump and place motion of the forklift.



See Also

Spatial Contact Force

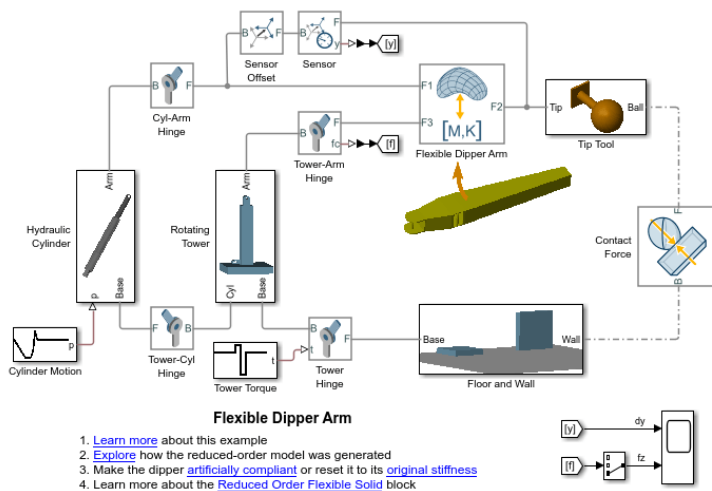
More About

- “Modeling Contact Force Between Two Solids” on page 3-36
- “Model Wheel Contact in a Car” on page 3-49
- “Use Contact Proxies to Simulate Contact” on page 3-40

Flexible Dipper Arm

This example shows how to model a flexible dipper arm, such as the arm for an excavator or a backhoe, by using the Reduced Order Flexible Solid block. This block captures the mechanical behavior of a deformable body through reduced-order stiffness and mass matrices that are associated with interface frame locations. In this example, the Partial Differential Equation Toolbox™ is used to create the reduced-order model.

The dipper is mounted on top of a rotating tower as part of a test rig. The cylinder actuates the arm vertically while the rotating tower allows the arm to swing left and right. A spherical test tool is attached to the tip of the dipper to allow contact with a solid wall. The Scope block allows us to visualize the deformations and reaction forces that result from the motion of the rig.



Copyright 2019-2022 The MathWorks, Inc.

See Also

Reduced Order Flexible Solid | Spatial Contact Force

More About

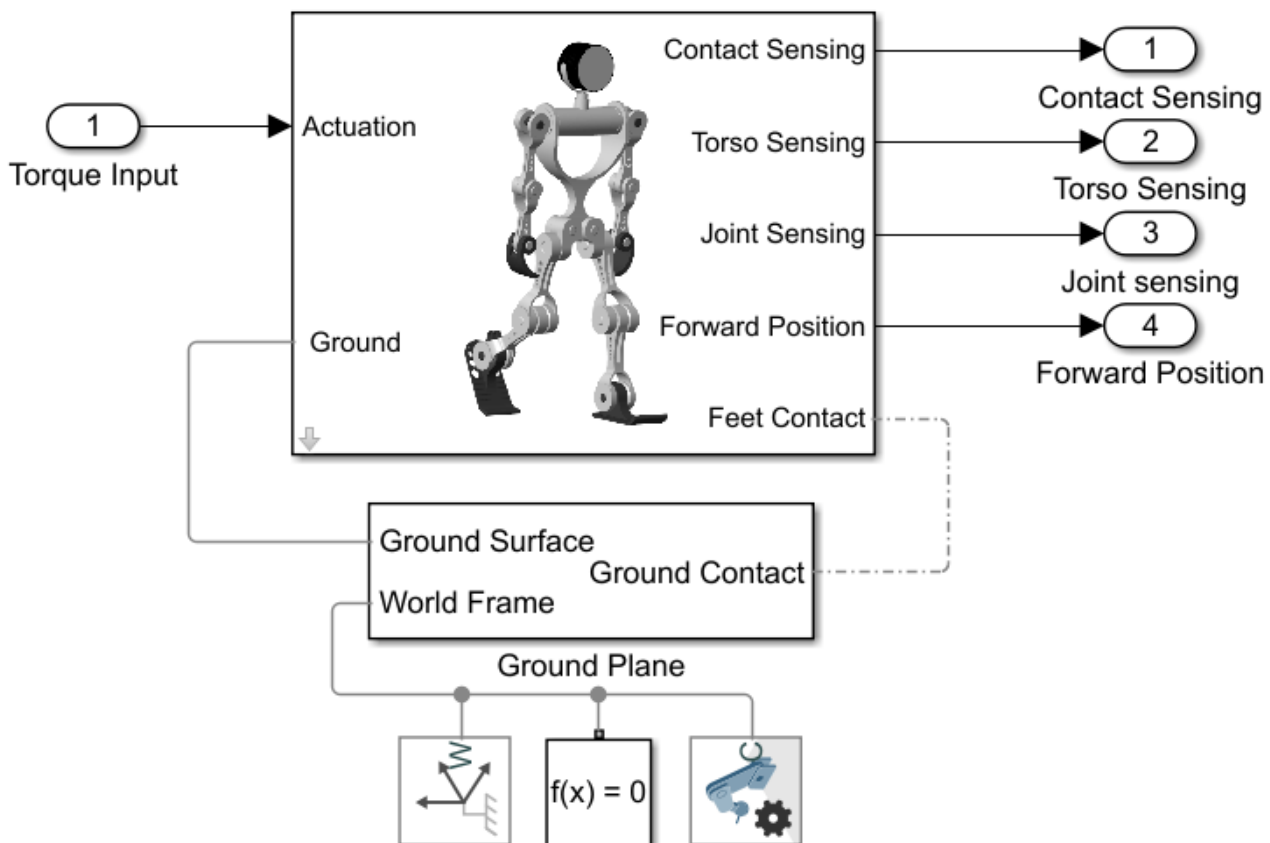
- “Model an Excavator Dipper Arm as a Flexible Body” on page 1-51

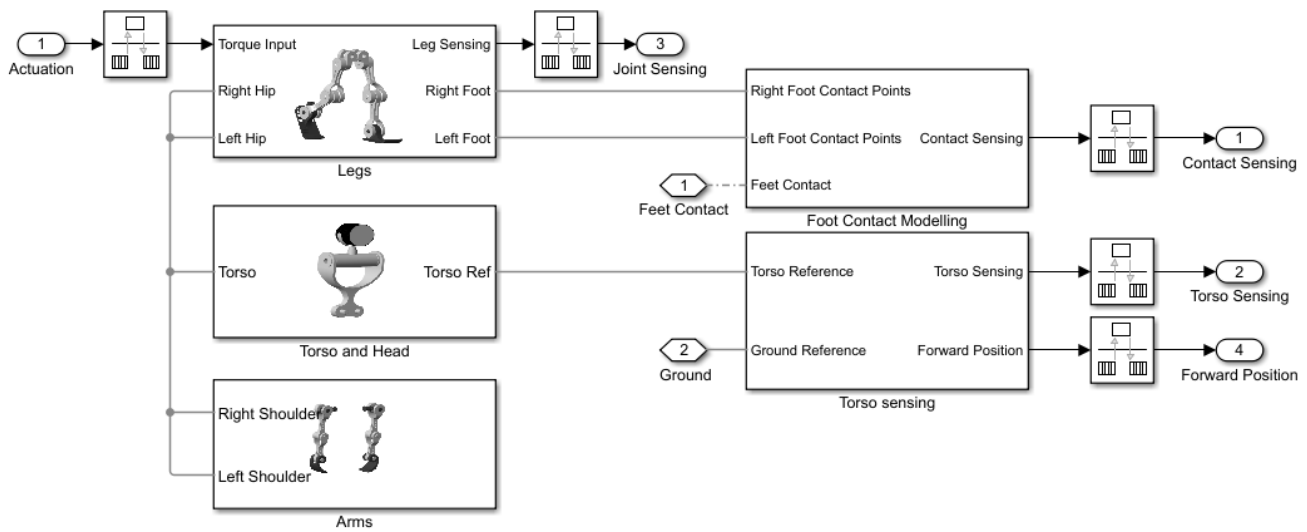
Train Humanoid Walker

This example shows how to model a humanoid robot using “Simscape Multibody”™ and train it using either a genetic algorithm (which requires a “Global Optimization Toolbox” license) or reinforcement learning (which requires “Deep Learning Toolbox”™ and “Reinforcement Learning Toolbox”™ licenses).

Humanoid Walker Model

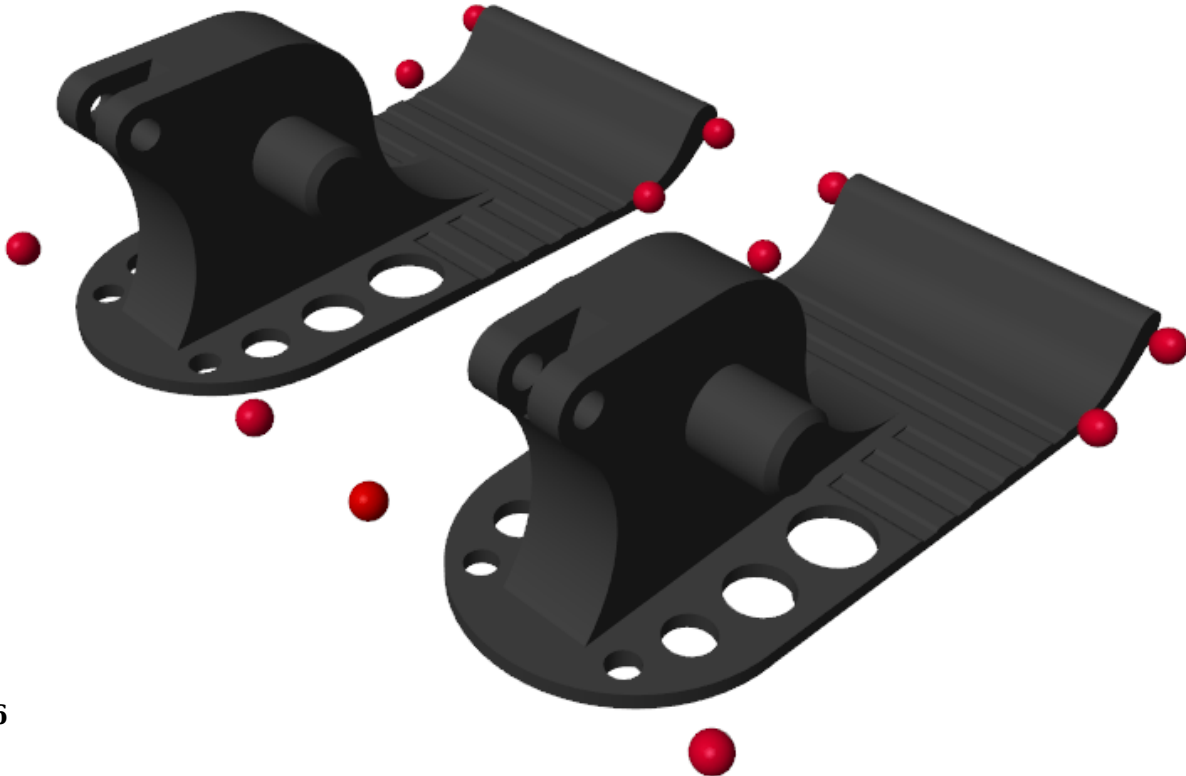
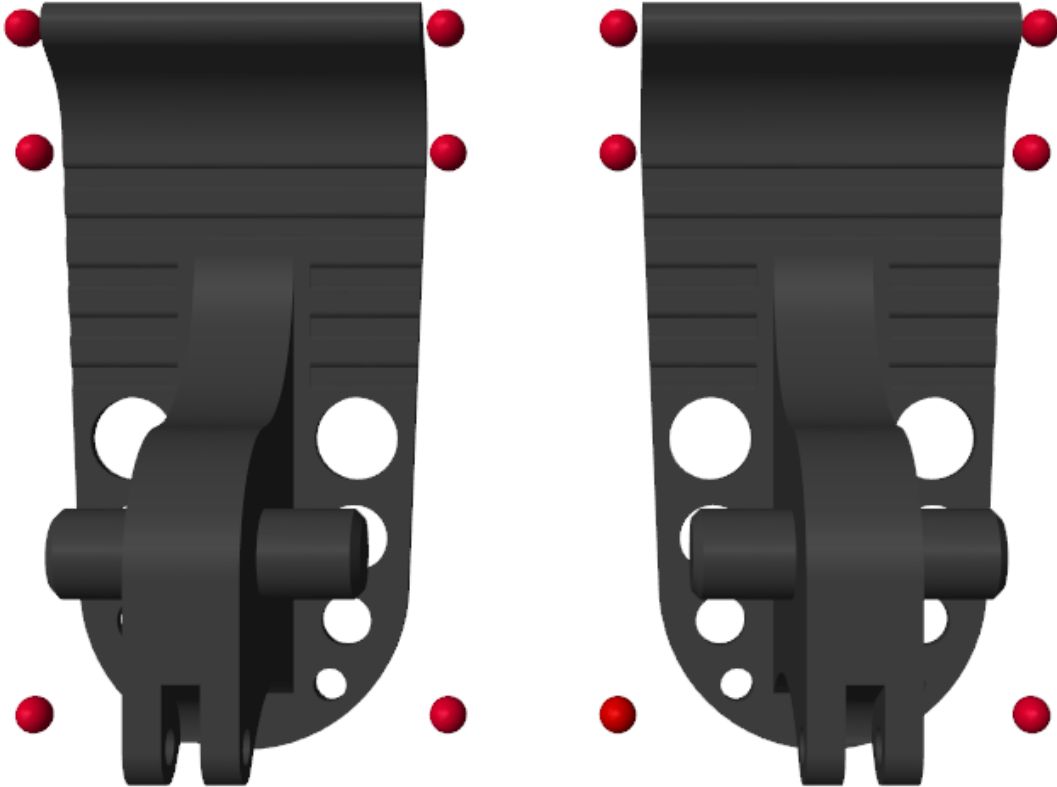
This example is based on a humanoid robot model. You can open the model by entering `sm_import_humanoid_urdf` in the MATLAB® command prompt. Each leg of the robot has torque-actuated revolute joints in the frontal hip, knee, and ankle. Each arm has two passive revolute joints in the frontal and sagittal shoulder. During the simulation, the model senses the contact forces, position and orientation of the torso, joint states, and forward position. The figure shows the Simscape Multibody model on different levels.





Contact Modeling

The model uses Spatial Contact Force blocks to simulate the contact between the feet and ground. To simplify the contact and speed up the simulation, red spheres are used to represent the bottoms of the robotic feet. For more details, see “Use Contact Proxies to Simulate Contact” on page 3-40.

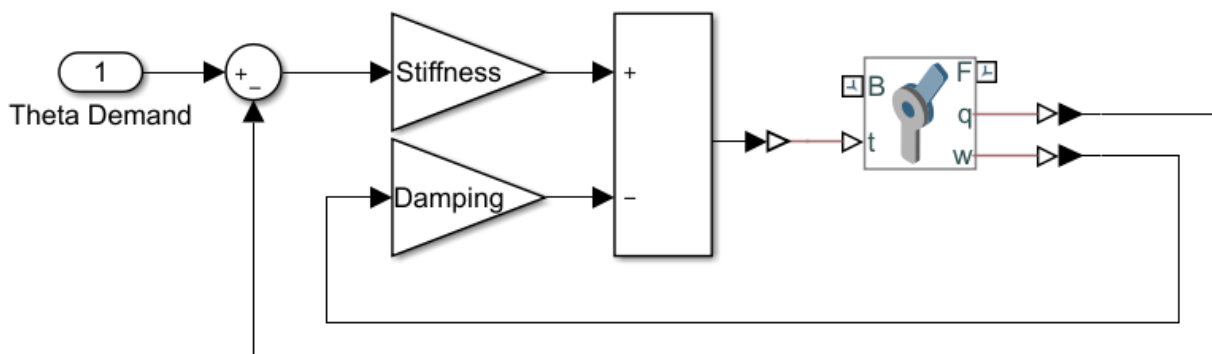


Joint Controller

The model uses a stiffness-based feedback controller to control each joint [1]. Model the joints as first-order systems with an associated stiffness (K) and damping (B), which you can set to make the joint behavior critically damped. The torque is applied when the setpoint θ_0 differs from the current joint position θ :

$$T = B\dot{\theta} + K(\theta_0 - \theta).$$

You can vary the spring set-point θ_0 to elicit a feedback response to move the joint. The figure shows the Simulink model of the controller.



Humanoid Walker Training

The goal of this example is to train a humanoid robot to walk, and you can use various methods to train the robot. The example shows the genetic algorithm and reinforcement learning methods.

The Walking Objective Function

This example uses an objective function to evaluate different walking styles. The model gives a reward (r_t) at each timestep:

$$r_t = w_1 v_y + w_2 t_s - w_3 p - w_4 \Delta z - w_5 \Delta x$$

Here:

- v_y — Forward velocity (rewarded)
- p — Power consumption (penalized)
- Δz — Vertical displacement (penalized)
- Δx — Lateral displacement (penalized)
- w_1, \dots, w_5 : Weights, which represent the relative importance of each term in the reward function

Additionally, not falling over is rewarded.

Consequently, the total reward (R) for a walking trial is:

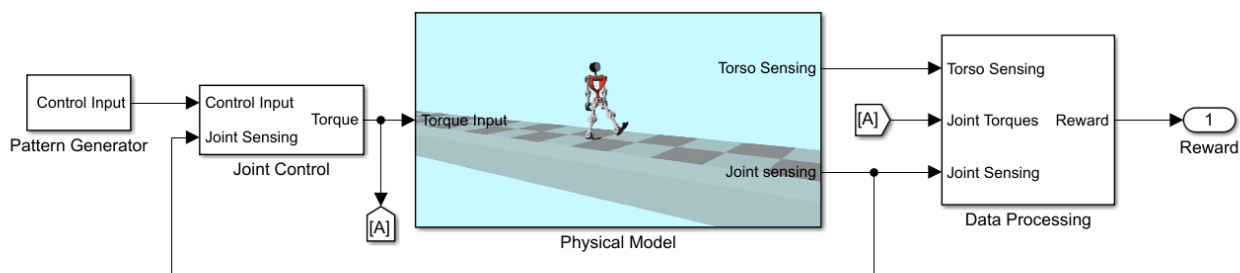
$$R = \sum_{t=0}^T r_t$$

Here T is the time at which the simulation terminates. You can change the reward weights in the `sm_humanoid_walker_rl_parameters` script. The simulation terminates when the simulation time is reached or the robot falls. Falling is defined as:

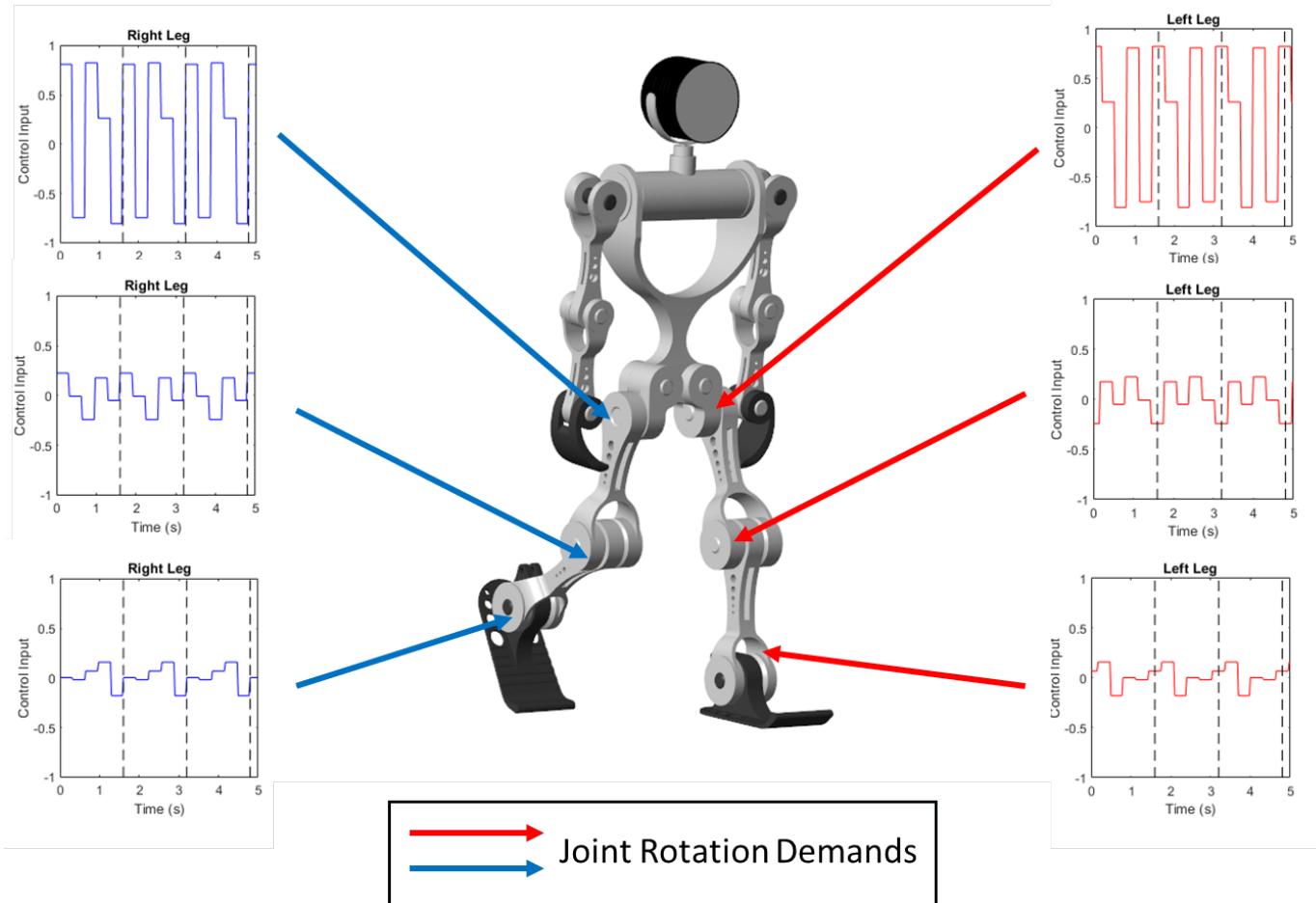
- The robot drops below 0.5 m.
- The robot moves laterally by more than 1 m.
- The robot torso rotates by more than 30 degrees.

Train with Genetic Algorithm

To optimize the walking of the robot, you can use a genetic algorithm. A genetic algorithm solves optimization problems based on a natural selection process that mimics biological evolution. Genetic algorithms are especially suited to problems when the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. For more information, see `ga` (Global Optimization Toolbox).



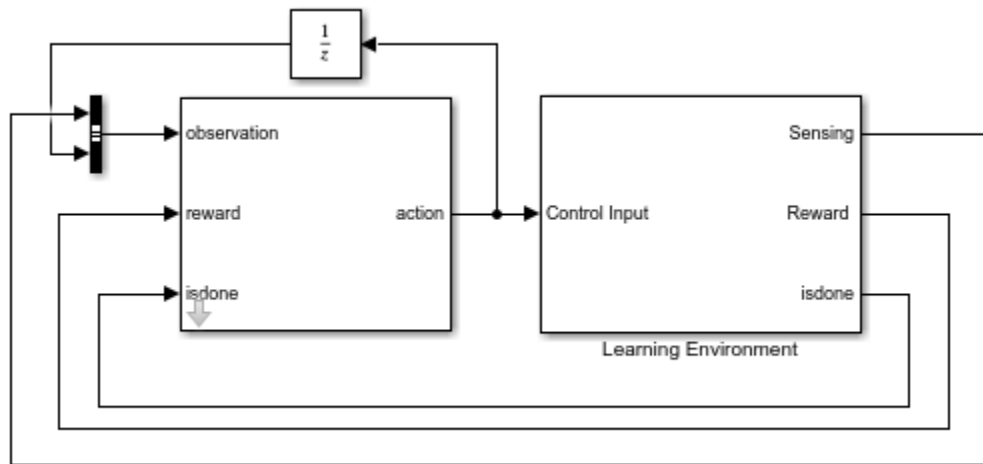
The model sets the angular demand for each joint to a repeating pattern that is analogous to the central pattern generators seen in nature [2]. The repeating pattern yields an open-loop controller. The periodicity of the signals is the gait period, which is the time taken to complete one full step. During each gait period, the signal switches between different angular demand values. Ideally, the humanoid robot walks symmetrically, and the control pattern for each joint in the right leg is transmitted to the corresponding joint in the left leg, with a delay of half a gait period. The pattern generator aims to determine the optimal control pattern for each joint and to maximize the walking objective function.



To train the robot with a genetic algorithm, open the `sm_humanoid_walker_ga_train` script. By default, this example uses a pretrained humanoid walker. To train the humanoid walker, set `trainWalker` to `true`.

Train with Reinforcement Learning

Alternatively, you can also train the robot using a deep deterministic policy gradient (DDPG) reinforcement learning agent. A DDPG agent is an actor-critic reinforcement learning agent that computes an optimal policy that maximizes the long-term reward. DDPG agents can be used in systems with continuous actions and states. For details about DDPG agents, see `rLDDPGAgent` (Reinforcement Learning Toolbox).



To train the robot with reinforcement learning, open the `sm_humanoid_walker_rl_train` script. By default, this example uses a pretrained humanoid walker. To train the humanoid walker, set `trainWalker` to `true`.

References

- [1] Kalveram, Karl T., Thomas Schinauer, Steffen Beirle, Stefanie Richter, and Petra Jansen-Osmann. "Threading Neural Feedforward into a Mechanical Spring: How Biology Exploits Physics in Limb Control." *Biological Cybernetics* 92, no. 4 (April 2005): 229-40. <https://doi.org/10.1007/s00422-005-0542-6>.
- [2] Jiang Shan, Cheng Junshi, and Chen Jiapin. "Design of Central Pattern Generator for Humanoid Robot Walking Based on Multi-Objective GA." In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)* (Cat. No.00CH37113), 3: 1930-35. Takamatsu, Japan: IEEE, 2000. <https://doi.org/10.1109/IROS.2000.895253>.

See Also

Point | Point Cloud | Spatial Contact Force

More About

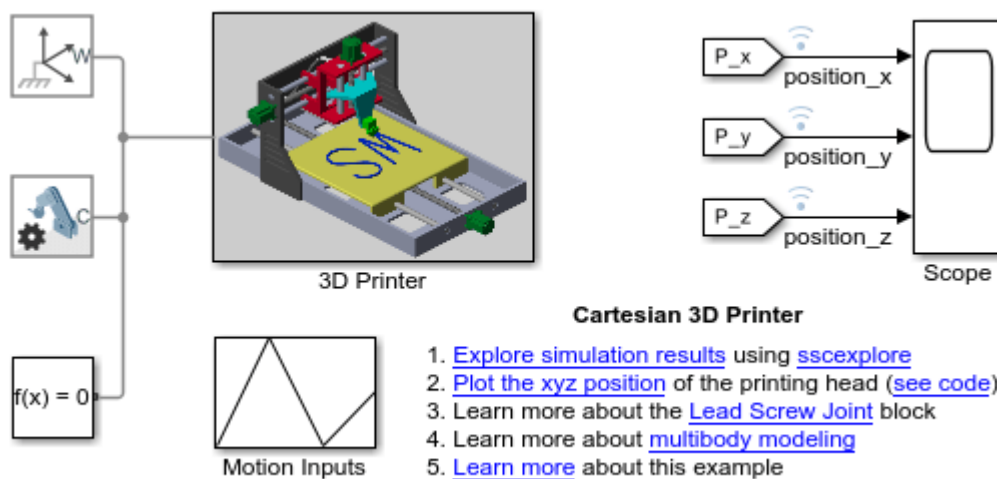
- "Deep Learning Toolbox"
- "Global Optimization Toolbox"
- "Humanoid Robot" on page 8-63
- "Import a URDF Humanoid Model" on page 6-40
- "Modeling Contact Force Between Two Solids" on page 3-36
- "Reinforcement Learning Toolbox"
- "Use Contact Proxies to Simulate Contact" on page 3-40

Cartesian 3D Printer

This example models a Cartesian 3D printer. The model allows you to specify the rotational motion of the motor on each axis to define a printing path. In this example, the printing head moves along the edges of two letters, S and M, using the predefined rotational motions.

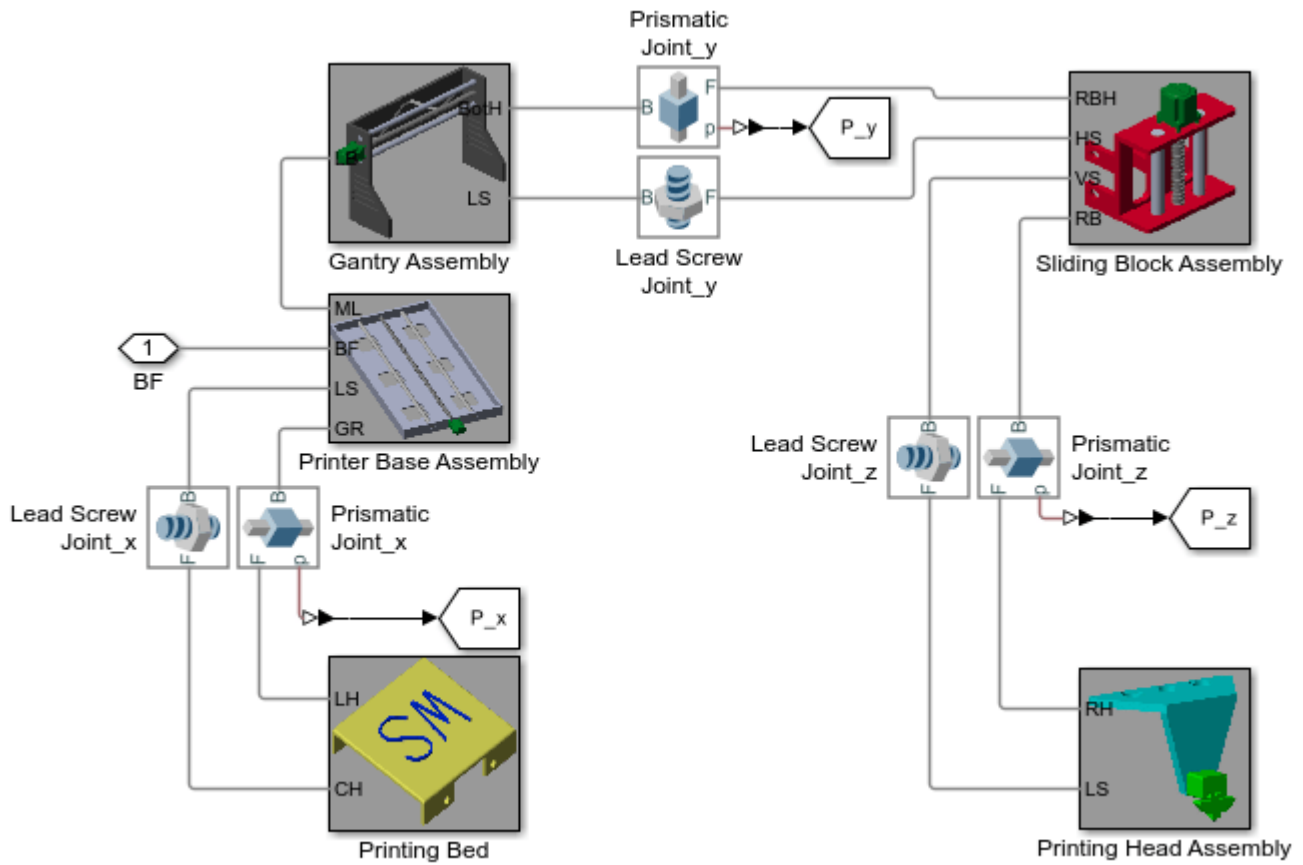
The printer has three linear actuators that drive the motion of the printing head along x -, y -, and z -directions. These actuators are modeled by the Lead Screw Joint blocks. The lead screw joints convert the rotational inputs to translational motions.

Model

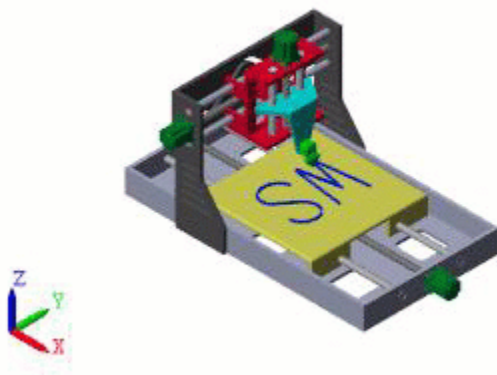


Subsystems

The 3D printer has three degrees of freedom and consists of five subsystems: the printing bed, printer base assembly, gantry assembly, sliding block assembly, and printing head assembly. The prismatic and lead screw joints constrain the subsystems and permit movements along x -, y -, and z -axes.

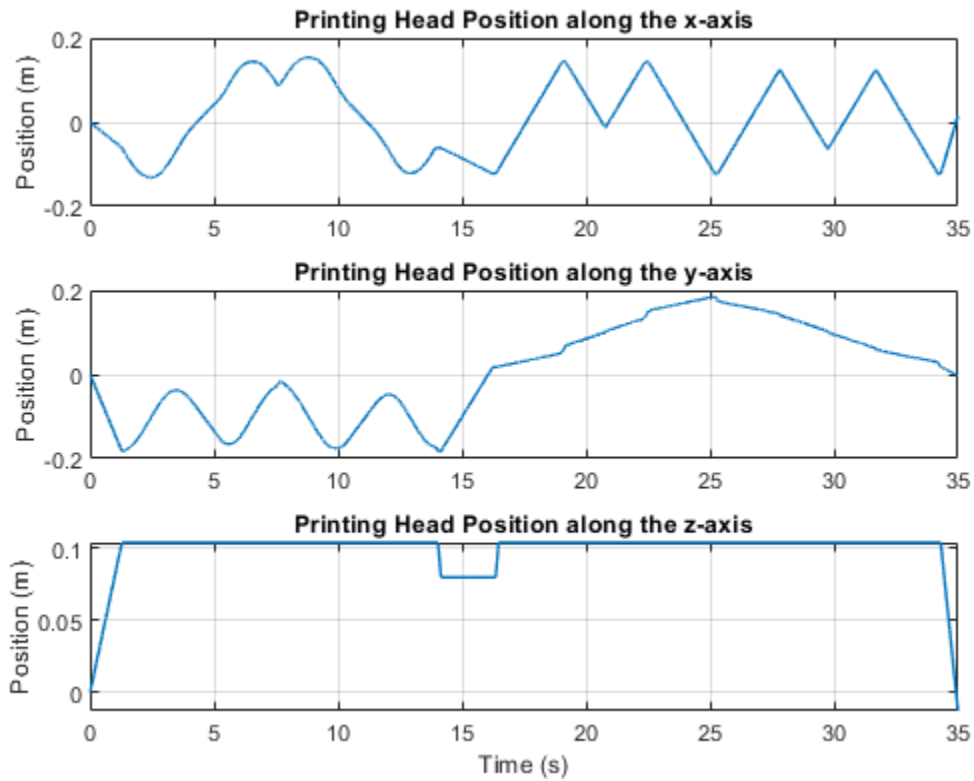


Mechanics Explorer Animation



Simulation Results

This plot shows the x -, y -, and z -positions of the printing head during the printing.



See Also

Lead Screw Joint | Revolute Joint

More About

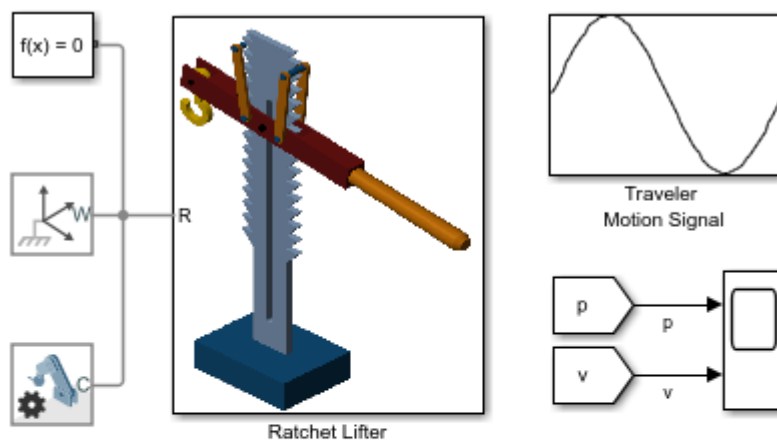
- “Joint Actuation Limitations” on page 3-26
- “Motion Sensing” on page 3-56
- “Specifying Joint Actuation Inputs” on page 3-19

Ratchet Lifter

This example models a ratchet lifter and demonstrates how to use contact proxies for contact problems that involve complex geometries.

In this example, as the traveler handle is pumped up and down, the traveler climbs up along the toothed rack to lift a load. To simplify the contacts between the toothed rack and climbers, the model represents the rack teeth and climber shaft-pins with proxies, such as cylinders and spheres, and simulates the contacts between these proxies.

Model

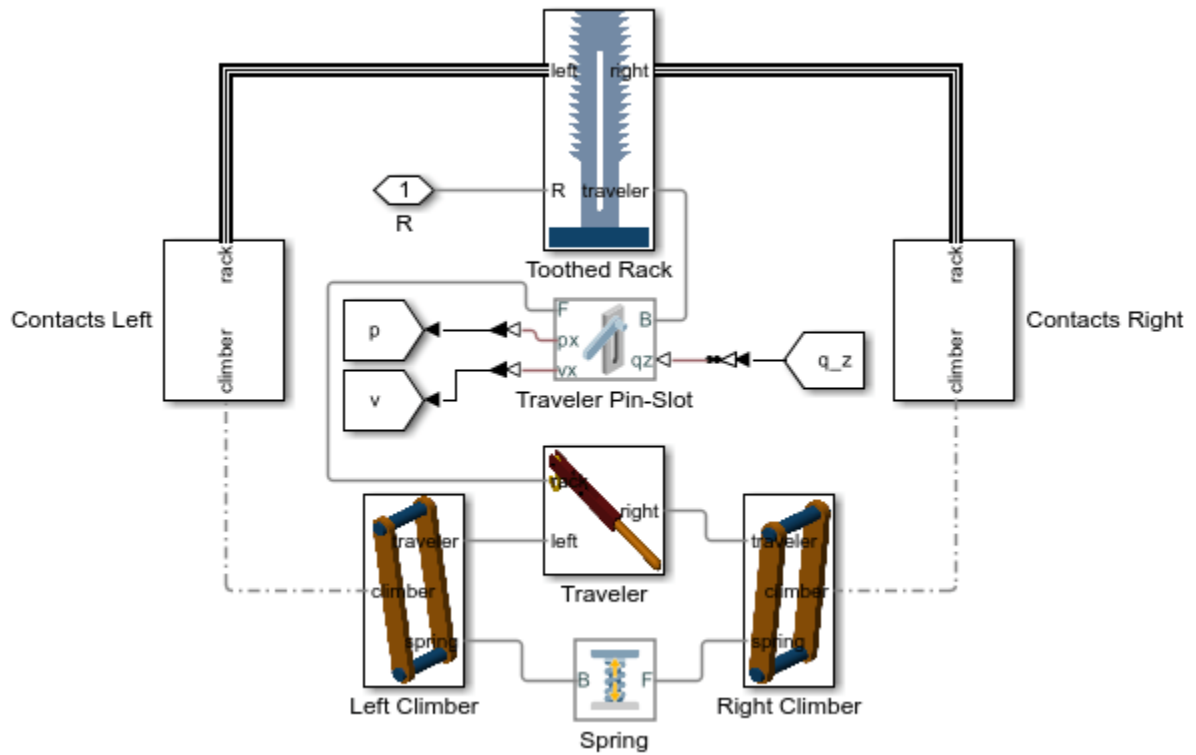


Ratchet Lifter

1. [Explore simulation results](#) using [sscexplore](#)
2. [Plot the position and velocity](#) of the traveler in the vertical direction ([see code](#))
3. Learn more about the [Spatial Contact Force Block](#)
4. Learn more about [multibody modeling](#)
5. [Learn more](#) about this example

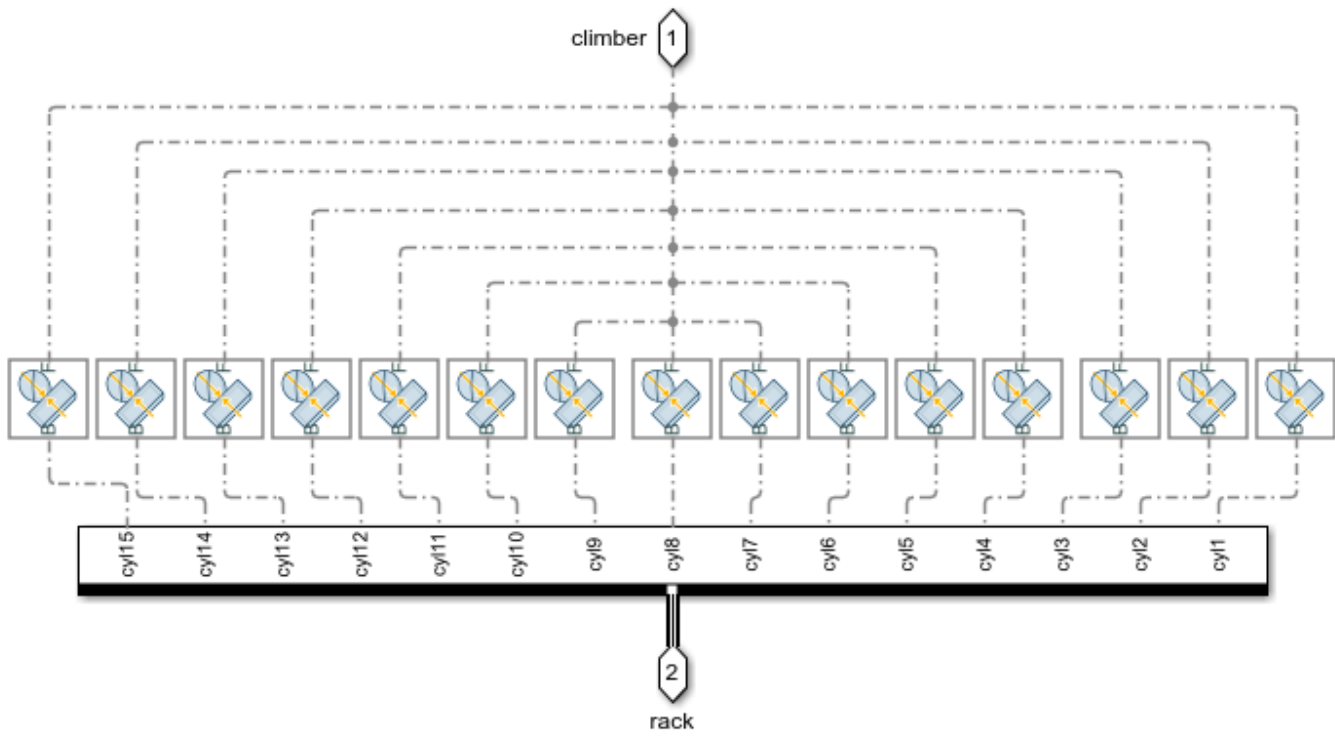
Subsystems

The ratchet lifter consists of five subsystems: the Toothed Rack, Traveler, Neg Climber, Pos Climber, and Contact subsystems. The Pin Slot Joint block connects the toothed rack and traveler. The climbers are connected by the traveler and Spring and Damper Force block.

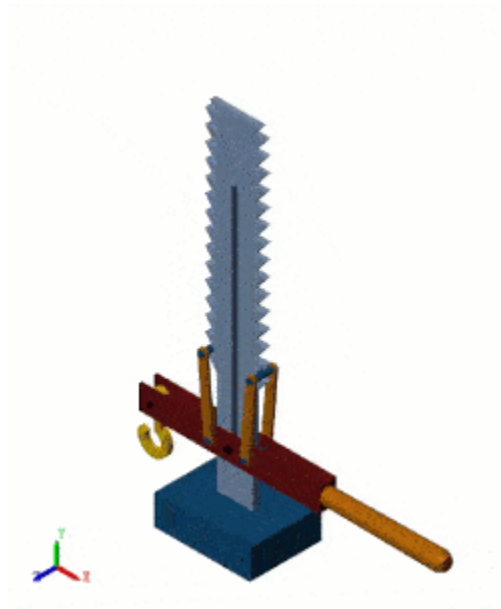


Contact Simulation

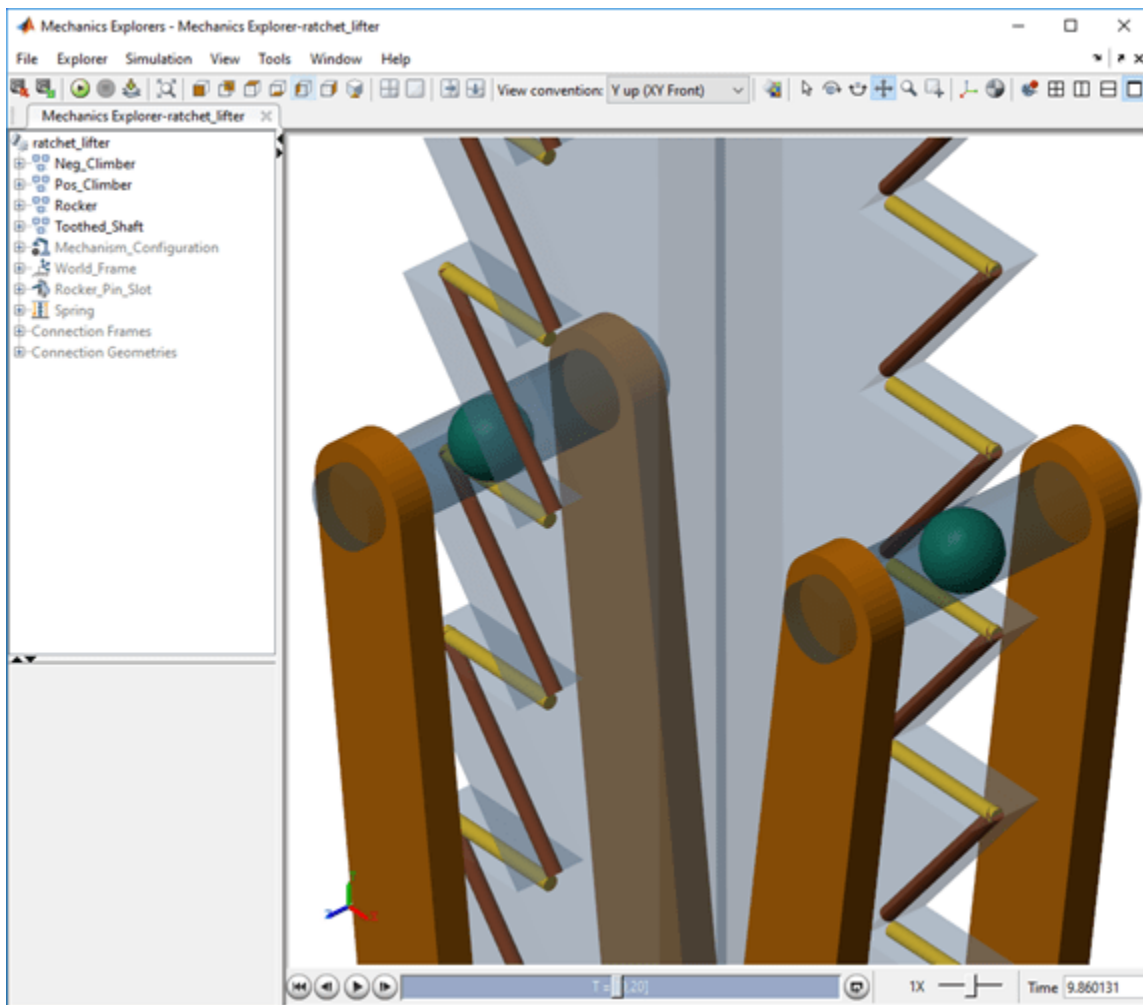
The Spatial Contact Force blocks that are used to model the contact forces between the climbers and rack teeth are housed in the Contact subsystems.



Animation and Proxies

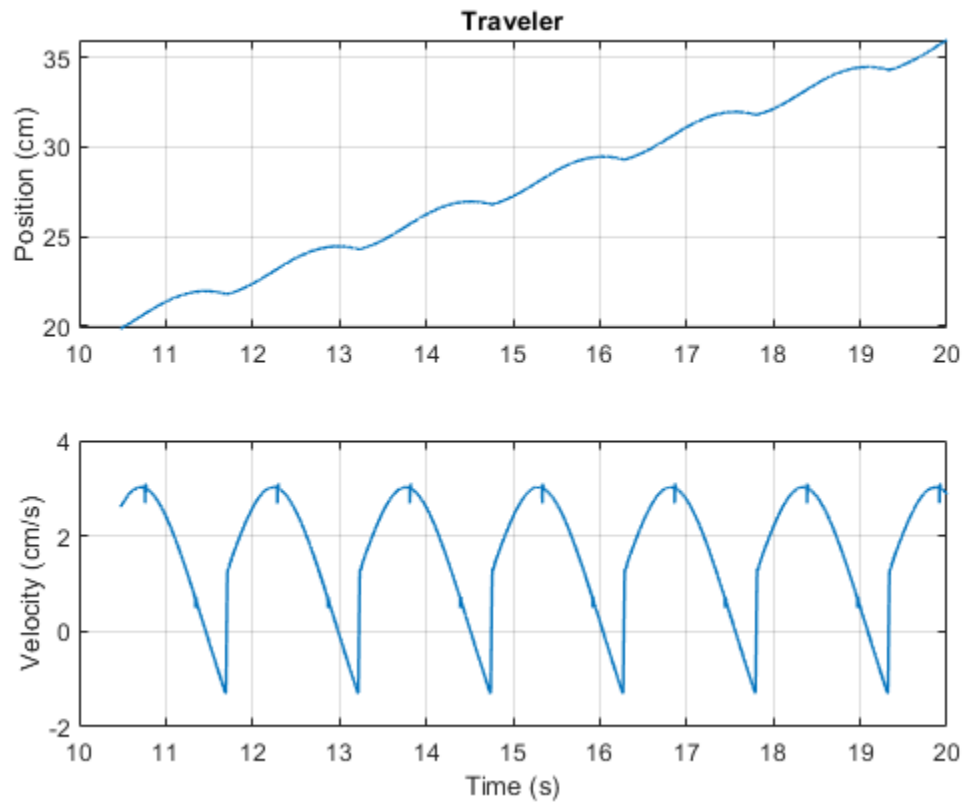


The image shows the proxies of the rack teeth and climber cylinders.



Simulation Results

This plot shows the position and velocity of the middle pin of the traveler.



See Also

Pin Slot Joint | Spatial Contact Force | Spring and Damper Force

More About

- "Modeling Contact Force Between Two Solids" on page 3-36
- "Model Wheel Contact in a Car" on page 3-49
- "Train Humanoid Walker" on page 8-114
- "Use Contact Proxies to Simulate Contact" on page 3-40
- "Zero-Crossing Detection"

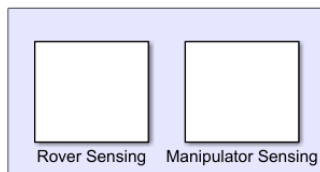
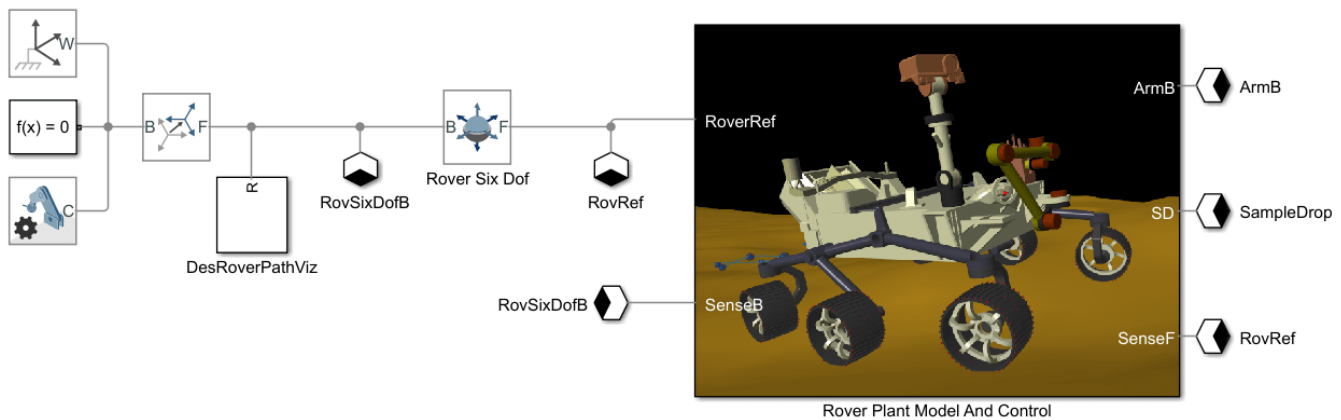
Modeling and Control of a Mars Rover

This example models a mars rover performing a sample retrieval task using “Simscape Multibody”™ and “Robotics System Toolbox”. The rover follows a desired path on a rigid terrain surface, stops at the target location and uses its manipulator to pick and store a sample from the surface. It uses the following key features to model different aspects of the application:

- Grid Surface for modeling the rigid terrain surface. (Requires a “Simscape Multibody”™ license)
- Point Cloud and Spatial Contact Force for modeling the contact between the rover wheels and the rigid terrain. (Requires a “Simscape Multibody”™ license)
- Pure Pursuit (Robotics System Toolbox) for path tracking control of the rover. (Requires a “Robotics System Toolbox” license)
- KinematicsSolver (requires a “Simscape Multibody”™ license) and Trapezoidal Velocity Profile Trajectory (Robotics System Toolbox) (requires a “Robotics System Toolbox” license) for joint space trajectory planning and control of the rover arm.
- Joint Mode Configuration (requires a “Simscape Multibody”™ license) for modeling the interaction between the end effector and the sample.

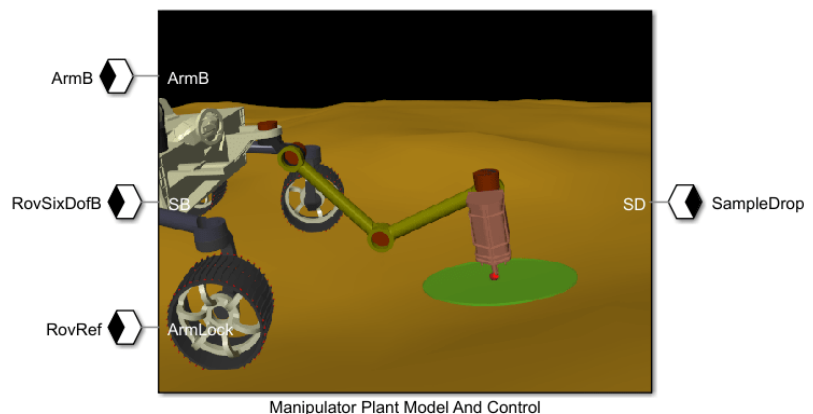
Mars Rover Model

Refer to the model `sm_mars_rover.slx` to view the subsystems mentioned in this example.

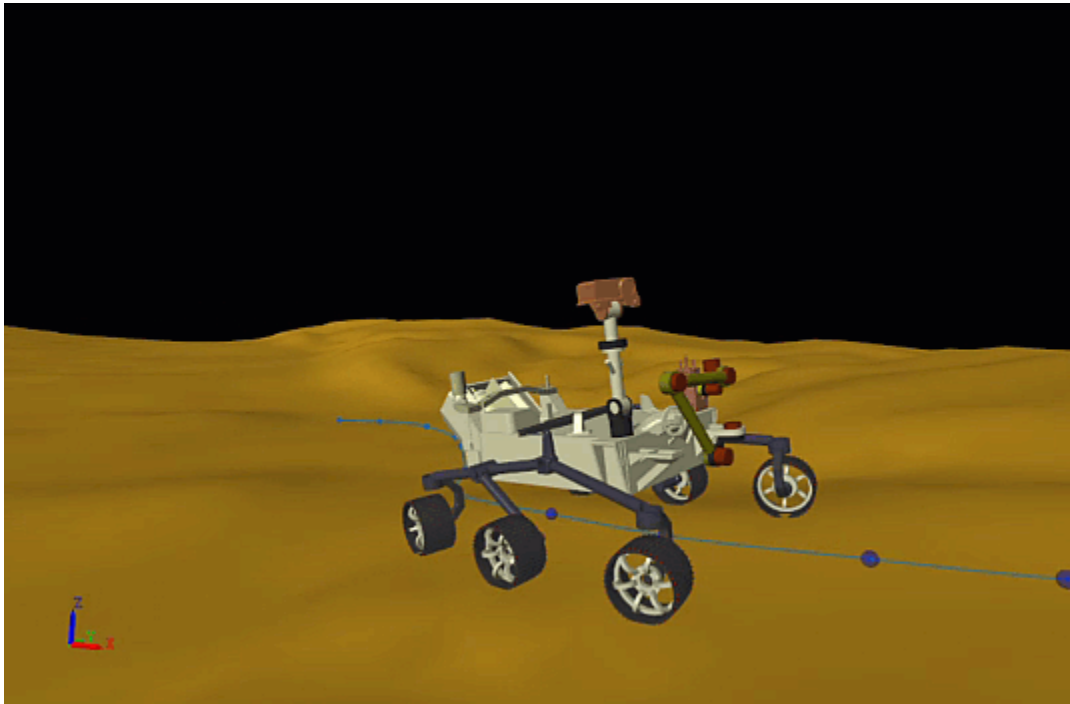


Mars Rover

1. Learn more about the [grid surface](#) block
2. Learn more about [contact modeling](#)
3. Learn more about the [KinematicsSolver class](#)
4. Learn more about [multibody modeling](#)
5. [Explore simulation results](#) using [sscxplore](#)

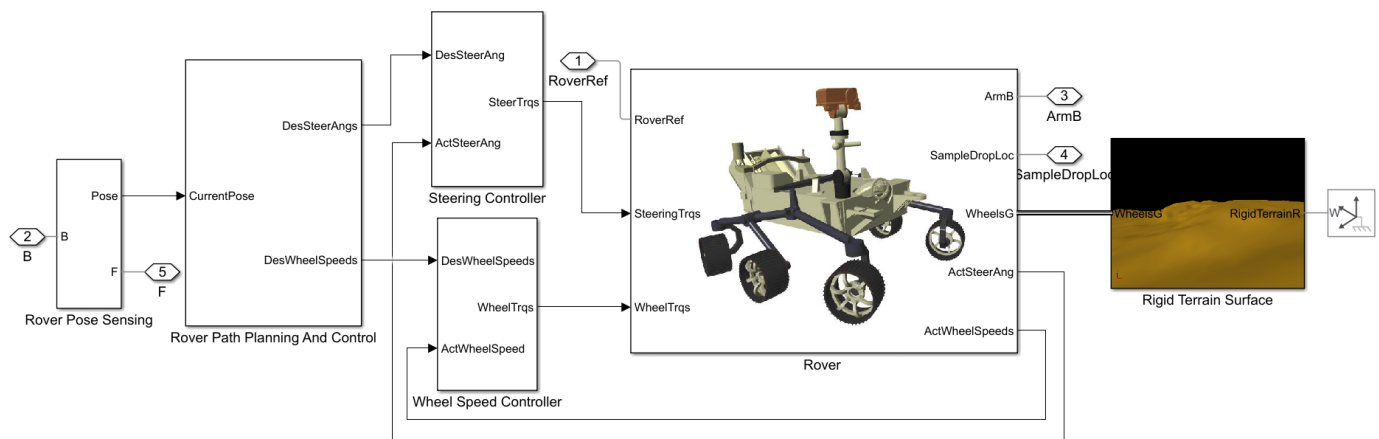


Mars Rover Animation



Rover Plant Model and Control

This subsystem models the rover, rigid terrain surface and the path planning and controls aspects of the system.



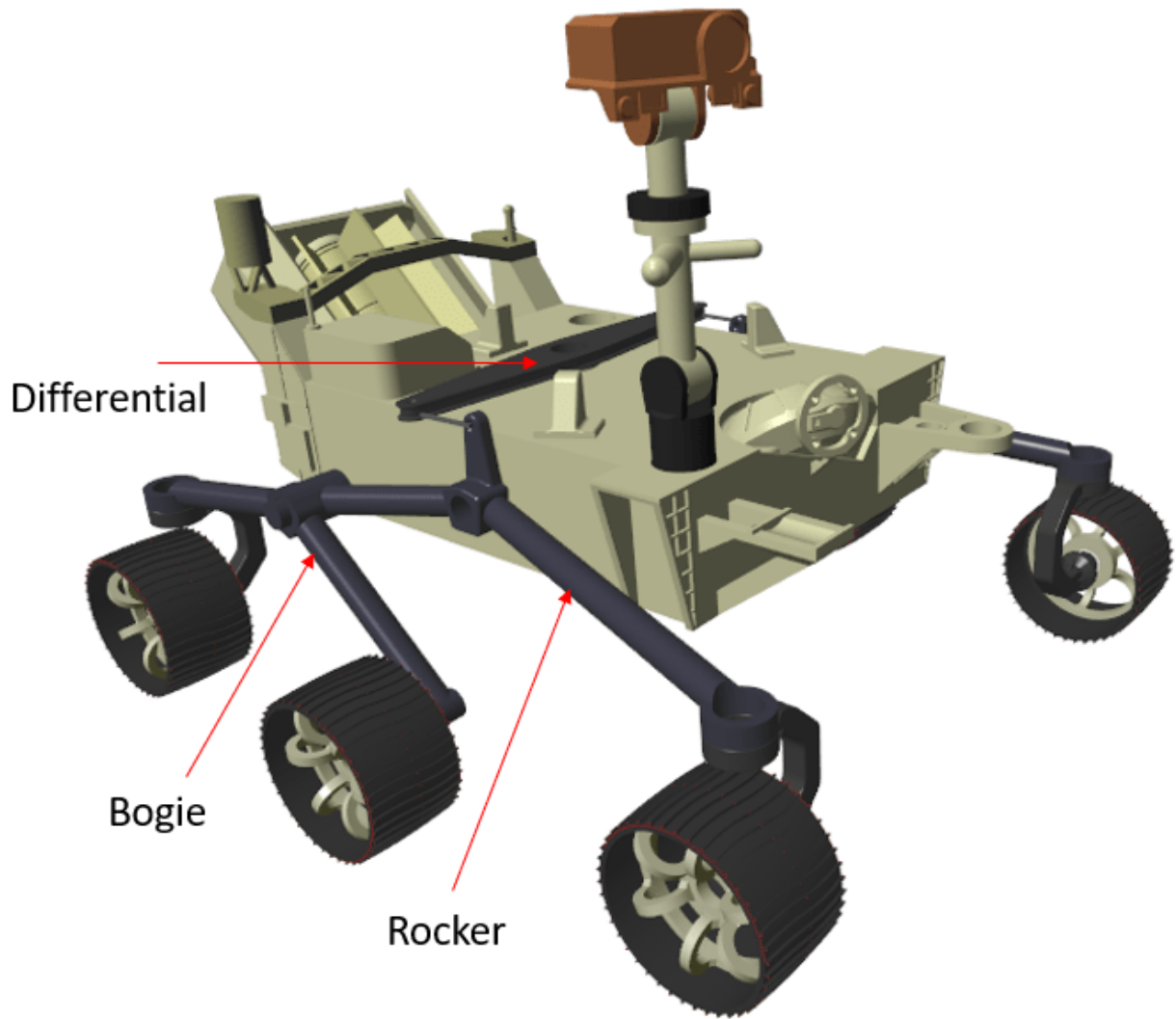
Rover Subsystem

This subsystem models various components of the rover like chassis, rocker-bogie suspension and wheels. The CAD parts for the geometry are imported into “Simscape Multibody”™ using the File Solid.

The rover's actuators correspond to the six torque-actuated revolute joints mounted to each of the six wheels for speed control and the four torque-actuated revolute joints mounted to the top of four

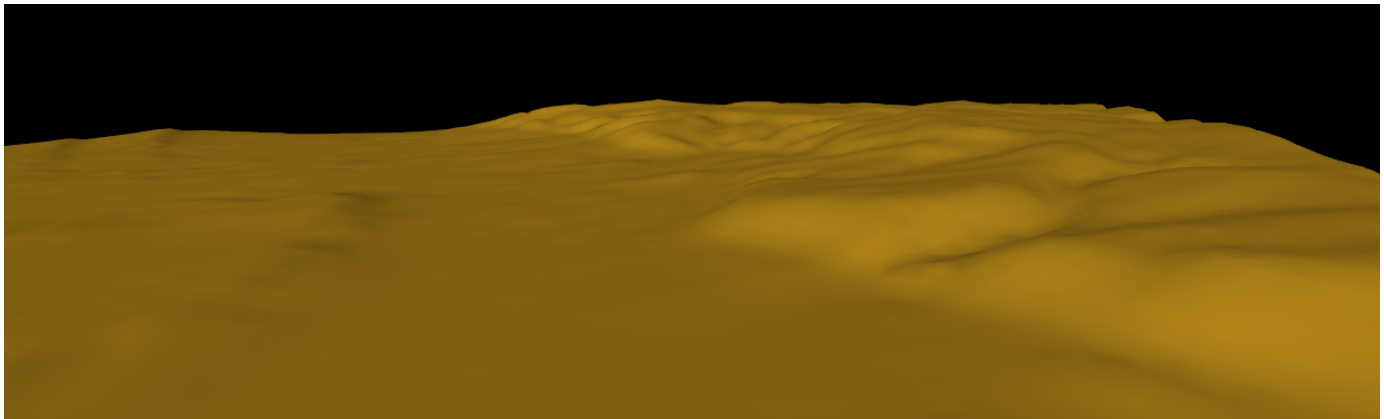
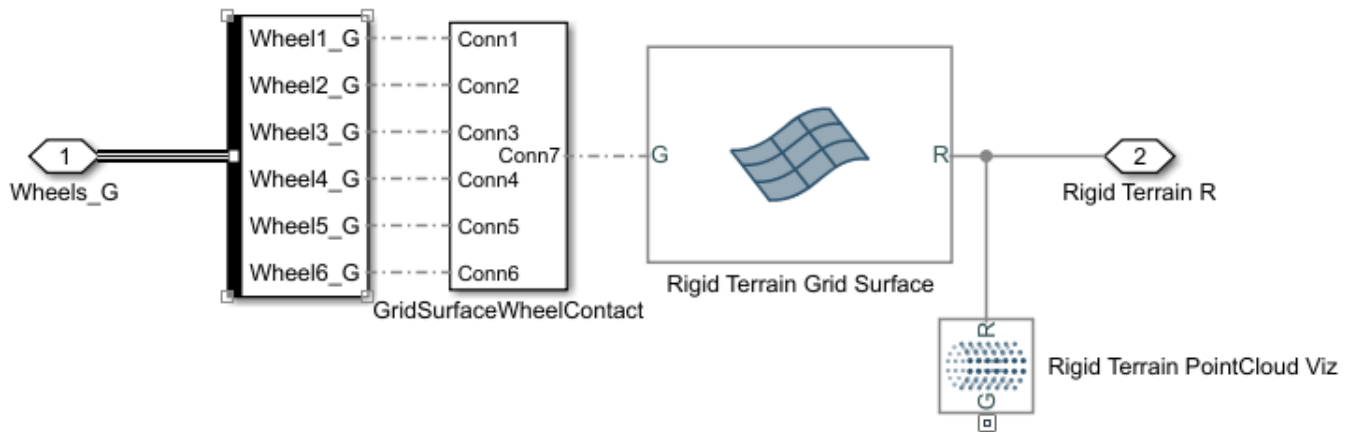
corner wheels used for steering. In addition to this, three main components of the suspension mechanism are also modeled, namely the differential arm, rocker and bogie.

The contact between the wheels and the rigid terrain is modeled using Point Cloud and Grid Surface contact pairs along with the Spatial Contact Force block. The points on each wheel's grousers are created using the Point Cloud block.



Rigid Terrain Surface

To model a Martian surface, a rigid terrain is created using the Grid Surface block. Refer to the file `rover_rigid_terrain_params.m` to setup the parameters needed to create the Grid Surface from a STL file.



Rover Path Planning and Control

This subsystem models rover's path tracking control system. The path consists of ordered waypoints in the X-Y plane which the rover is desired to pass through. These waypoints are assumed to be provided by a high-level path planner and would represent an obstacle free path for the rover. These waypoints can be loaded using the `roverDesiredPath.mat` file.

The goal of this subsystem is to first, compute the necessary steering angles and the wheel speeds needed to follow a desired path and a desired chassis linear velocity and second, to compute the necessary actuator torques needed to achieve these steering angles and wheel speeds.

For developing the path tracking controller, the following considerations are made:

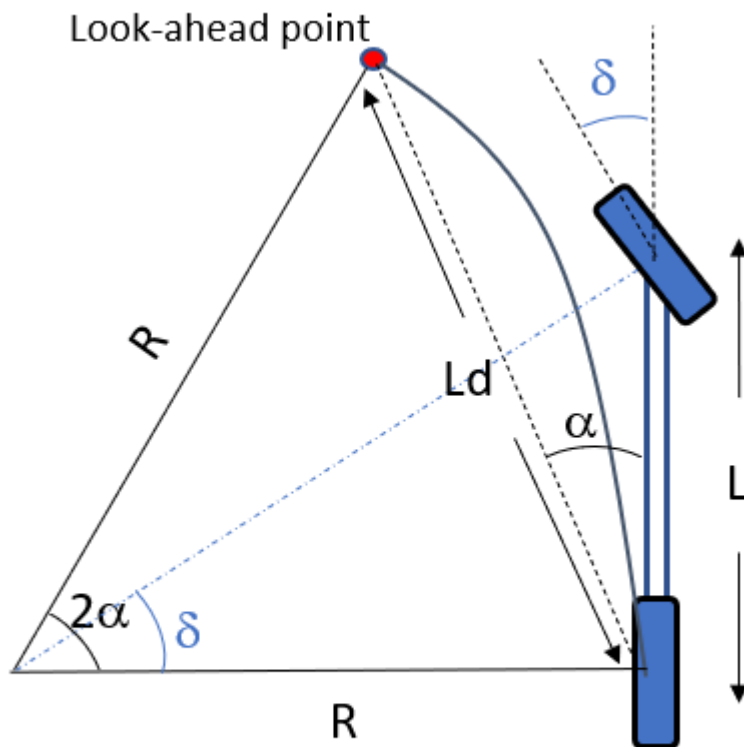
- Mars rovers are typically assumed to have low forward velocity (on the order of cm/s), therefore the dynamics of the motion are ignored and the controls problem is approached using kinematic equations only.[1]

- To simplify the kinematics formulation, the rover is assumed to be moving on a planar surface.[1]
- The four corner wheels of the rover have independent steering which can enable the rover to perform Ackerman steers. Based on this capability, the rover is considered to be using Ackerman steering.
- The Ackerman steering geometry is simplified by assuming a 2D geometric bicycle model with an equivalent turn radius. This simplification is done by representing each pair of wheels by a single wheel located in the middle and a single steering angle corresponding to the turn radius of the center of the rover.[1][2]
- The front and the rear wheels are considered to be steered symmetrically.
- The wheels are assumed to roll without slipping.

Based on the above considerations, a six wheel rover can be equivalently represented by a geometric bicycle model.[1]

Pure Pursuit (Robotics System Toolbox) is used for path tracking. This is a geometric algorithm that computes a target direction angle (α) needed to move the robot from its current position to reach some look-ahead point in front of the robot.[1]

Steering Angles Formulation



Pure Pursuit with bicycle geometry

The steering angles for each of the four corner wheels of the rover are derived in two steps. We first use the geometric bicycle model and the target direction angle (α) (provided by the Pure Pursuit Controller) to obtain the bicycle steering angle (δ) and the turn radius (R) as shown below.[2]

$$\delta = \tan^{-1}\left(\frac{2L \sin(\alpha)}{L_d}\right)$$

$$R = \frac{L}{\tan(\delta)}$$

where,

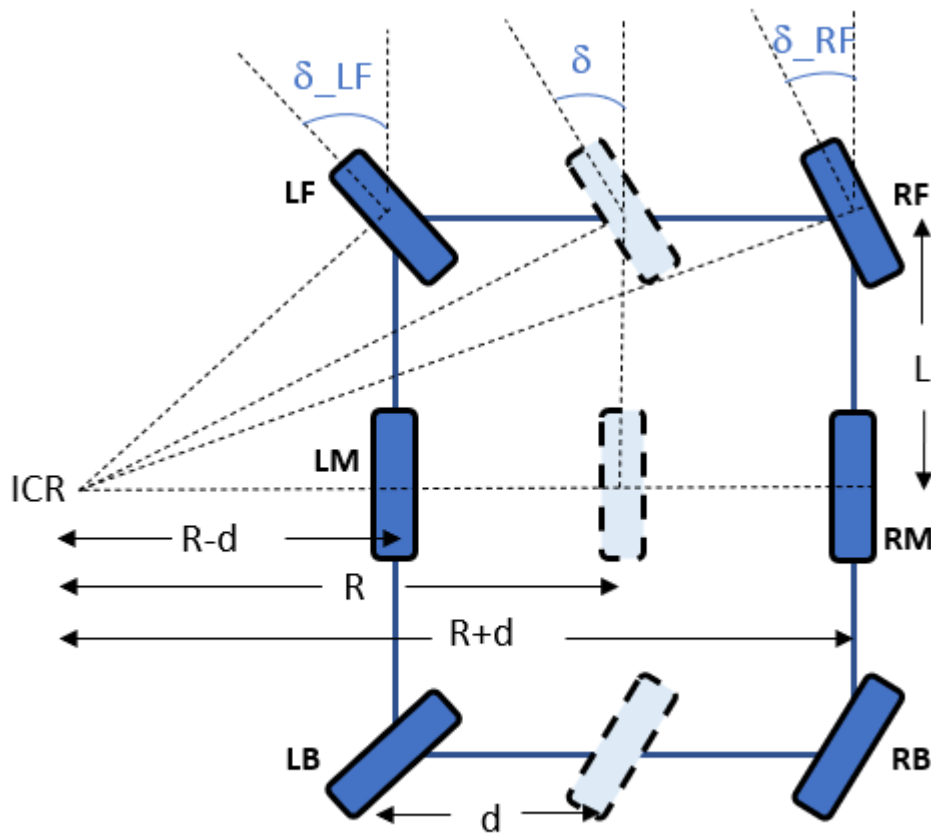
α : Target Dir Angle

δ : Bicycle Steering Angle

L : Bicycle Length

L_d : Look ahead distance for Pure Pursuit

R : Turn radius of rover center



Ackerman steering geometry of the rover

Based on the computed bicycle steering angle (δ) and the turn radius of the rover (R), we then obtain the individual steering angles ($\delta_{LF}, \delta_{LB}, \delta_{RF}, \delta_{RB}$) using the Ackerman steering geometry.[3]

$$\delta_{LF} = \tan^{-1}\left(\frac{L}{R-d}\right)$$

$$\delta_{RF} = \tan^{-1}\left(\frac{L}{R+d}\right)$$

$$\delta_{LB} = -\delta_{LF}$$

$$\delta_{RB} = -\delta_{RF}$$

where,

$\delta_{LF}, \delta_{LB}, \delta_{RF}, \delta_{RB}$: Steering angles of respective corner wheels

L : Bicycle Length ($0.5 * \text{Chassis Length}$)

d : $0.5 * \text{Chassis Width}$

ICR: Instantaneous Center of Rotation

Wheel Speed Formulation

Based on the Ackerman steering geometry and the turn radius (R), we also obtain the relationship between the chassis speed (V_c) and the wheel speeds (ω) as shown below [3] :

$$\omega_{LF} = \frac{V_C}{R_w} * \frac{\left(\sqrt{L^2 + (R-d)^2}\right)}{R}$$

$$\omega_{RF} = \frac{V_C}{R_w} * \frac{\left(\sqrt{L^2 + (R+d)^2}\right)}{R}$$

$$\omega_{LM} = \frac{V_C}{R_w} * \frac{(R-d)}{R}$$

$$\omega_{RM} = \frac{V_C}{R_w} * \frac{(R+d)}{R}$$

$$\omega_{LB} = \omega_{LF}$$

$$\omega_{RB} = \omega_{RF}$$

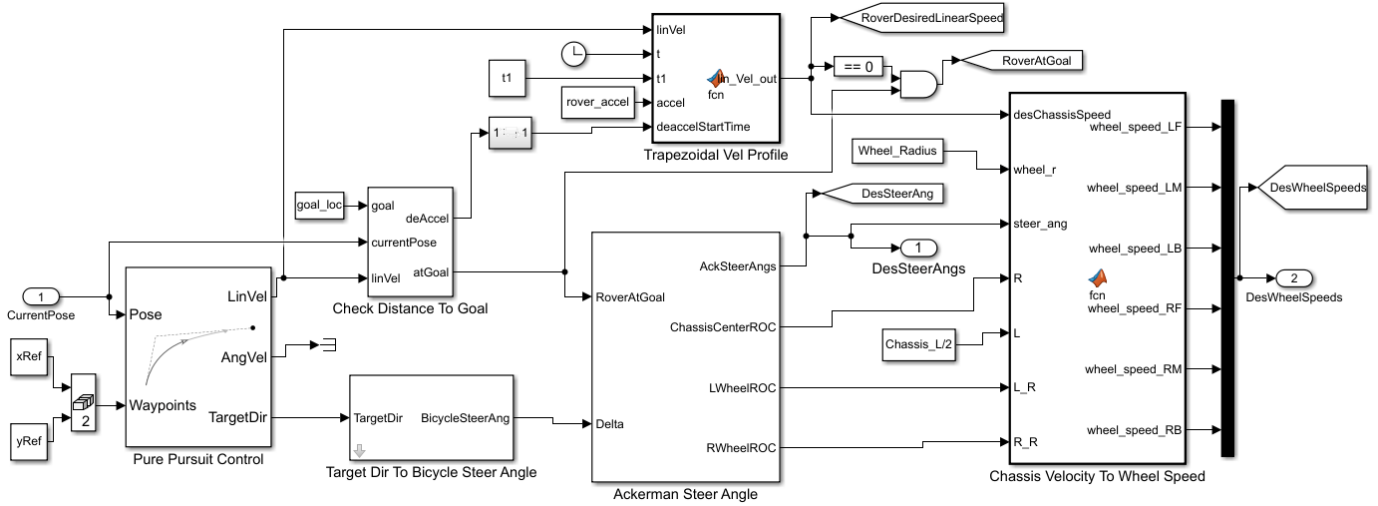
where,

$\omega_{LF}, \omega_{LM}, \omega_{LB}, \omega_{RF}, \omega_{RM}, \omega_{RB}$: Angular speeds of respective wheels

V_c : Linear speed of the rover chassis

R_w : Radius of the wheel

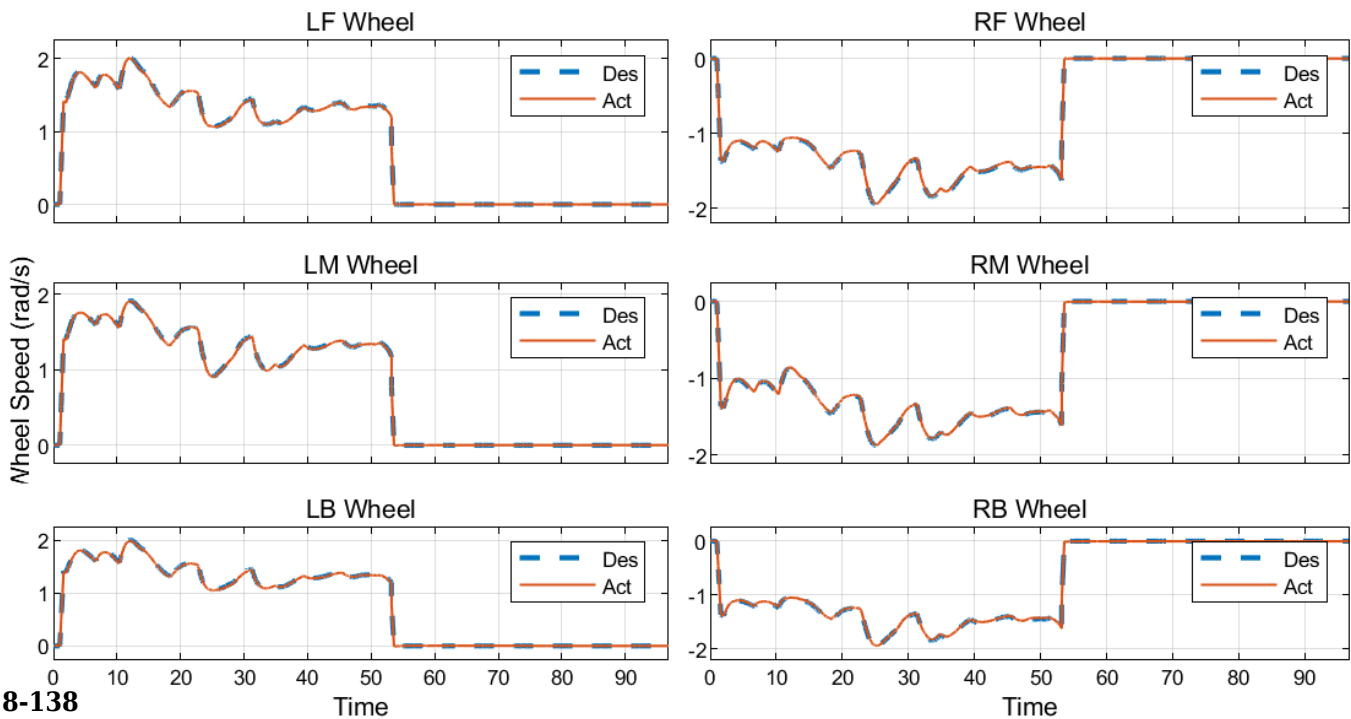
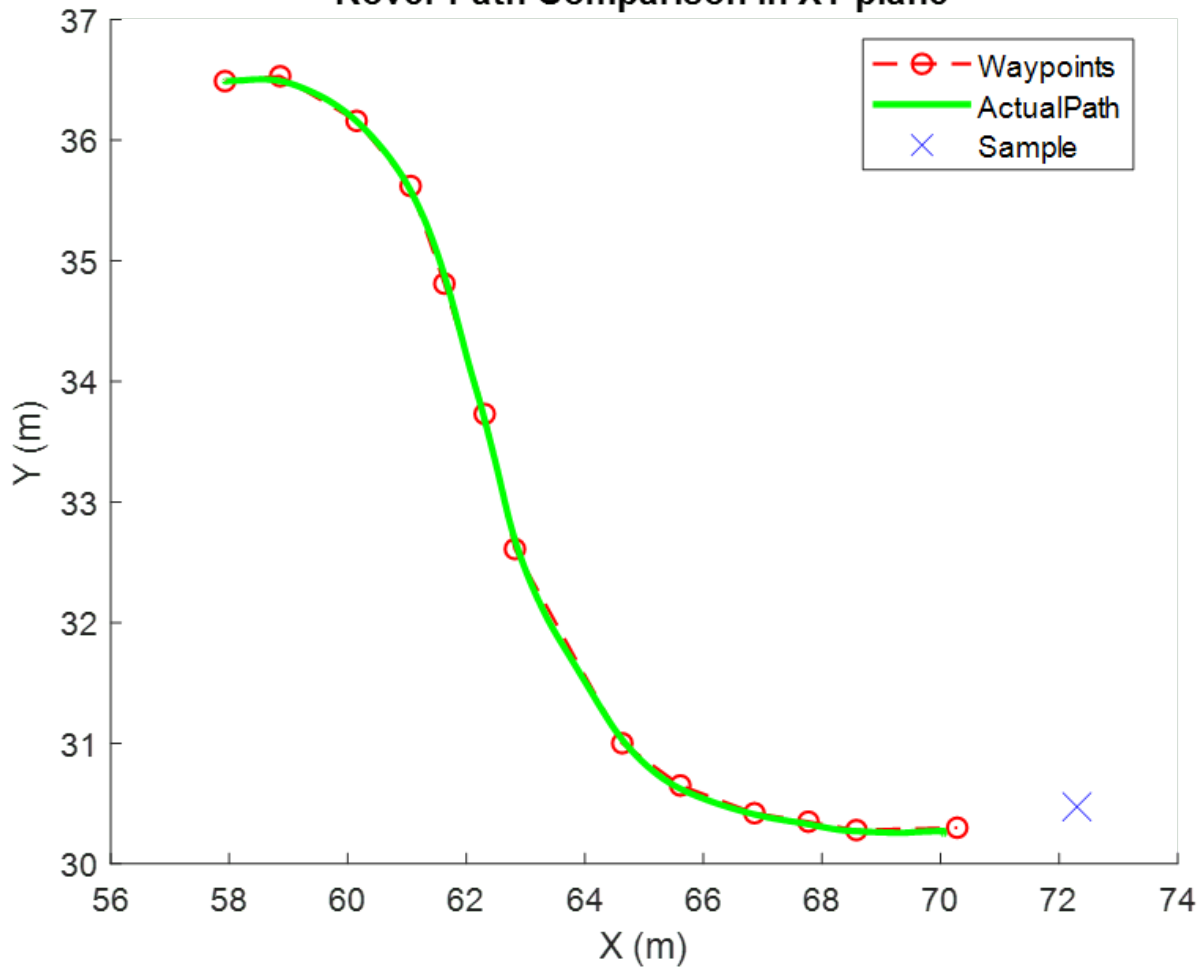
Once both the steering angles and the wheel speeds are formulated for the desired path and the desired chassis linear velocity, a PID controller is used to drive the actual steering angles and the angular rates of the actuators (revolute joints) to their desired values.

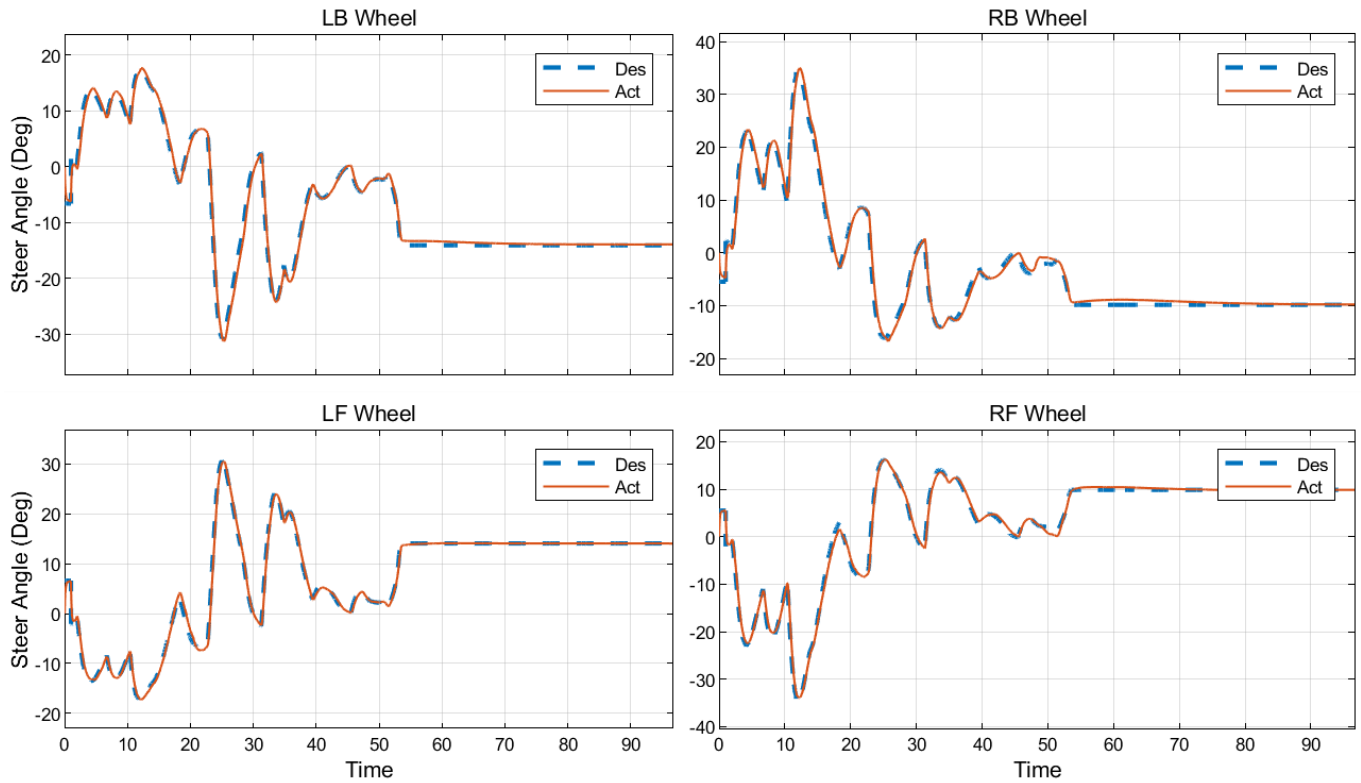


Rover Simulation Results : Path 1

Results for path 1 when the rover is moving at ~0.3 m/s on an uneven terrain. More quantities can be viewed at Rover Sensing subsystem in the model.

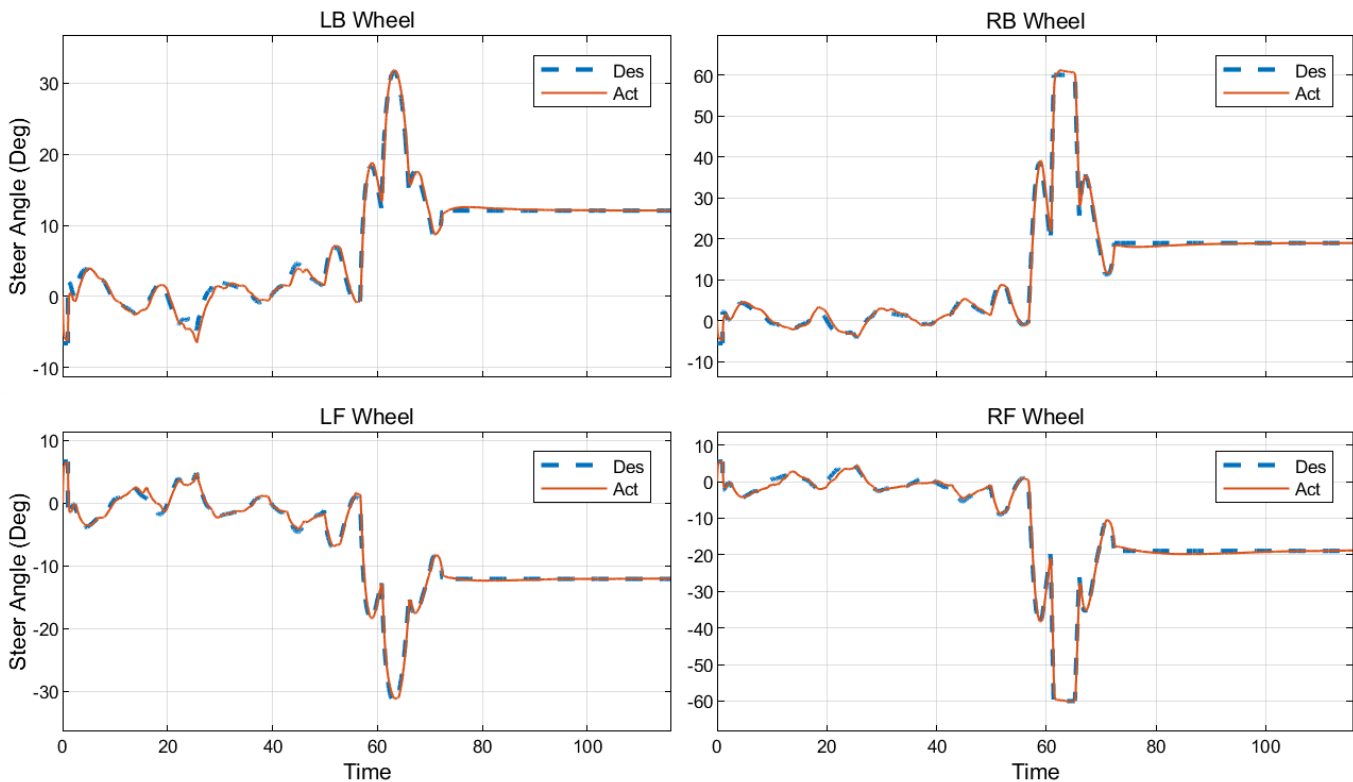
Rover Path Comparison in XY plane





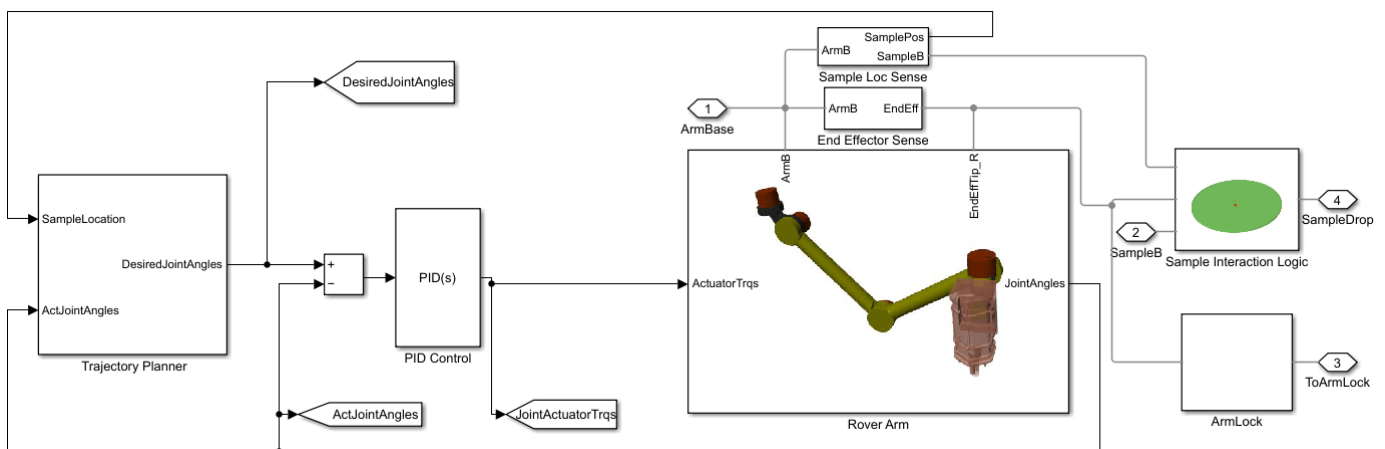
Rover Simulation Results : Path 2

Results for path 2 when the rover is moving at ~ 0.3 m/s on an uneven terrain. More quantities can be viewed at Rover Sensing subsystem in the model.



Manipulator Plant Model and Control

This subsystem models the rover's robot arm and its trajectory planning and control to pick and collect a sample from the surface.



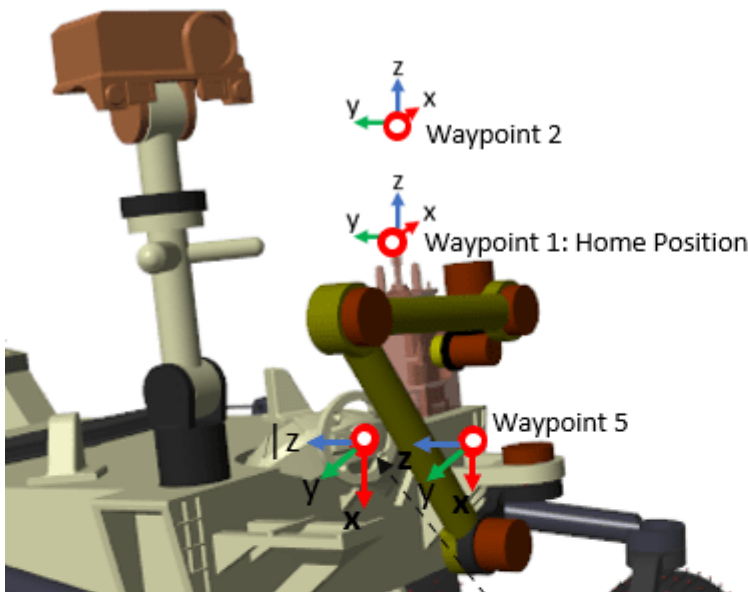
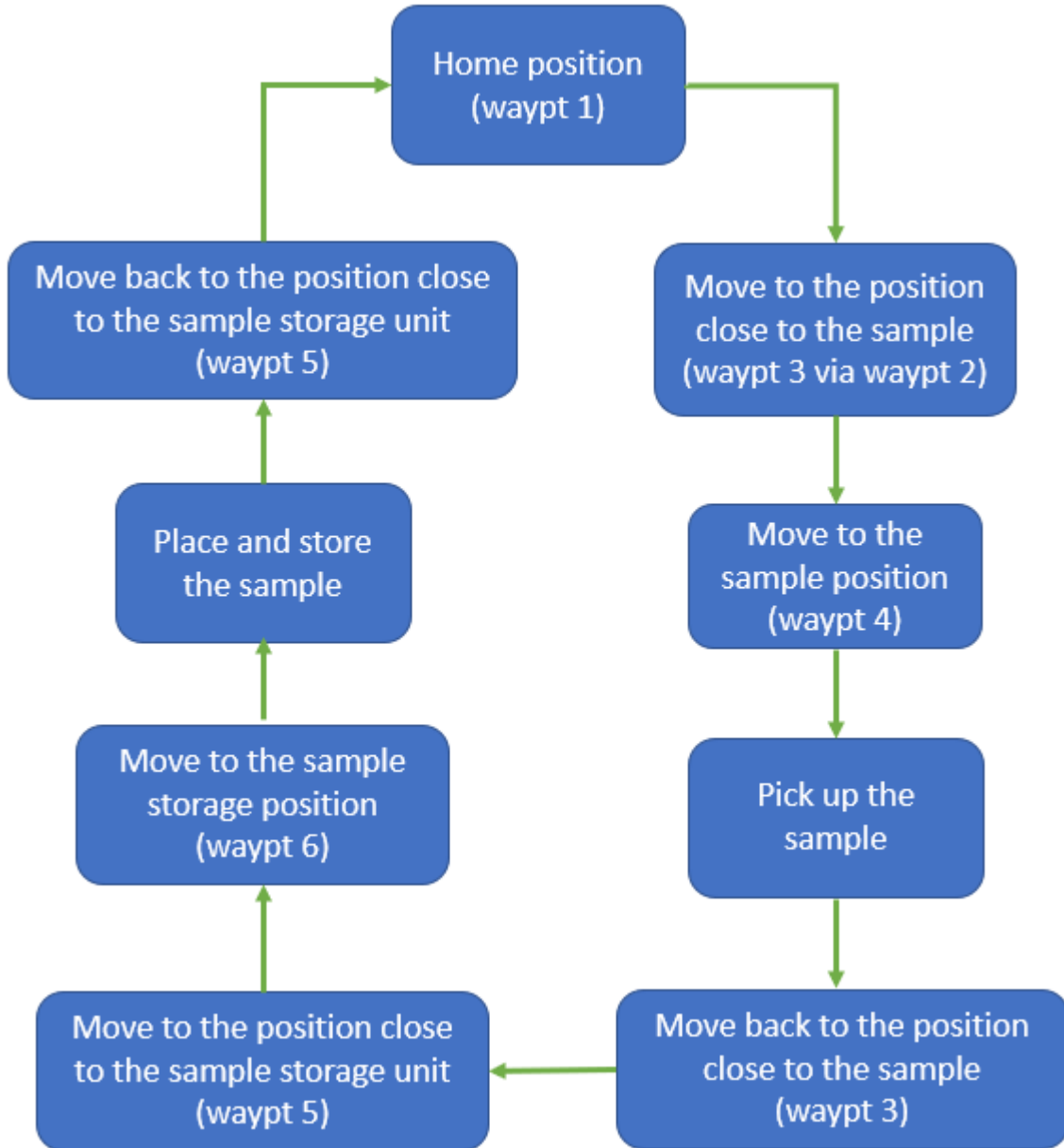
Rover Arm

The manipulator is modeled as a 6-DOF arm mounted on the front end of the chassis. Its actuators correspond to six torque-actuated revolute joints. To mimic sensors like encoders, the subsystem outputs the joint angles from each of the six revolute joints. We use a simplified model for the interaction between the end effector and the sample that leverages **joint mode switching**. When the end effector is sufficiently close to the sample, an initially disengaged 6-DOF joint connecting them

becomes engaged. This 6-DOF joint has tight position limits to keep the sample nearly constrained to the end effector. When the end effector containing the sample comes sufficiently close to the sample storage location, the engaged 6-DOF joint between the end effector and the sample is disengaged and an initially disengaged 6-DOF joint between the sample and the storage location is engaged. This 6-DOF joint also has tight position limits to keep the sample nearly constrained to the storage location on the chassis.

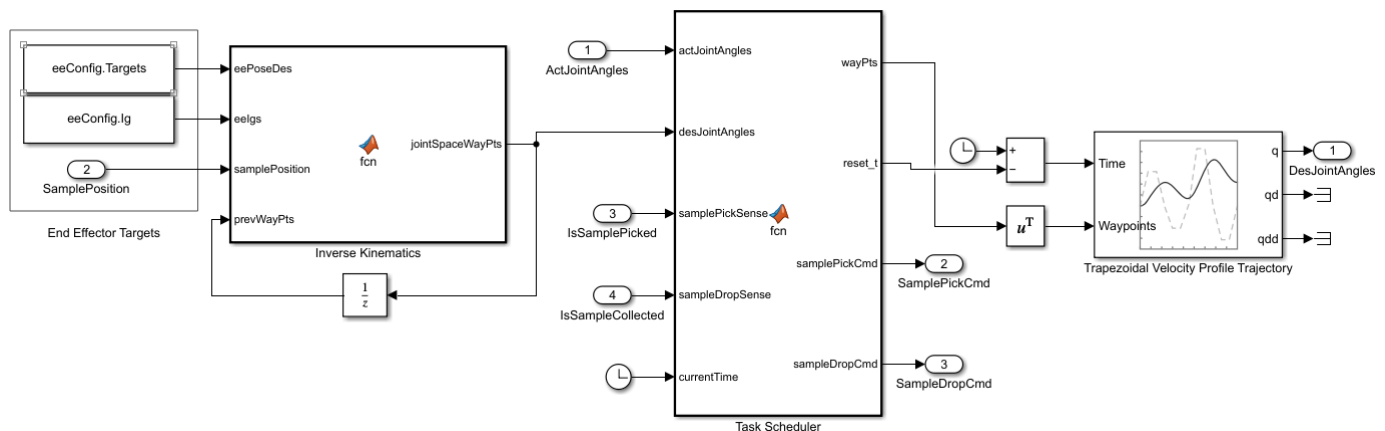
Planning and Control

The manipulator planning and control subsystems are enabled once the rover stops at the target position. Once the rover stops, the sample's location with respect to the arm base is computed using a Transform Sensor block (to mimic the camera on board).



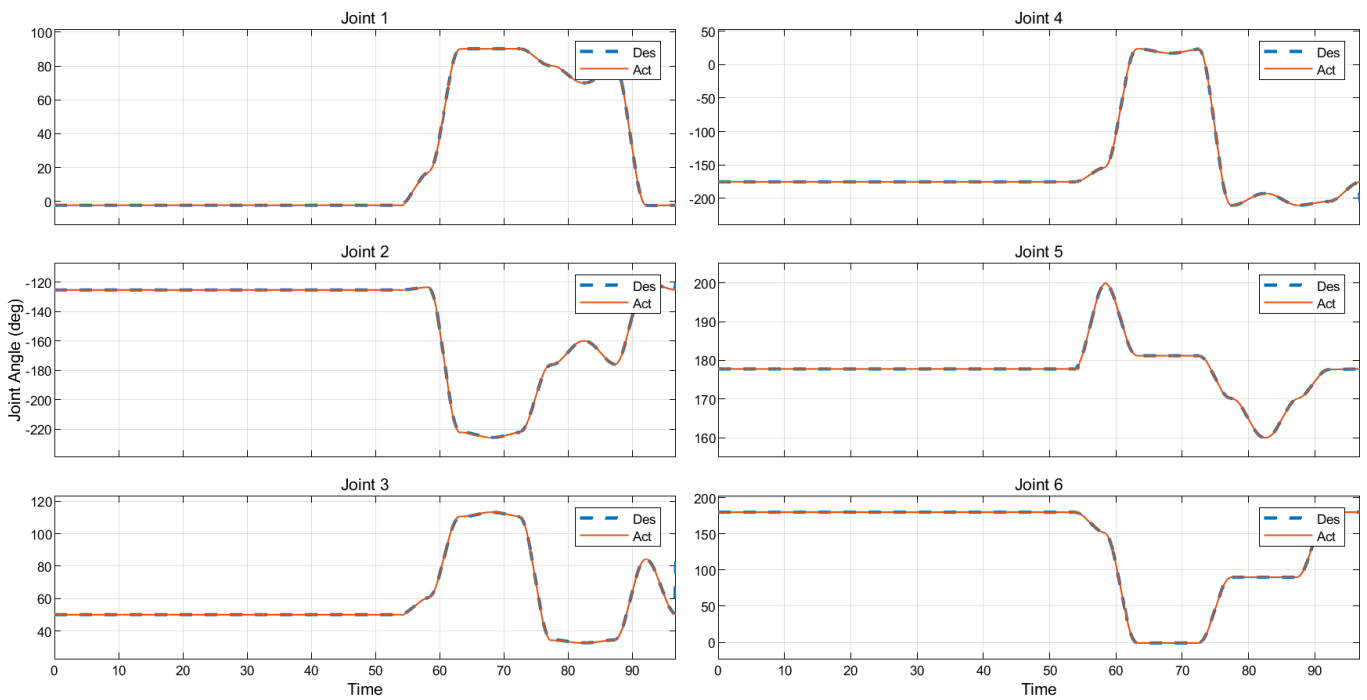
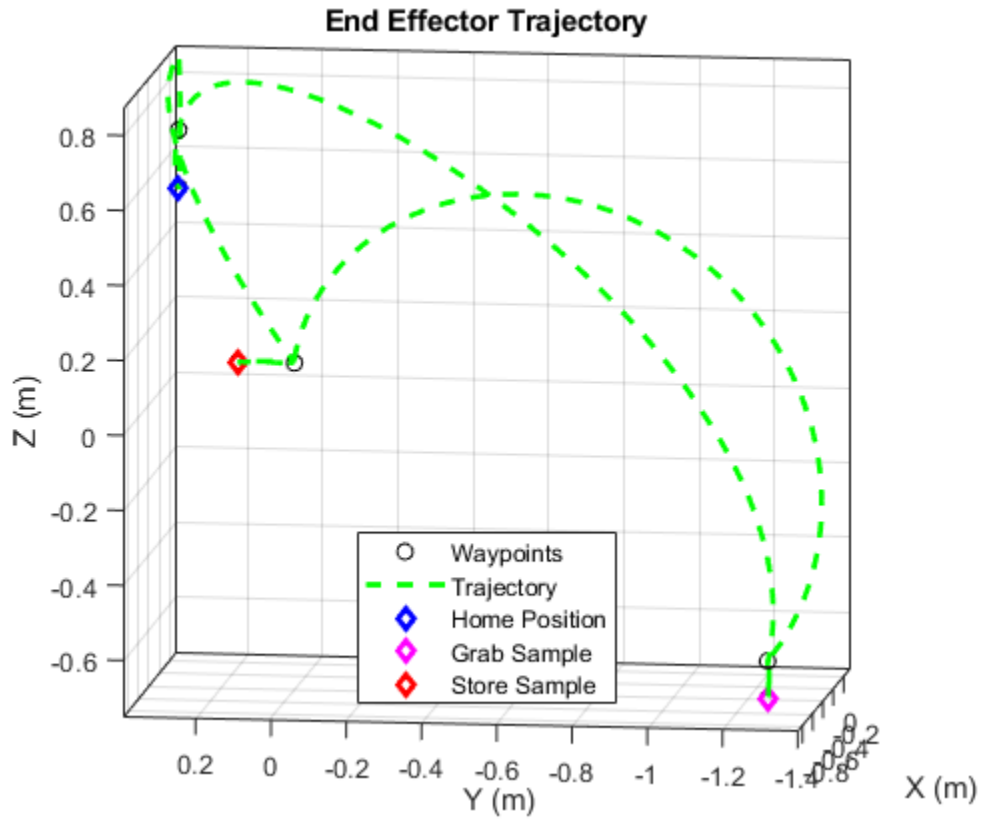
To plan the trajectory of the end effector, six waypoints are defined in the task space as shown above. Four of these waypoints (1, 2, 5 and 6) are precomputed and derived from the geometry of the rover chassis and the location of the storage unit. These can be loaded from `roverArmTaskSpaceConfig.mat`. Waypoints 3 and 4 are computed based on the sample position obtained from the transform sensor block. Using these six waypoints, the trajectory of the end effector is planned via a joint space trajectory planning approach. The planner first converts all the six task space waypoints to joint space waypoints using the inverse kinematics module (refer to `sm_mars_rover_arm_ik.m` implemented using `KinematicsSolver`). These joint space waypoints are then used by a MATLAB function block Task Scheduler to advance the arm through the series of modes as shown above. The scheduler advances the end effector to the next mode when it reaches a target position within some tolerance value. Each task from the scheduler is an input to the Trapezoidal Velocity Profile Trajectory (Robotics System Toolbox) block, which computes a smooth trajectory between each waypoint in the joint space.

Once a joint space trajectory is generated, a PID controller is used to drive the actual positions of the actuators to their desired values.



Manipulator Simulation Results

Results showing the end effector trajectory passing through the desired waypoints and the comparison between desired and actual actuator joint angles. More quantities can be viewed at Manipulator Sensing subsystem in the model.



References

- [1] Filip, Jan, Martin Azkarate, and Gianfranco Visentin. "Trajectory control for autonomous planetary rovers." In *14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. 2017.
- [2] Snider, Jarrod M. "Automatic steering methods for autonomous automobile path tracking." *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).
- [3] X. Wu, L. Yang and M. Xu, "Speed following control for differential steering of 4WID electric vehicle," *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, 2014, pp. 3054-3059, doi: 10.1109/IECON.2014.7048945.

See Also

KinematicsSolver | Grid Surface | Point Cloud

Creating a Mobile Robot using a MATLAB App

This example demonstrates how a multibody system can be built using an interactive MATLAB app. In this example an application for exploring the design space of a multibody system is shown. The system here is a mobile manipulator with four omni-directional wheels.


MATLAB App Designer is used to build a GUI that allows the user to vary various parameters governing the mobile robot design, and to see an interactive visualization of the robot as the parameters are changed. It also has a button to create a Simulink Model based on the current design parameters.

Each parameter change triggers a call to the custom function **makeMobileRobot**. This is the top level function which calls different sub functions like **createMobileRobotChassis**, **createOmniDirectionalWheel** and **createRobotArm** to create and connect the various components of the mobile robot. Each of these custom functions use different classes and methods under the **simscape.multibody.*** package for building the different components.

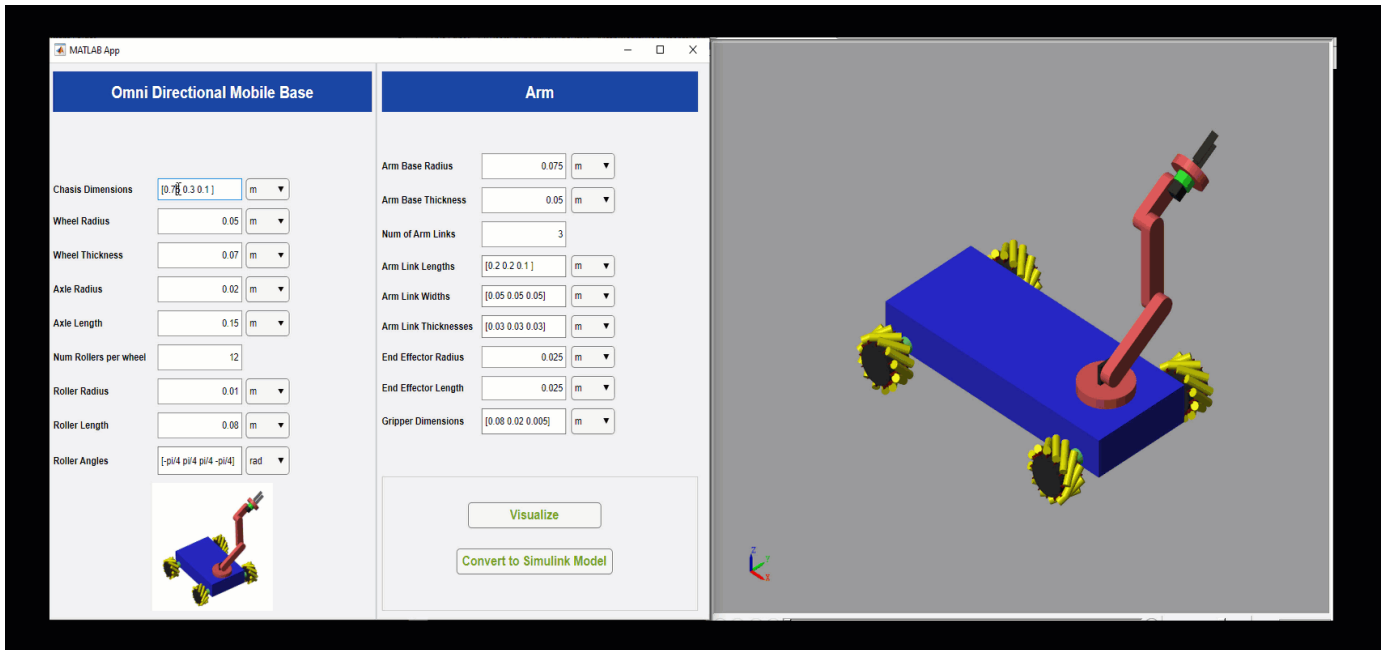
We can also build this system by setting up the parameters directly in MATLAB in a programmatic way and then calling the function **makeMobileRobot**. Refer to the script **buildMobileRobot.m** to learn more.

To view the app run the following command :

```
openExample('sm/MobileRobotBuilderAppExample');  
run('buildMobileRobotApp.mlapp');
```

Omni Directional Mobile Base	Arm
<p>Chasis Dimensions <input type="text" value="[0.75 0.3 0.1]"/> m ▾</p> <p>Wheel Radius <input type="text" value="0.05"/> m ▾</p> <p>Wheel Thickness <input type="text" value="0.07"/> m ▾</p> <p>Axle Radius <input type="text" value="0.02"/> m ▾</p> <p>Axle Length <input type="text" value="0.15"/> m ▾</p> <p>Num Rollers per wheel <input type="text" value="12"/></p> <p>Roller Radius <input type="text" value="0.01"/> m ▾</p> <p>Roller Length <input type="text" value="0.08"/> m ▾</p> <p>Roller Angles <input type="text" value="[-pi/4 pi/4 pi/4 -pi/4]"/> rad ▾</p>	<p>Arm Base Radius <input type="text" value="0.075"/> m ▾</p> <p>Arm Base Thickness <input type="text" value="0.05"/> m ▾</p> <p>Arm Link Lengths <input type="text" value="[0.2 0.2 0.1]"/> m ▾</p> <p>Arm Link Widths <input type="text" value="[0.05 0.05 0.05]"/> m ▾</p> <p>Arm Link Thicknesses <input type="text" value="[0.03 0.03 0.03]"/> m ▾</p> <p>End Effector Radius <input type="text" value="0.025"/> m ▾</p> <p>End Effector Length <input type="text" value="0.025"/> m ▾</p> <p>Gripper Dimensions <input type="text" value="[0.08 0.02 0.005]"/> m ▾</p>
	<p><input type="checkbox"/> Auto Update Visualization</p> <p><input type="button" value="Update Visualization"/></p> <p><input type="button" value="Convert to Simulink Model"/></p>

App Interface



See Also

`simscape.multibody.Multibody`

Creating a Robotic Gripper Multibody in MATLAB

This example constructs a robotic gripper multibody in MATLAB. It demonstrates how various classes under **simscape.multibody.*** package can be used to build a hierarchical multibody system.

The gripper has a palm which is a **RigidBody** component and has fingers which are themselves **Multibody** components. Each of the articulated finger multibody comprises of rigid finger segments connected via revolute joint knuckles.

To construct the gripper, run the following code in MATLAB :

```
import simscape.Value;

% Number of fingers
numFingers = 5;

% Palm dimensions
palmDims = Value([8 40], 'mm');

% Finger segment number and dimensions
segmentDims = Value([30 8; ... % Proximal segment
                    30 6; ... % Middle segment
                    40 5], ... % Distal segment
                    'mm');

% Finger tip dimensions
tipRad = Value(9, 'mm');

% Finger bend angles
bendAngles = Value([-30, +30, +25], 'deg');

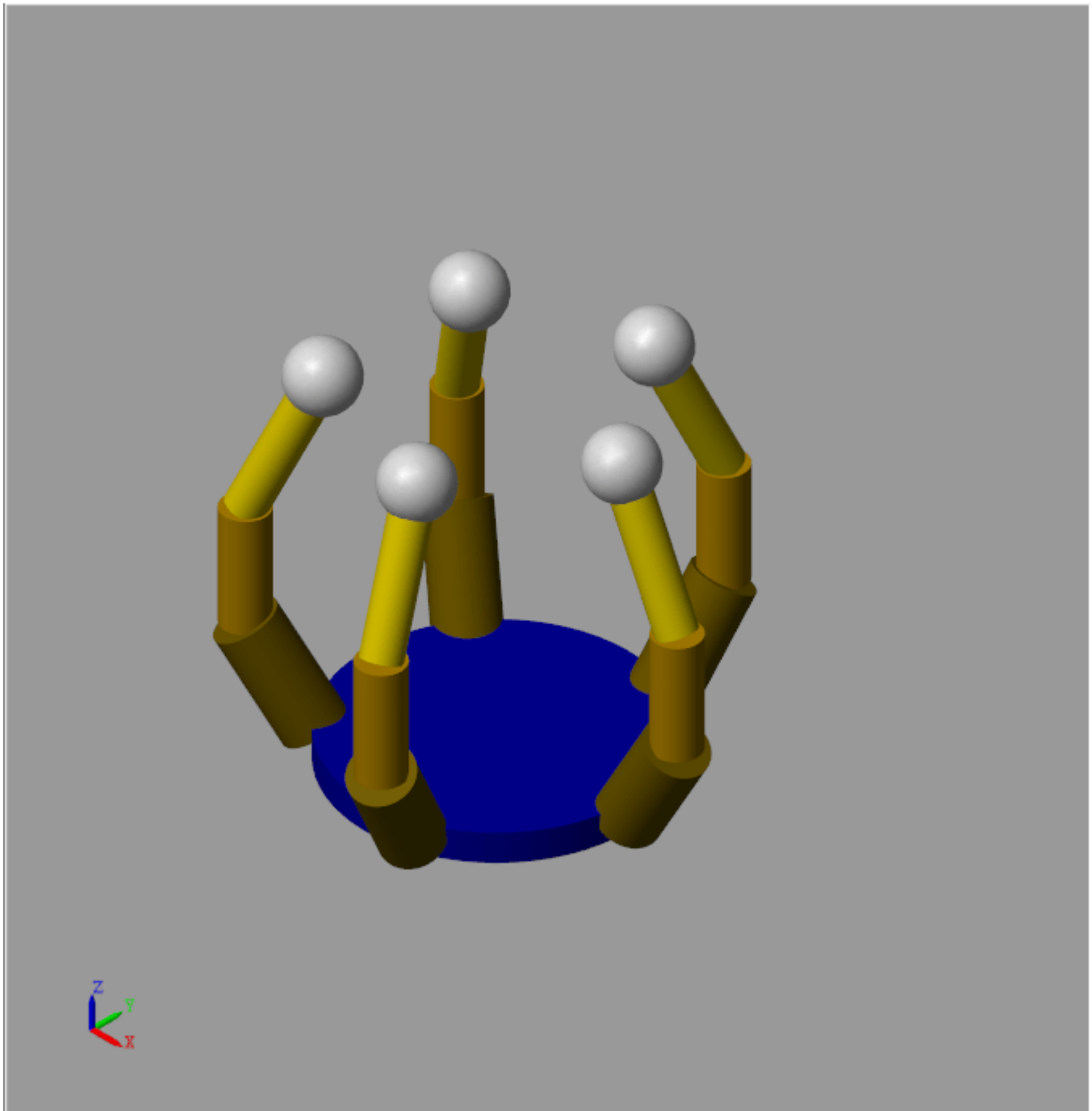
% Finger segment colors
colors = [0 0 .7; ... % Palm
         .5 .4 0; ... % Proximal segment
         .8 .6 0; ... % Middle segment
         1 .9 0; ... % Distal segment
         1 1 1]; % Tip

% Construct the gripper
[gripper, gripper_op] = robotGripper(numFingers, palmDims, segmentDims, tipRad, bendAngles, colors);

% Visualize the gripper
cmb = compile(gripper);
visualize(cmb, computeState(cmb, gripper_op), 'vizGripper');
```

To perform any simulation workflows, a simulink model can be created from the gripper multibody object using its **makeBlockDiagram** method.

```
makeBlockDiagram(gripper, gripper_op, 'gripperModel');
```



See Also

`simscape.multibody.Multibody` | `simscape.multibody.RigidBody`

Creating a Four Bar Multibody Mechanism in MATLAB

This example constructs a four bar mechanism in MATLAB using Simscape Multibody. It demonstrates various classes under **simscape.multibody.*** package to build a multibody system in MATLAB.

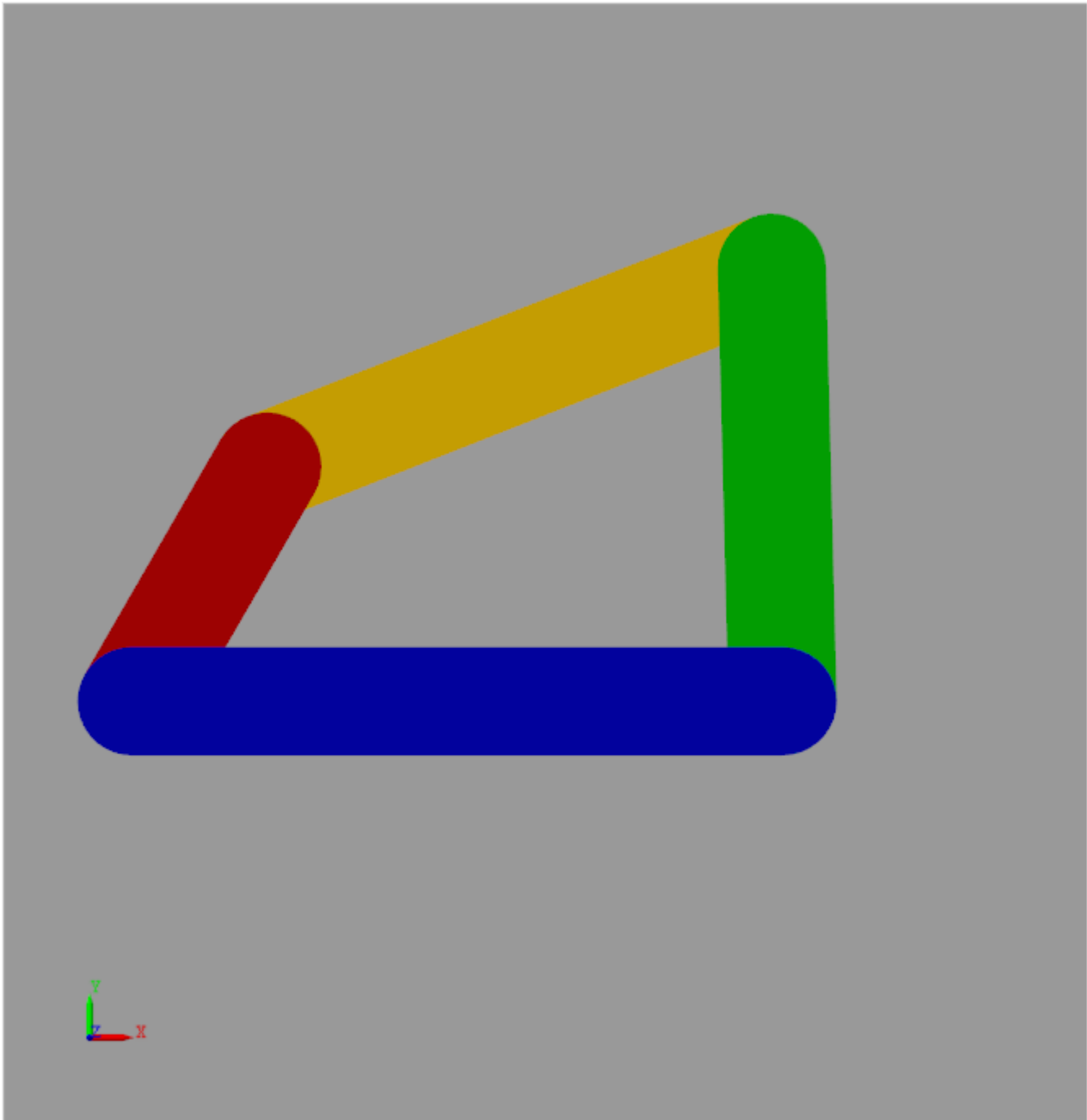
A custom function **fourbar** constructs the four-bar multibody with user specified link lengths. Each parameter is expected to be a Simscape Value, specifying a scalar length.

```
openExample('sm/CreateAFourBarMechanismInMATLAB');  
[fb, op] = fourBar(simscape.Value(12, 'cm'), simscape.Value(10, 'cm'), simscape.Value(5, 'cm'), s
```

The four-bar is constructed so that the X-Y plane is the plane of motion, with gravity in the -Y direction. Output `op` is an operating point that sets the bottom left joint angle to a nominal value with high priority.

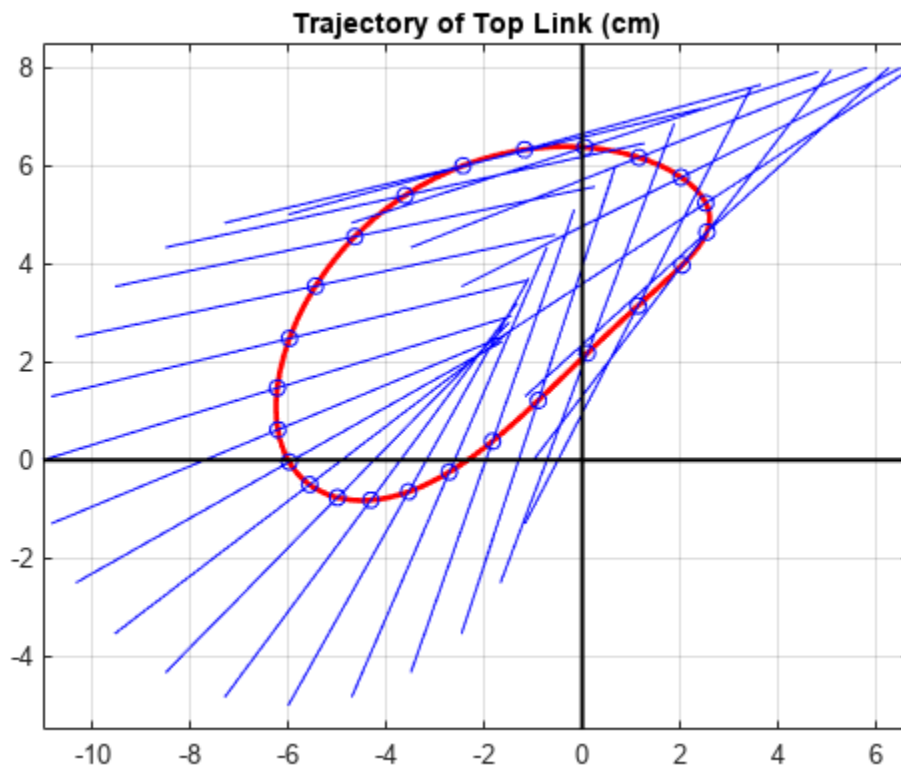
To visualize the mechanism, run the following :

```
cmb = compile(fb);  
visualize(cmb, computeState(cmb, op), 'vizFourBar');
```



Once the four bar mechanism is created, we can also perform some analysis on it like computing and plotting the trajectory of the top (distal) link of the specified four-bar mechanism. The function **fourBarMotionPlot** plots the trajectory of the top link over a full 360 deg range of the crank angle (when the left link is a crank of the four bar). This function varies the crank angle over a full 360-deg range in steps of 1 deg. The trajectory of the midpoint of the top link is plotted over these 1-deg steps to form a (typically) smooth curve. The coordinates along this curve are relative to the world frame.

```
fourBarMotionPlot(fb);
```



See Also

`simscape.multibody.Multibody`

Creating a Simple Pendulum in MATLAB

This example constructs a simple pendulum in MATLAB. It demonstrates various classes under **simscape.multibody.*** package to build a simple multibody system in MATLAB.

The pendulum consists of a single link suspended at one end from a pivot. It is considered to swing in the X-Y plane with gravity in the -Y direction. The zero position corresponds to the pendulum hanging straight down and the positive velocities correspond to counter-clockwise motion about the pivot when the pendulum is viewed from the +Z axis.

The following sections of code builds the simple pendulum.

```
% Import MATLAB Classes package so the "simscape.multibody." qualifier is not needed in front of
% Also import useful classes from Simscape. (This shortcut can also be used at the command line
import simscape.multibody.* simscape.Value simscape.op.*
```

Construct the pendulum link with a brick geometry.

```
linkDimensions = Value([20 2 1], "cm");
link = RigidBody;
color = [0 1 1];
addComponent(link, "Body", "reference", Solid(Brick(linkDimensions), SimpleVisualProperties(color));
addFrame(link, "pin", "reference", ...
    RigidTransform(StandardAxisTranslation(linkDimensions(1)/2, Axis.NegX));
addConnector(link, "pin");
```

Construct a rigid transform that aligns the +X axis of the follower frame with the -Y direction of the base frame. This is used to position the pivot so that the pendulum swigs in the X-Y plane and that it hangs straight down at the zero position.

```
rotator = RigidTransform(AlignedAxesRotation(Axis.PosX, Axis.NegY, Axis.PosZ, Axis.PosZ));
```

Create the multibody object, and modify gravitational acceleration to be in the -Y direction instead of the default -Z direction.

```
pendulum = Multibody;
pendulum.Gravity = circshift(pendulum.Gravity, -1);
```

Add the necessary components to the multibody object

```
addComponent(pendulum, "World", WorldFrame);
addComponent(pendulum, "Rotator", rotator);
addComponent(pendulum, "Link", link);
addComponent(pendulum, "Pivot", RevoluteJoint);
```

Make appropriate connections between components. Multibody object's connectVia method is used to connect two frames together across a joint.

```
connect(pendulum, "World/W", "Rotator/B");
connectVia(pendulum, "Pivot", "Rotator/F", "Link/pin");
```

Construct an operating point to position the pendulum link 30 degrees counter-clockwise from the vertical, and rotating counter-clockwise at 1 revolution/sec.

```
op = OperatingPoint;  
op("Pivot/Rz/q") = Target(simscape.Value(30, "deg"), "High");  
op("Pivot/Rz/w") = Target(simscape.Value(1, "rev/s"), "High");
```

Visualize the pendulum.

```
compiledPendulum = compile(pendulum);  
state = computeState(compiledPendulum, op);  
visualize(compiledPendulum, state, "simple_pendulum");
```

Generate a Simulink model of the pendulum

```
makeBlockDiagram(pendulum, op, "simple_pendulum")
```

See Also

`simscape.multibody.Multibody` | `simscape.multibody.RigidBody`

Creating a Multibody with different joints in MATLAB

This example shows how to build a multibody containing various joints in MATLAB. It also shows a way of setting operating point targets for all the joint primitives for all the joint types. The following sections of code demonstrates the approach.

Create a multibody object and add the necessary components like WorldFrame and Gravity.

```
import Simscape.Value Simscape.Multibody.*;

mb = Multibody;
addComponent(mb, 'World', WorldFrame());

sep = Value(20, 'cm'); % Separation between blocks

% Translucent ground plane
ground = Solid(Brick([5 4] * sep, Value(1, 'cm'))), UniformDensity, ...
    SimpleVisualProperties(0.7 * [1 1 1], 0.6));
addComponent(mb, 'Ground', ground);
addComponent(mb, 'Ground_Pose', RigidTransform(CartesianTranslation([2 1.5 0] * sep)));
connectVia(mb, 'Ground_Pose', 'World/W', 'Ground/R');

% Zero gravity
mb.Gravity = Value([0 0 0], 'm/s^2');
```

Add all types of joints to the multibody object using the custom function **addJoint**.

```
% Row 1: Single-primitive joints
addJoint(mb, 'Revolute', [0 0] * sep, [1 0 0]);
addJoint(mb, 'Prismatic', [1 0] * sep, [1 1 0]);
addJoint(mb, 'Spherical', [2 0] * sep, [0 .8 0]);
addJoint(mb, 'LeadScrew', [3 0] * sep, [0 0 .9]);
addJoint(mb, 'ConstantVelocity', [4 0] * sep, [.8 0 .8]);

% Row 2: 2-DOF multi-primitive joints
addJoint(mb, 'Universal', [0.5 1] * sep, [.95 .4 0]);
addJoint(mb, 'Rectangular', [1.5 1] * sep, [.4 .9 0]);
addJoint(mb, 'Cylindrical', [2.5 1] * sep, [0 .7 .7]);
addJoint(mb, 'PinSlot', [3.5 1] * sep, [.6 0 1]);

% Row 3: 3-DOF and 4-DOF multi-primitive joints
addJoint(mb, 'Gimbal', [0 2] * sep, [.37 .86 .57]);
addJoint(mb, 'Cartesian', [1 2] * sep, [.28 .55 .96]);
addJoint(mb, 'Planar', [2 2] * sep, [.91 .63 .10]);
addJoint(mb, 'Bearing', [3 2] * sep, [.96 .49 .80]);
addJoint(mb, 'Telescoping', [4 2] * sep, [.8 .8 .5]);

% Row 4: 0-DOF and 6-DOF joints
addJoint(mb, 'Weld', [1 3] * sep, .1 * [1 1 1]);
addJoint(mb, 'Bushing', [2 3] * sep, .8 * [1 1 1]);
addJoint(mb, 'SixDof', [3 3] * sep, 1 * [1 1 1]);
```

Now in order to create operating points for specifying position or velocity targets to any joint, we will need the path to each of the joint primitives. The Multibody object provides a method **jointPrimitivePaths** to list the paths to all the joint primitives in the multibody. We use this list as one

of the inputs to the custom function **radnomOpPoint** which creates an operating point specifying random velocities for all the joints.

```
jointPrimPaths = jointPrimitivePaths(mb);
op = randomOpPoint(jointPrimPaths, Value(90, 'deg/s'), Value(1, 'cm/s'));
```

Once the operating point is created we can compile and use the **computeState** method to view if the above random velocity targets are achieved.

```
cmb = compile(mb);
state = computeState(cmb,op)

state =
  State:

  Status: Valid

  Assembly diagnostics:
  x
  Revolute_Joint
    Joint successfully assembled
    Rz
      Free position value: +0 (deg)
      High priority velocity target +23.5864 (deg/s) achieved
  Prismatic_Joint
    Joint successfully assembled
    Pz
      Free position value: +0 (m)
      High priority velocity target +0.210897 (cm/s) achieved
  Spherical_Joint
    Joint successfully assembled
    S
      Free position value: [+0 +0 +0], +0 (deg)
      High priority velocity target [-80.4494 -62.6214 -85.1939] (deg/s) achieved
  LeadScrew_Joint
    Joint successfully assembled
    LSz
      Free position value: +0 (deg)
      High priority velocity target +0.0848914 (cm/s) achieved
  ConstantVelocity_Joint
    Joint successfully assembled
    CV
      High priority position target +90 (deg) achieved
      High priority velocity target [-5 +10] (deg/s) achieved
  Universal_Joint
    Joint successfully assembled
    Rx
      Free position value: +0 (deg)
      High priority velocity target +73.9708 (deg/s) achieved
    Ry
      Free position value: +0 (deg)
      High priority velocity target +10.0194 (deg/s) achieved
  Rectangular_Joint
    Joint successfully assembled
    Px
      Free position value: +0 (m)
      High priority velocity target +0.162366 (cm/s) achieved
    Py
```

```
      Free position value: +0 (m)
      High priority velocity target +0.00724535 (cm/s) achieved
Cylindrical_Joint
  Joint successfully assembled
  Rz
    Free position value: +0 (deg)
    High priority velocity target -12.8893 (deg/s) achieved
  Pz
    Free position value: +0 (m)
    High priority velocity target +0.984314 (cm/s) achieved
PinSlot_Joint
  Joint successfully assembled
  Px
    Free position value: +0 (m)
    High priority velocity target +0.454274 (cm/s) achieved
  Rz
    Free position value: +0 (deg)
    High priority velocity target +21.4513 (deg/s) achieved
Gimbal_Joint
  Joint successfully assembled
  Rx
    Free position value: +0 (deg)
    High priority velocity target +89.4072 (deg/s) achieved
  Ry
    Free position value: +0 (deg)
    High priority velocity target +28.0298 (deg/s) achieved
  Rz
    Free position value: +0 (deg)
    High priority velocity target +25.9604 (deg/s) achieved
Cartesian_Joint
  Joint successfully assembled
  Px
    Free position value: +0 (m)
    High priority velocity target -0.25679 (cm/s) achieved
  Py
    Free position value: +0 (m)
    High priority velocity target -0.640975 (cm/s) achieved
  Pz
    Free position value: +0 (m)
    High priority velocity target +0.594411 (cm/s) achieved
Planar_Joint
  Joint successfully assembled
  Px
    Free position value: +0 (m)
    High priority velocity target +0.433686 (cm/s) achieved
  Py
    Free position value: +0 (m)
    High priority velocity target +0.215342 (cm/s) achieved
  Rz
    Free position value: +0 (deg)
    High priority velocity target -60.8224 (deg/s) achieved
Bearing_Joint
  Joint successfully assembled
  Pz
    Free position value: +0 (m)
    High priority velocity target +0.908122 (cm/s) achieved
  Rx
    Free position value: +0 (deg)
```

```

    High priority velocity target +2.55215 (deg/s) achieved
  Ry
    Free position value: +0 (deg)
    High priority velocity target -63.2583 (deg/s) achieved
  Rz
    Free position value: +0 (deg)
    High priority velocity target -63.7585 (deg/s) achieved
Telescoping_Joint
  Joint successfully assembled
  S
    Free position value: [+0 +0 +0], +0 (deg)
    High priority velocity target [-89.3162 -43.8057 -52.8736] (deg/s) achieved
  Pz
    Free position value: +0 (m)
    High priority velocity target -0.844793 (cm/s) achieved
Weld_Joint
  Joint successfully assembled
Bushings_Joint
  Joint successfully assembled
  Px
    Free position value: +0 (m)
    High priority velocity target +0.66129 (cm/s) achieved
  Py
    Free position value: +0 (m)
    High priority velocity target +0.148752 (cm/s) achieved
  Pz
    Free position value: +0 (m)
    High priority velocity target -0.792454 (cm/s) achieved
  Rx
    Free position value: +0 (deg)
    High priority velocity target -44.7294 (deg/s) achieved
  Ry
    Free position value: +0 (deg)
    High priority velocity target -32.5625 (deg/s) achieved
  Rz
    Free position value: +0 (deg)
    High priority velocity target -71.8616 (deg/s) achieved
SixDof_Joint
  Joint successfully assembled
  Px
    Free position value: +0 (m)
    High priority velocity target +0.664527 (cm/s) achieved
  Py
    Free position value: +0 (m)
    High priority velocity target -0.294531 (cm/s) achieved
  Pz
    Free position value: +0 (m)
    High priority velocity target -0.806505 (cm/s) achieved
  S
    Free position value: [+0 +0 +0], +0 (deg)
    High priority velocity target [-24.8307 -63.8315 -74.1798] (deg/s) achieved

```

Finally, to perform any simulation workflows we can generate the simulink model using the **makeBlockDiagram** method.

```
makeBlockDiagram(mb,op,'jointZooModel');
```

Warning: Unrecognized function or variable 'registerTICCS'.

Warning: Unrecognized function or variable 'customizationticcs'.

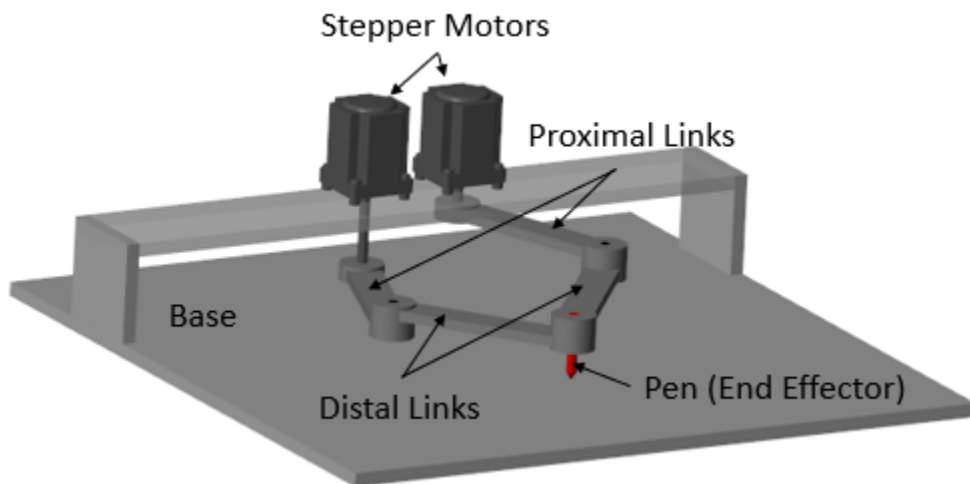
See Also

`simscape.multibody.Multibody` | `simscape.multibody.RigidBody` |
`simscape.multibody.Joint`

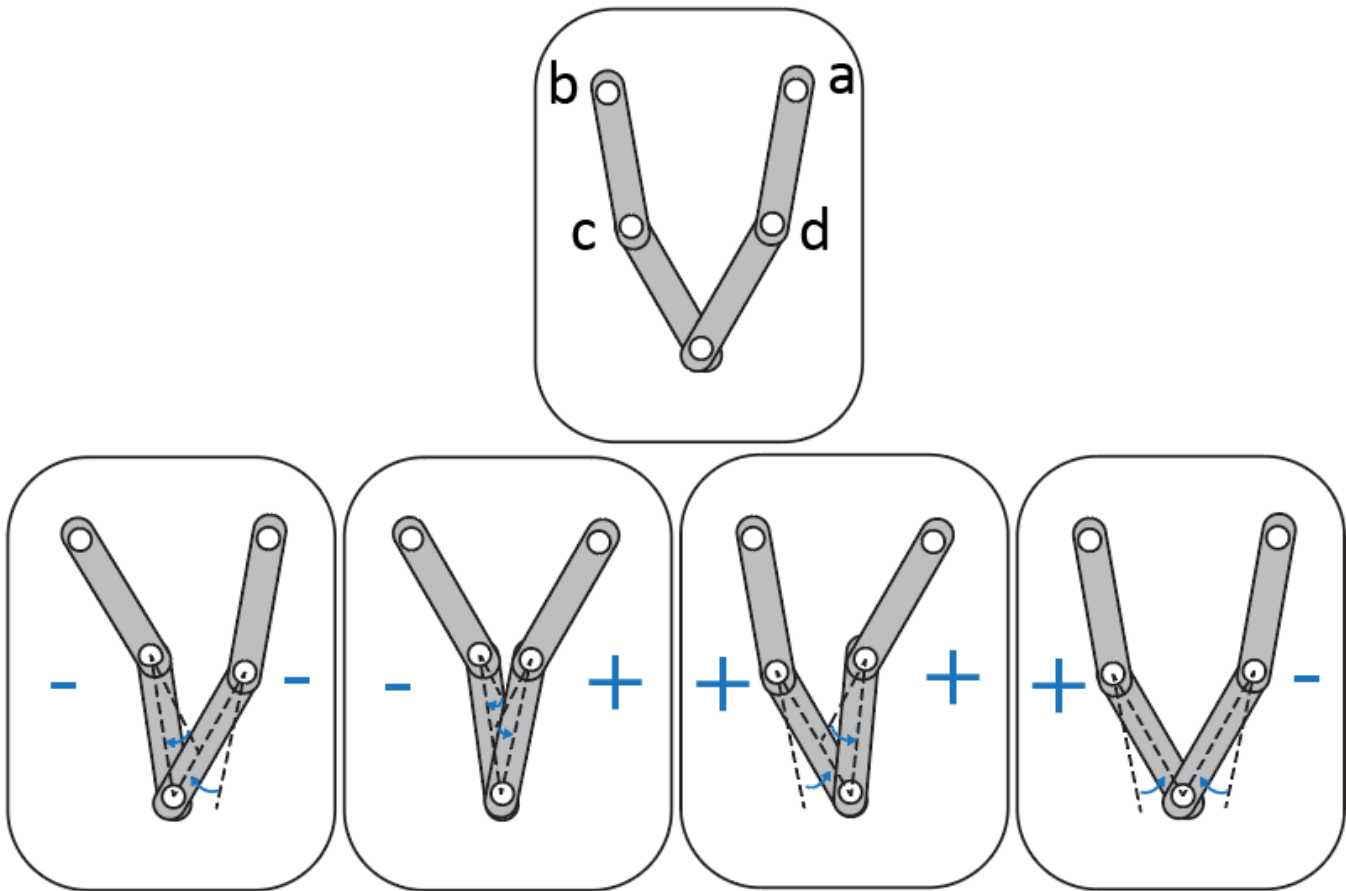
Perform Forward and Inverse Kinematics on a Five-Bar Robot

This example shows how to use the `KinematicsSolver` object to perform forward kinematics (FK) and inverse kinematics (IK) on a five-bar robotic mechanism. First, the example demonstrates how to perform FK analyses to calculate a singularity-free workspace for a five-bar robot. Then the example shows how to perform IK analyses to compute the motor angles that correspond to an end-effector trajectory within that workspace.

A five-bar robot, which is also called a five-bar planar manipulator, is a two-degree-of-freedom parallel mechanism. In this example, the robot has four equal-length links that cannot collide with each other. The robot uses two stepper motors to drive the proximal links to move the pen.



The five-bar robot has four configuration space regimes: $--$, $-+$, $++$, and $+ -$. As shown in the image, each sign indicates the sign of the offset angle between the corresponding proximal and distal links. The active revolute joints a and b are directly controlled by the motors, and the passive revolute joints c and d connect the proximal and distal links.



Configurations in which one or both of the arms become fully extended cause the end effector to lose at least one of its degrees of freedom. These configurations correspond to kinematic singularities and occur when the robot transitions from one regime to another. To avoid the kinematic singularities in the FK analysis, this example only studies the five-bar robot workspace in the +- regime.

Run Forward Kinematics on the Five-Bar Robot

This section demonstrates how to perform an FK analysis to compute a singularity-free workspace for the pen that corresponds to a set of motor angles. Specifically, the example assigns the active joint angles of the robot as target variables and the x and y coordinates of the pen as output variables. The example also assigns the passive joints as initial guess variables, which are used to help bias the solutions towards the +- regime.

1. To perform the FK analysis, first load the five-bar robot model into memory and create a `KinematicsSolver` object (`fk`) for the model.

```
mdl = 'sm_five_bar_robot';
load_system(mdl);
fk = Simscape.Multibody.KinematicsSolver(mdl);
```

2. To specify and study the relationships between the pen, world frame, active joints, and passive joints of the robot, you need to create frame variables that correspond to the position of the pen with respect to the world frame. Note that you don't need to create joint variables because they are created automatically when the object is constructed.

Add a group of three translational frame variables for the pen to fk. Specify the world frame as the base and the **Tip** frame of the pen as the follower. Name the frame variable group Pen.

```
base = mdl + "/World Frame/W";
follower = mdl + "/Five Bar Robot/Pen/Pen/Tip";
addFrameVariables(fk, 'Pen', 'Translation', base, follower);
```

3. Assign the position-based joint variables of the active joints (j5.Rz.q and j4.Rz.q) as targets. To find the IDs of the position-based joint variables, use the jointPositionVariables object function.

```
jointPositionVariables(fk)
```

```
ans=5×4 table
```

ID	JointType	BlockPath
"j1.Rz.q"	"Revolute Joint"	"sm_five_bar_robot/Five Bar Robot/Passive_Joint_c"
"j2.Rz.q"	"Revolute Joint"	"sm_five_bar_robot/Five Bar Robot/Passive_Joint_d"
"j3.Rz.q"	"Revolute Joint"	"sm_five_bar_robot/Five Bar Robot/Passive_Joint_e"
"j4.Rz.q"	"Revolute Joint"	"sm_five_bar_robot/Five Bar Robot/Step_Motor_a/Active_Joint"
"j5.Rz.q"	"Revolute Joint"	"sm_five_bar_robot/Five Bar Robot/Step_Motor_b/Active_Joint"

```
targetIDs = ["j5.Rz.q"; "j4.Rz.q"];
addTargetVariables(fk, targetIDs);
```

4. Assign these variables as outputs:

- The translational x and y components of the pen (Pen.Translation.x and Pen.Translation.y)
- The joint variables of the passive joints (j1.Rz.q and j2.Rz.q)

To find the IDs of the frame variables, use the frameVariables object function.

```
frameVariables(fk)
```

```
ans=3×4 table
```

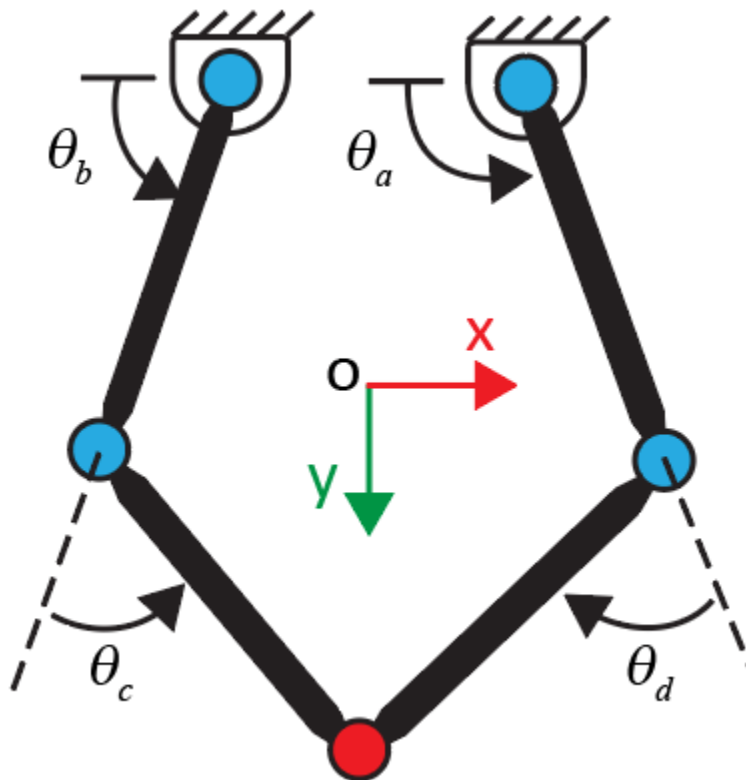
ID	Base	Follower
"Pen.Translation.x"	"sm_five_bar_robot/World Frame/W"	"sm_five_bar_robot/Five Bar Robot/Tip"
"Pen.Translation.y"	"sm_five_bar_robot/World Frame/W"	"sm_five_bar_robot/Five Bar Robot/Tip"
"Pen.Translation.z"	"sm_five_bar_robot/World Frame/W"	"sm_five_bar_robot/Five Bar Robot/Tip"

```
outputIDs = ["Pen.Translation.x"; "Pen.Translation.y"; "j1.Rz.q"; "j2.Rz.q"];
addOutputVariables(fk, outputIDs);
```

5. Assign the joint variables of the passive joints (j1.Rz.q and j2.Rz.q) as guesses.

```
guessIDs = ["j1.Rz.q"; "j2.Rz.q"];
addInitialGuessVariables(fk, guessIDs);
```

6. The image is a sketch of the five-bar robot. The angles of the active joints are θ_a and θ_b , and the passive joints are θ_c and θ_d . Use a nested for loop to compute the workspace of the pen for a set of active joint angles that vary between [95, 150] and [30, 85] degrees, respectively. To ensure that the robot is in the +- regime, use [50 -50] degrees as the initial values for the guess variables (j1.Rz.q and j2.Rz.q) and then use the latest values of the guess variables to guide the next FK solution.



```

guesses = [50 -50]; % Initial guesses for theta_c and theta_d
index = 1;

% Generate angles for theta_a and theta_b
theta_a = linspace(95,150,31);
theta_b = linspace(30,85,31);

% Allocate memory for output data
ik_data = zeros(length(theta_a)*length(theta_b),length(outputIDs));

% Calculate the robot's workspace for the given theta_a and theta_b
for i = 1:length(theta_a)
    for j = 1:length(theta_b)
        targets = [theta_b(j),theta_a(i)];
        [outputVec,statusFlag] = solve(fk,targets,guesses);
        if statusFlag == 1 % Save a solution if its statusFlag is 1
            ik_data(index,1:length(outputIDs)) = outputVec;
            index = index + 1;
            guesses = [outputVec(3),outputVec(4)]; % Update next guess with last solution
        end
    end
end
end

```

7. Plot the angles of the passive joints to verify if the five-bar robot is in the +- regime. If the robot is in the +- regime, the values of θ_c and θ_d are consistently positive and negative, respectively.

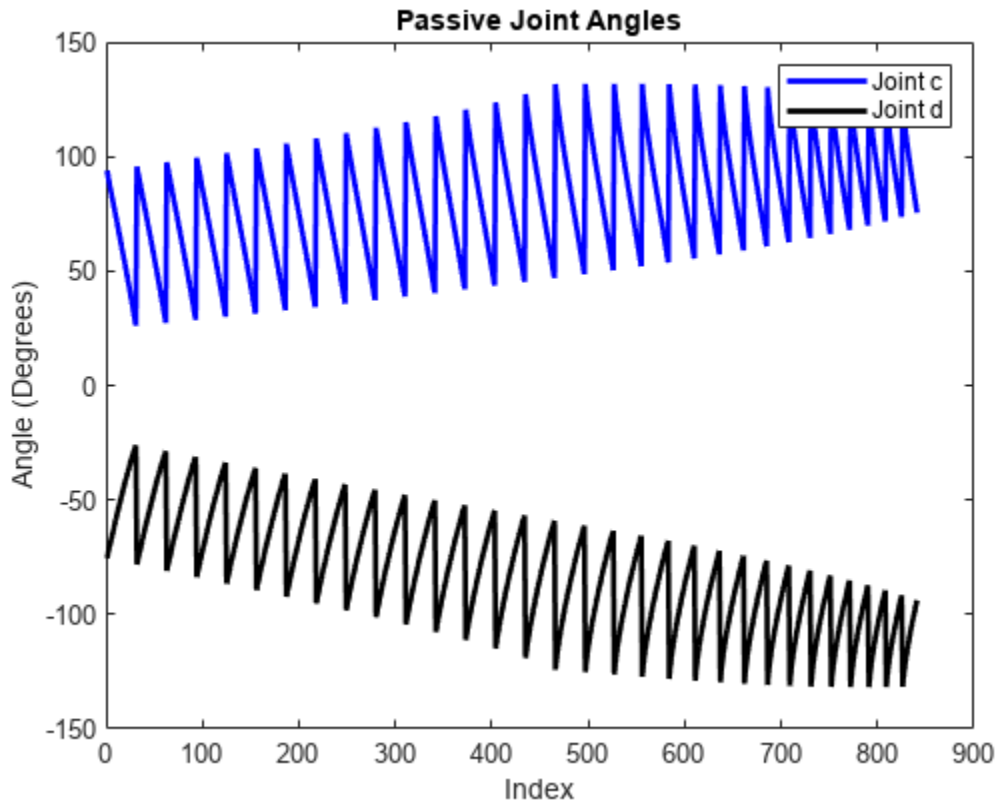
```

figure
plot(ik_data(1:index-1,3),'b','LineWidth',2); % Note that the total number of the solutions that
hold on

```

```
plot(ik_data(1:index-1,4),'k','LineWidth',2);
hold off
```

```
title('Passive Joint Angles')
xlabel('Index')
ylabel('Angle (Degrees)')
legend('Joint c','Joint d')
```



The plot shows that the five-bar robot is in the \pm regime. Moreover, the computed workspace is singularity-free because the values of θ_c and θ_d are away from zero.

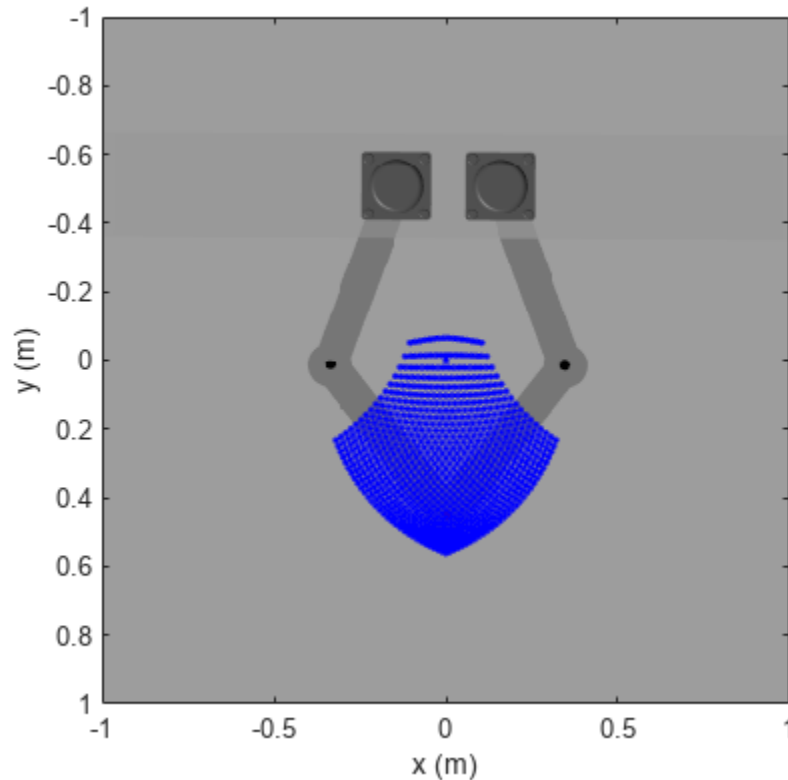
8. Plot the workspace data on the top-view image of the five-bar robot.

```
% Plot the top-view image of the five-bar robot
figure
x = [-1 1];
y = [1 -1];
C = imread('top_view_five_bar_robot.png');
image(x,y,C)

hold on

% Plot the calculated workspace as blue dots
plot(ik_data(:,1),ik_data(:,2),'b.')
ylabel('y (m)')
xlabel('x (m)')
```

```
axis equal
axis([-1 1 -1 1]);
hold off
```



Run Inverse Kinematics on the Five-Bar Robot

In this section, the example shows how to perform an IK analysis to compute the motor angles that correspond to an end-effector trajectory. Specifically, the example assigns the translational x and y components of the pen as target variables and assigns the motor angles as output variables. Then the example computes the motor angles for a trajectory of the pen. To ensure the robot is in the +- regime, use the active and passive joint angles as guesses to bias the IK solutions. Note that you need to run the FK section before running this section.

1. Define a circular end-effector trajectory within the previously calculated workspace. Save the coordinates of the circle for the IK analyses.

```
figure
% Plot the calculated workspace as blue dots on the top-view image of the five-bar robot
image(x,y,C)
hold on
plot(ik_data(:,1),ik_data(:,2),'b.')
```

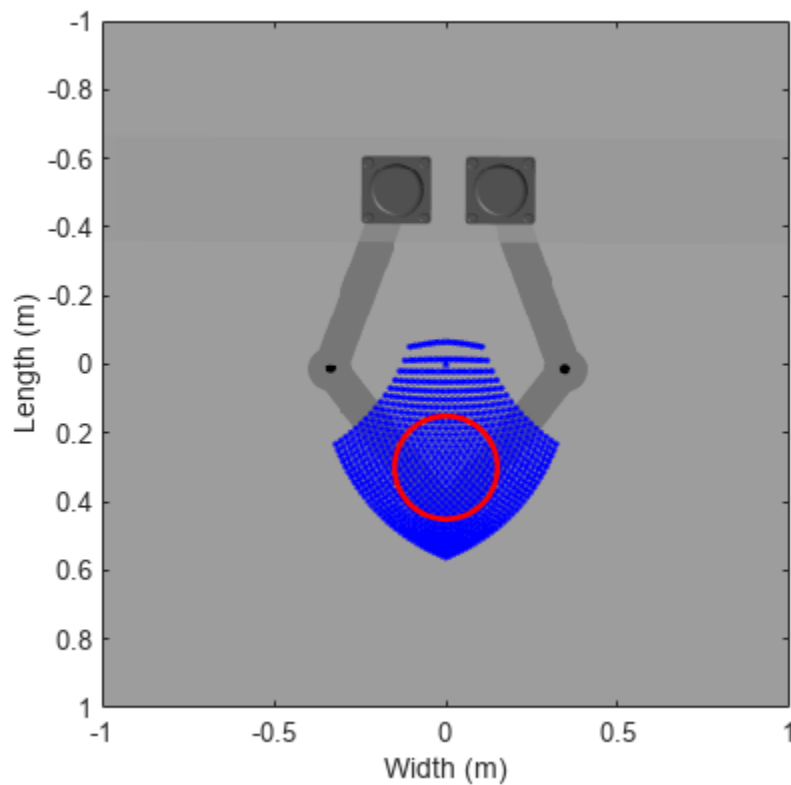
```
% Specify the center and radius of the circle
center_x = 0; % m
center_y = 0.3; % m
radius = 0.15; % m
th = 0:pi/50:2*pi; % radians
```

```

% Save the x and y coordinates of the circle for the IK analyses
coordinates_x = radius * cos(th) + center_x;
coordinates_y = radius * sin(th) + center_y;

% Plot the circle
plot(coordinates_x,coordinates_y,'r','LineWidth',2)
axis equal
axis([-1 1 -1 1]);
ylabel('Length (m)')
xlabel('Width (m)')
hold off

```



2. To perform the IK analysis, first create a KinematicsSolver object (ik) for the model.

```
ik = simscape.multibody.KinematicsSolver mdl;
```

3. Create frame variables for the position of the pen with respect to the world frame and add them to the object (ik). You can reuse the base and follower variables from the FK section.

```
addFrameVariables(ik, 'Pen','Translation', base, follower);
```

4. Assign the variables that correspond to the translational x and y components of the pen (Pen.Translation.x and Pen.Translation.y) as targets. To find the IDs of the frame variables, use the frameVariables object function.

```
frameVariables(ik)
```

```
ans=3x4 table
      ID                               Base                               Follower
-----
"Pen.Translation.x" "sm_five_bar_robot/World Frame/W" "sm_five_bar_robot/Five Bar Robot"
"Pen.Translation.y" "sm_five_bar_robot/World Frame/W" "sm_five_bar_robot/Five Bar Robot"
"Pen.Translation.z" "sm_five_bar_robot/World Frame/W" "sm_five_bar_robot/Five Bar Robot"
```

```
targetIds = ["Pen.Translation.x";"Pen.Translation.y"];
addTargetVariables(ik,targetIds);
```

5. Assign these variables as outputs:

- The position-based joint variables of the active joints (j5.Rz.q and j4.Rz.q)
- The position-based joint variables of the passive joints (j1.Rz.q and j2.Rz.q)

To find the IDs of the position-based joint variables, use the `jointPositionVariables` object function.

```
jointPositionVariables(ik)
```

```
ans=5x4 table
      ID                               JointType                               BlockPath
-----
"j1.Rz.q" "Revolute Joint" "sm_five_bar_robot/Five Bar Robot/Passive_Joint_c"
"j2.Rz.q" "Revolute Joint" "sm_five_bar_robot/Five Bar Robot/Passive_Joint_d"
"j3.Rz.q" "Revolute Joint" "sm_five_bar_robot/Five Bar Robot/Passive_Joint_e"
"j4.Rz.q" "Revolute Joint" "sm_five_bar_robot/Five Bar Robot/Step_Motor_a/Active_Joint"
"j5.Rz.q" "Revolute Joint" "sm_five_bar_robot/Five Bar Robot/Step_Motor_b/Active_Joint"
```

```
outputIds = ["j5.Rz.q";"j4.Rz.q";"j1.Rz.q";"j2.Rz.q"];
addOutputVariables(ik,outputIds);
```

6. Use a for loop to implement the IK analysis. Specifically, use the for loop to assign the coordinates of the predefined trajectory to the pen and calculate the corresponding motor angles.

```
N = length(coordinates_x); % Number of points on trajectory
index_in_ik = 1;
ik_data = zeros(N,length(outputIds)); % Allocate memory for the variable

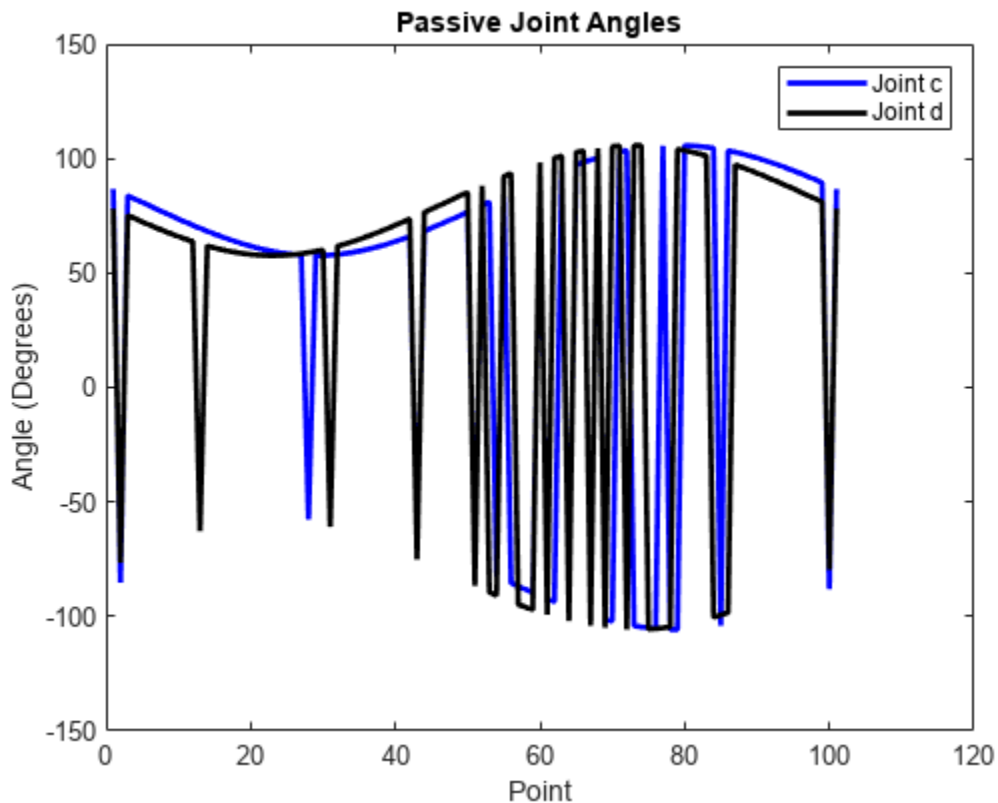
for ind = 1:N
    targets = [coordinates_x(ind), coordinates_y(ind)]; % Use the x and y coordinates of the pre
    [outputVec_ik,statusFlag_ik] = solve(ik,targets);
    if statusFlag_ik == 1
        viewSolution(ik) % View the solution in the Kinematics Solver Viewer
        ik_data(index_in_ik,1:length(outputIds)) = outputVec_ik; % save the output data
        index_in_ik = index_in_ik+1;
    else
        error('Did not hit targets');
    end
end
closeViewer(ik);
```

7. Plot the passive joints of the robot to see if the robot is in the +- regime.

```

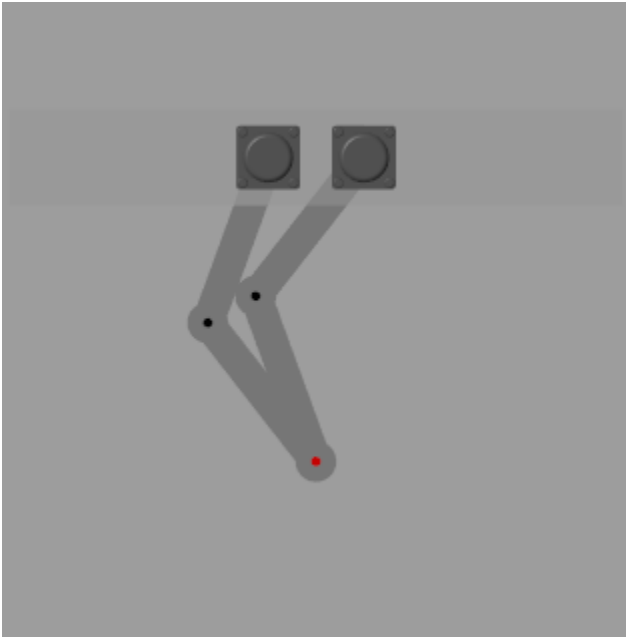
% Plot the angles of the passive joints
figure
plot(ik_data(:,3),'b','LineWidth',2);
hold on
plot(ik_data(:,4),'k','LineWidth',2);
hold off
title('Passive Joint Angles')
xlabel('Point')
ylabel('Angle (Degrees)')
legend('Joint c','Joint d')

```



The plot shows that the robot is not always in the +- regime.

During the IK analysis, the Kinematics Solver Viewer opens and shows an animation of the process. The animation also shows that the result is not desired because the five-bar robot is not always in the +- regime. The image shows a snapshot of the animation in the ++ regime.



8. To help ensure that all solutions stay in the \pm regime, assign the joint variables of the passive ($j1.Rz.q$ and $j2.Rz.q$) and active ($j5.Rz.q$ and $j4.Rz.q$) joints as guesses to bias the IK solutions.

```
guessIds = ["j1.Rz.q";"j2.Rz.q";"j5.Rz.q";"j4.Rz.q"];
addInitialGuessVariables(ik,guessIds);
```

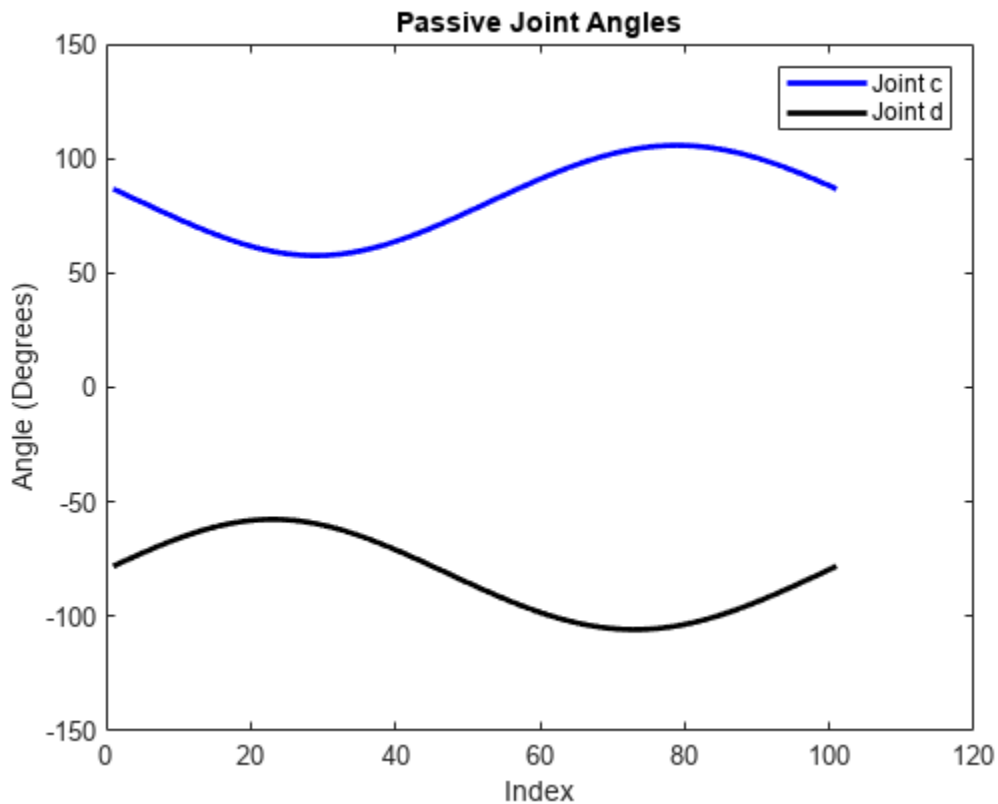
Use [50 -50 45 135] degrees as initial values for the guess variables and use the latest values of the guess variables to guide the next IK solution.

```
guesses = [50 -50 45 135]; % Assign initial guesses for passive and active joints
N = length(coordinates_x);
index_in_ik = 1;
ik_data = zeros(N,length(outputIds)); % Allocate memory for the variable
for ind = 1:N
    targets = [coordinates_x(ind), coordinates_y(ind)]; % Use the x and y coordinates of the previous point
    [outputVec_ik,statusFlag_ik] = solve(ik,targets, guesses);
    if statusFlag_ik == 1
        viewSolution(ik) % View the solution in the Kinematics Solver Viewer
        ik_data(index_in_ik,1:length(outputIds)) = outputVec_ik; % Save the rotations of the motor angles
        index_in_ik=index_in_ik+1;
        guesses = [outputVec_ik(3) outputVec_ik(4) outputVec_ik(1) outputVec_ik(2)]; % Update next guess
    else
        error('Did not hit targets');
    end
end
closeViewer(ik);
```

The plot indicates that the passive joint angles are always well away from zero and are always in the \pm regime. Therefore, this set of motor angles corresponds to the end effector moving in a smooth motion along the circular trajectory without passing through any kinematic singularities.

```
% Plot the angles of the passive joints
figure
```

```
plot(ik_data(:,3), 'b', 'LineWidth', 2);  
hold on  
plot(ik_data(:,4), 'k', 'LineWidth', 2);  
hold off  
title('Passive Joint Angles')  
xlabel('Index')  
ylabel('Angle (Degrees)')  
legend('Joint c', 'Joint d')
```



See Also

[KinematicsSolver](#) | [frameVariables](#) | [initialGuessVariables](#) | [jointPositionVariables](#) | [outputVariables](#) | [targetVariables](#) | [addFrameVariables](#) | [addInitialGuessVariables](#) | [addOutputVariables](#) | [addTargetVariables](#) | [solve](#) | [viewSolution](#) | [closeViewer](#)

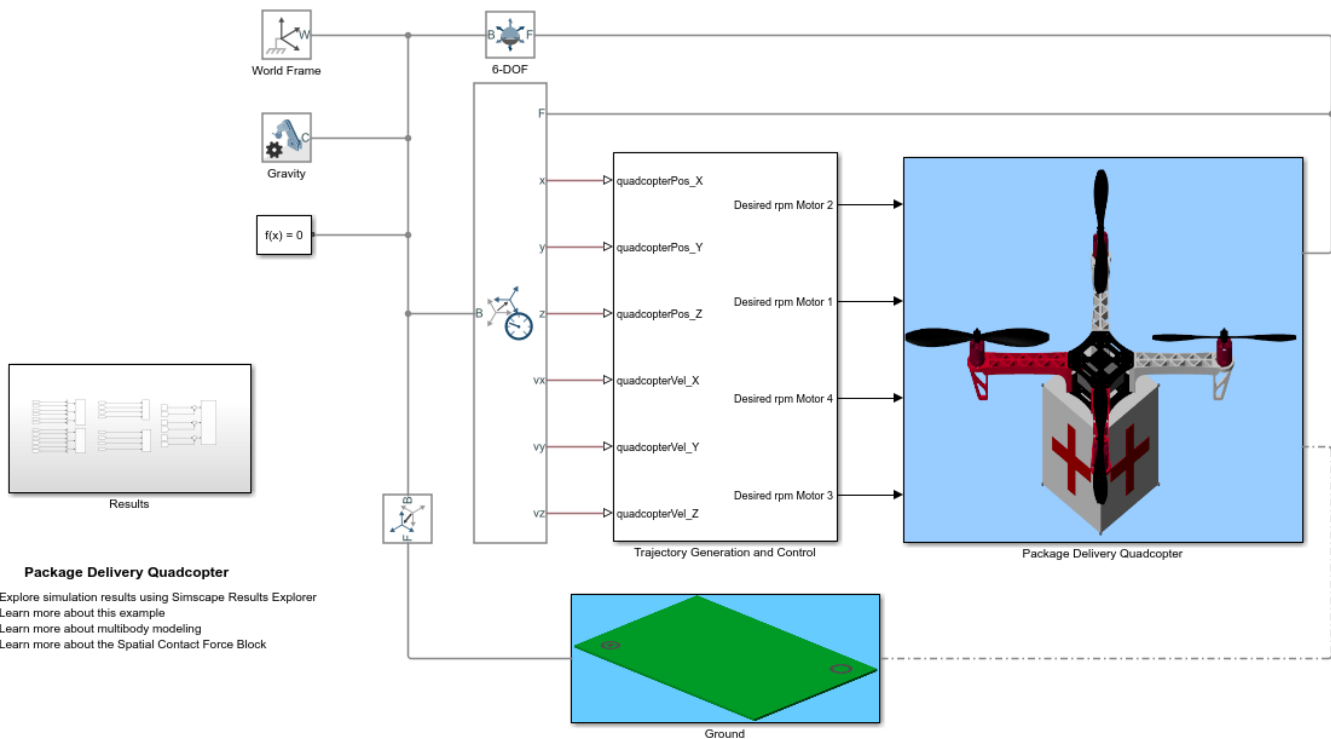
More About

- “Pick and Place Robot Using Forward and Inverse Kinematics” on page 8-75

Package Delivery Quadcopter

This example models a package delivery quadcopter. The quadcopter takes off from the launchpad and delivers the package to the drop-off location while following a desired trajectory.

Model



Package Delivery Quadcopter Subsystem

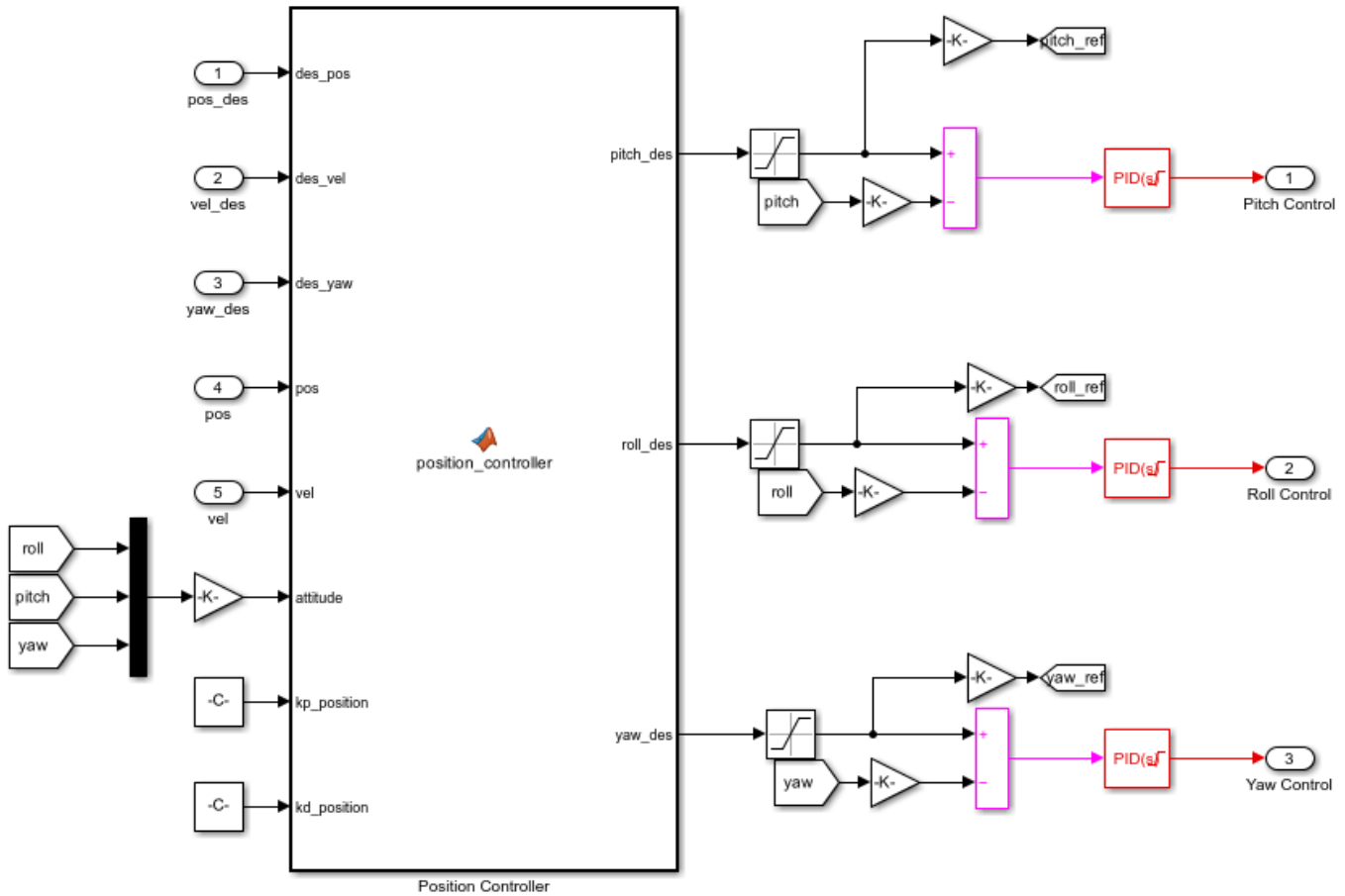
The quadcopter comprises of a chassis and four motor-propeller pairs, each spinning alternately in clockwise and counterclockwise direction. The quadcopter chassis has four identical arms rigidly attached to the top and bottom plates using the Rigid Transform block. The motor comprises of a motor base, a motor shaft which is connected to the motor base via revolute joint and a motor cap. The propeller is rigidly attached to the motor shaft. The motor base is rigidly connected to the arm using the Rigid Transform Block. Based on the rotational speed of the propeller, the thrust coefficient and thrust generated by each propeller are calculated. The package is attached to the quadcopter via weld joint. Whenever the quadcopter carrying the package gets in the proximity of the drop-off location, the MATLAB Function block Package Delivery Quadcopter/Package Release Trigger disengages the weld joint to deliver the package. The Spatial Contact Force blocks are used to model the contact forces between the package and the ground surface. See the block mask for more information.

Open Package Delivery Quadcopter Subsystem

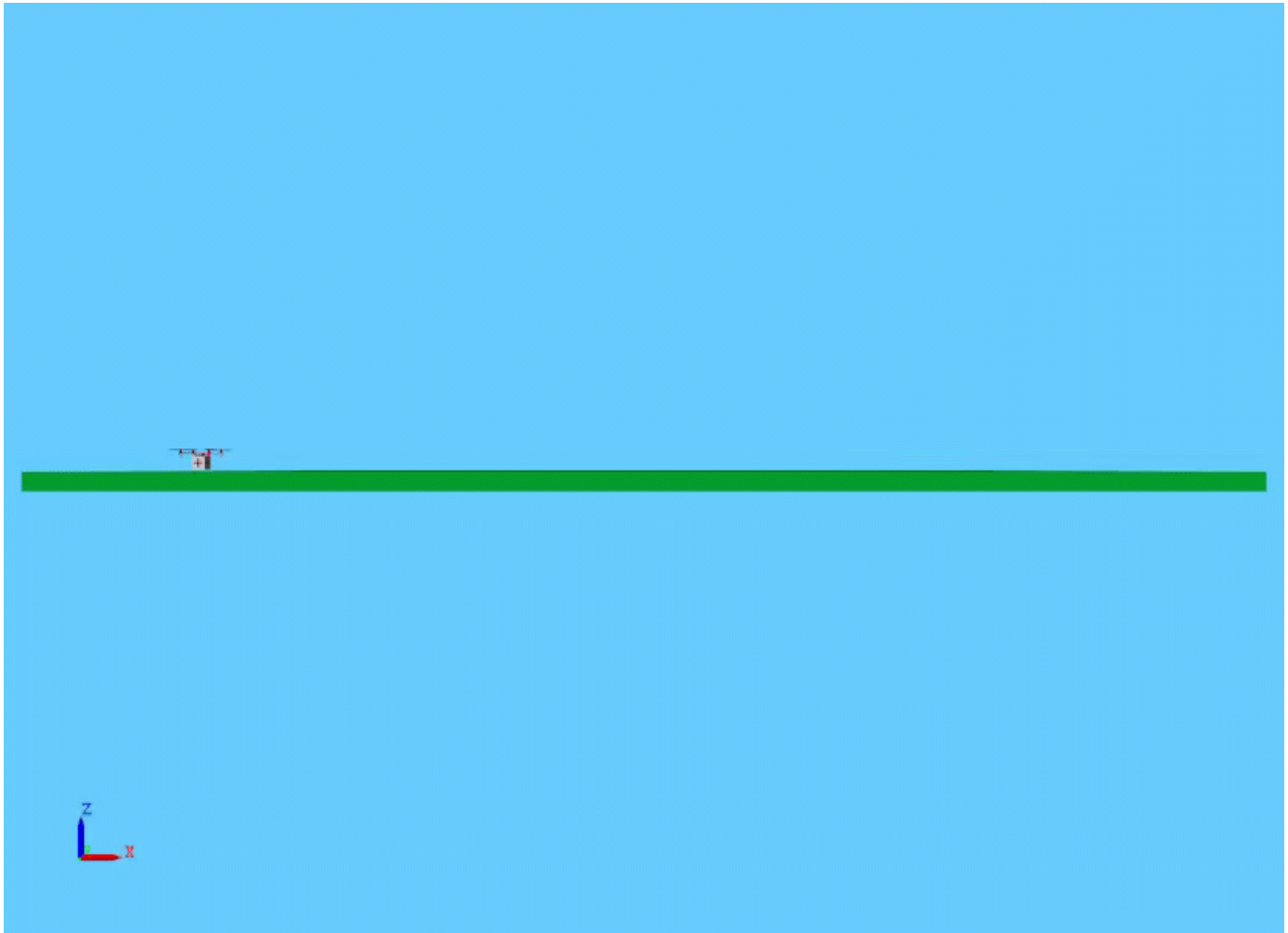
Position and Attitude Control Subsystem

A cascade control is used to drive the position of the quadcopter to the desired state. The inner loop contains a PID controller which controls the attitude dynamics of the quadcopter. A backstepping controller inside MATLAB Function block Position and Attitude Controller/Position Controller computes the desired roll and pitch angles necessary to achieve the control objective. See the block mask for more information.

Open Position and Attitude Control Subsystem

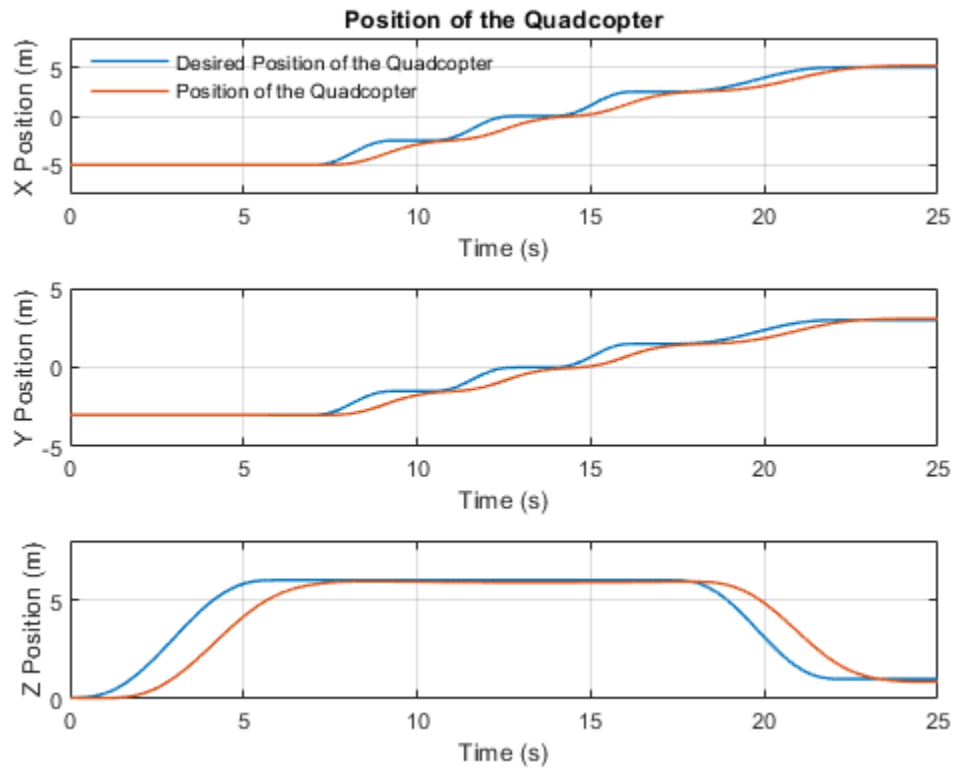


Mechanics Explorer Animation



Simulation Results from Simscape Logging

The plot below shows the actual and desired position of the Quadcopter



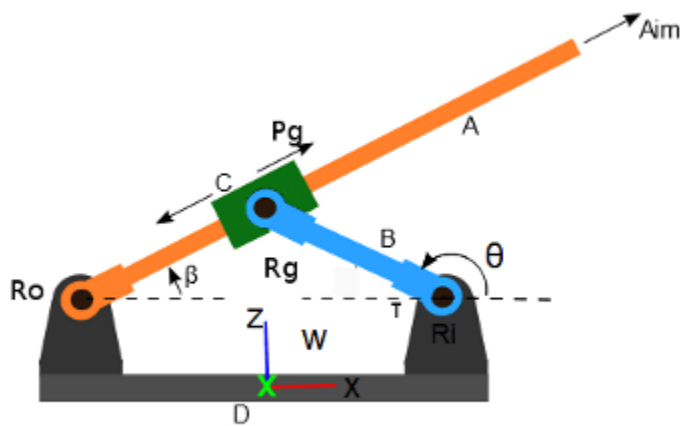
References : [1] Das, A., Lewis, F. and Subbarao, K., 2009. Backstepping approach for controlling a quadrotor using lagrange form dynamics. *Journal of Intelligent and Robotic Systems*, 56(1), pp.127-151.

How to Build a Multibody System in MATLAB

This example highlights key concepts and recommended steps for building a multibody system in MATLAB. A simple design problem has been chosen to serve this purpose. The following section describes the design problem and subsequent sections discuss how to solve it.

Problem Description

The following figure shows a mechanism which functions as an aiming system.



The problem is simplified to aiming within the plane of the mechanism. The figure shows the schematic sketch of the mechanism and only captures the essentials of how the mechanism operates (which is usually the case during the early stages of a design process). The link **C** can slide on the link **A**. A motor applies torque τ at the revolute joint **Ri** and the task is to track a particular trajectory of the revolute angle β .

Building the Mechanism

A key principle to follow is to begin with a simple approximation to get the basic mechanism working and then in subsequent iterations add complexity to the model. The recommended model building process can be broken down into the following steps:

- 1 Identify the rigid bodies in the mechanism.
- 2 Identify how the rigid bodies are connected to each other (joints, constraints etc).
- 3 Consider each rigid body in isolation. Build a simple approximation of the rigid body, and define the frames rigidly attached to it.
- 4 Assemble the rigid bodies by connecting them to joints and/or constraints
- 5 Identify any issues with the model assembly.
- 6 Visualize the assembled multibody to identify and fix other issues with the system.
- 7 Utilize the **OperatingPoint** to guide assembly to desired configuration.
- 8 Create a block diagram model from the **Multibody** object to simulate the system.

- 9 Add details to the individual rigid bodies to make the model a more accurate representation of the actual mechanism.

The following sections describe these steps in more detail.

Identifying Rigid Bodies and Joints

The mechanism has four rigid bodies

- **Rigid Body A** (orange)
- **Rigid Body B** (blue)
- **Rigid Body C** (green)
- **Rigid Body D** (grey)

The mechanism has the following joints

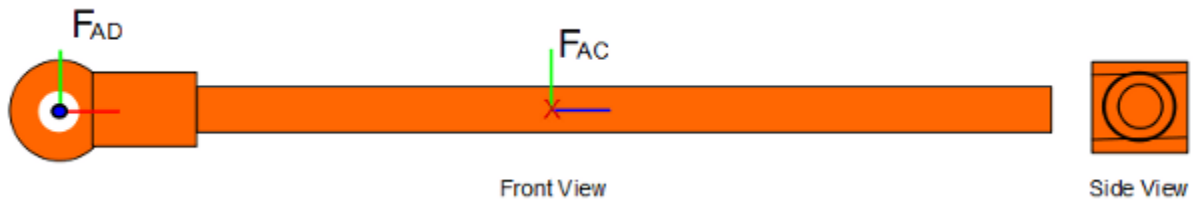
- Rigid bodies A and D are connected via a revolute joint **Ro**.
- Rigid bodies A and C are connected via a prismatic joint **Pg**.
- Rigid bodies C and B are connected via a revolute joint **Rg**.
- Rigid bodies B and D are connected via a revolute joint **Ri**.

In addition, the rigid body **D** is rigidly connected to the world frame **W** since it is motionless.

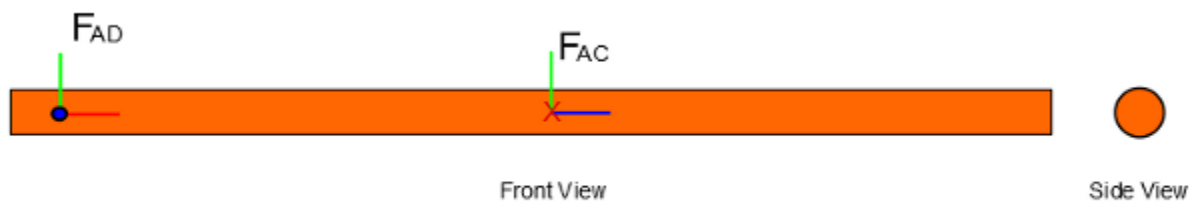
Defining the Rigid Bodies and their Interface

You define a rigid body by specifying its geometry, mass properties and interface with other parts. Each rigid body is identified and defined in isolation. In the above example, the mechanism is composed of four rigid bodies: **A**, **B**, **C** and **D**.

The rigid body **A** is shown in isolation below.



Once the shape of the object is defined and its density is specified, Simscape Multibody can compute the inertia automatically. Instead of defining the fairly complicated shape shown above, as a first approximation, you can define the shape of the rigid body as a simple cylinder.



The above shown rigid body can be defined in MATLAB using a high-level function like **dcrankaim_approx_body_A** which makes use of objects like **simscape.multibody.RigidBody**, **simscape.multibody.Solid** etc. This function takes input arguments like length, radius, density and color and returns a rigidbody object containing a cylindrical solid with one frame at the end of the link and the other at the center.

```
radius = simscape.Value(2, 'cm');
length = simscape.Value(1, 'm');
color = [0 0 1];
density = simscape.Value(2700, 'kg/m^3') ;
bodyA = dcrankaim_approx_body_A(length, radius, density, color);
```

To understand the use of these different objects, let us take a look at the implementation of this function. In the **dcrankaim_approx_body_A**, we first import the packages needed to use Simscape Value, OperatingPoint and Simscape Multibody MATLAB classes.

```
import simscape.Value simscape.op.* simscape.multibody.*;
```

We then create an object of the class **RigidBody**. The **RigidBody** class is a container which can comprise of solids, inertias, graphics, frames (which act as interface to connect it to the other parts of Multibody system) and other rigid body objects.

```
link = RigidBody;
```

We then define and add a cylindrical solid to approximate the shape of **BodyA**. This is created using the cylindrical geometry, density, and its visual properties.

```
cylinder = Solid(Cylinder(radius, length),...
                UniformDensity(density),...
                SimpleVisualProperties(color));
addComponent(link, 'Link', 'reference', cylinder);
```

Once we have defined the shape (first approximation) of the rigid body **A** and specified its density. We now has enough information to compute the inertial properties of the solid. You can define a simply shaped geometry in MATLAB using the following geometry classes.

- `simscape.multibody.Brick`
- `simscape.multibody.Sphere`
- `simscape.multibody.Cylinder`
- `simscape.multibody.Ellipsoid`
- `simscape.multibody.RegularExtrusion`
- `simscape.multibody.GeneralExtrusion`
- `simscape.multibody.Revolution`

These geometry objects are contained in the **Solid** class. Solids are characterized by geometry, inertia and visual properties and by using it with the above geometry objects, it allows us to create simple solids of different shapes. Solids are always rigid and always have an implicit reference frame, and the geometry and inertia are specified relative to this frame.

The interface of a rigid body is established by defining frames attached to the rigid body. A rigid body is connected to other parts of the mechanism via the rigidly attached frames. In Simscape Multibody, joints establish a time-varying relationship between two frames. For instance, the **Revolute Joint** establishes the relationship that the **Z**-axes of the attached frames are parallel and the origins of the

frames are coincident. The **Prismatic Joint** establishes the relationship that the **Z**-axes of the attached frames are collinear and the **X** and **Y** axes are always parallel. Note that the frames themselves are defined independently of the joint; the joint only establishes a relationship between the already existing frames. Note also that the **Z**-axis is the axis of rotation in the case of the revolute joint and is the axis of sliding in the case of the prismatic joint. This information is essential when we define the interface of a rigid body by defining the frames rigidly attached to it.

In this example, the rigid body **A** has a cylindrical hole at one end that fits onto a peg so that **A** can rotate about the axis of the cylindrical hole. This suggests that a frame should be defined at the hole center with its **Z**-axis aligned with the axis of the hole (the axis of rotation). This frame is labelled as F_{AD} above. The choice of orientation of the **X** and **Y** axes of F_{AD} partly determines the zero configuration of the joint to which F_{AD} would be connected (see discussion on Zero Configuration below). **A** also acts as the shaft on which part **C** slides. This suggests that a frame should be defined at the center of **A** (an arbitrarily selected position) with its **Z**-axis aligned along the length of **A** (along the direction of sliding). This frame is labelled as F_{AC} above. The frames F_{AD} and F_{AC} define the interface for the rigid body **A**.

The following code shows how the frames are defined and added to the rigidbody **A** in the function **dcrankaim_approx_body_A**.

```
rt_Fad = RigidTransform( ...
    StandardAxisRotation(Value(90, 'deg'), Axis.PosY), ...
    StandardAxisTranslation(length/2, Axis.PosZ));

rt_Fac = RigidTransform( ...
    StandardAxisRotation(Value(180, 'deg'), Axis.PosY));

addFrame(link, 'Fad', 'reference', rt_Fad);
addConnector(link, 'Fad');

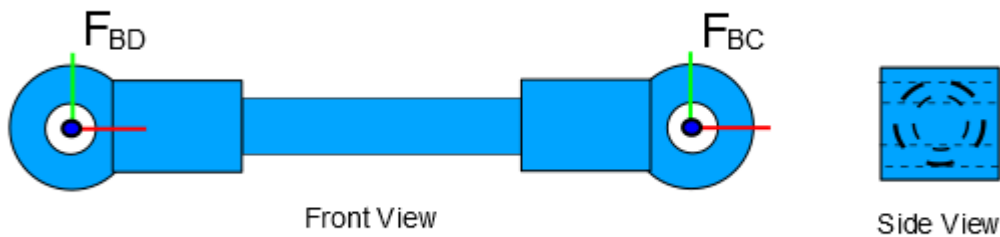
addFrame(link, 'Fac', 'reference', rt_Fac);
addConnector(link, 'Fac');
```

The frames F_{AC} and F_{AD} are defined with respect to the reference frame of the rigidbody.

To visualize this rigid body run the following code.

```
mb = Multibody;
addComponent(mb, 'BodyA', bodyA);
cmb = compile(mb);
visualize(cmb, computeState(cmb, OperatingPoint), 'BodyVizA');
```

Now let us consider the rigid body **B**. The shape of the rigid body can again be approximated with a simple cylinder. The rigid body has cylindrical holes at both ends that fit onto pegs. The rigid body **B** can rotate about either hole axis. This suggests that two frames should be defined: one at each hole center with its **Z**-axis aligned with the axis of the hole.



The function `dcrankaim_approx_body_B` shows how the **RigidBody**, **Solid** and **RigidTransform** objects have been used to define the shape, inertia and interface of the rigid body **B**.

```
radius = simscape.Value(2, 'cm');
length = simscape.Value(1, 'm');
color = [0 0 1];
density = simscape.Value(2700, 'kg/m^3') ;

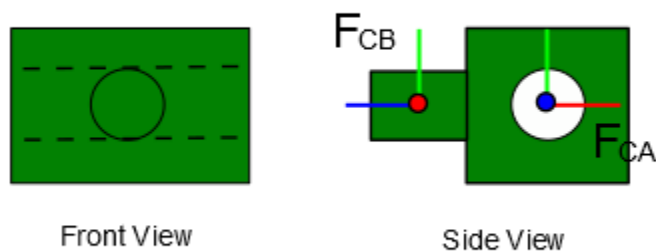
bodyB = dcrankaim_approx_body_B(length, radius, density, color);
```

To visualize this rigid body run the following code.

```
mb = Multibody;
addComponent(mb, 'BodyB', bodyB);
cmb = compile(mb);
visualize(cmb, computeState(cmb, OperatingPoint), 'BodyVizB');
```

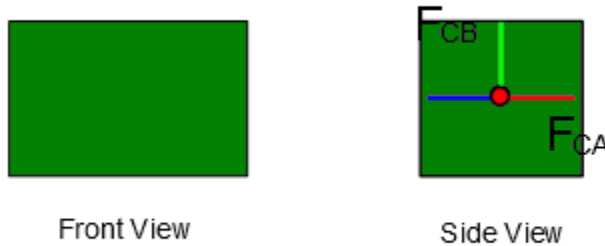
A similar approach can be taken for building a first approximation of the rigid body **D**.

Now let us consider the rigid body **C**.



This rigid body has a cylindrical hole that slides on a peg. It also has a peg about which another body can rotate. This suggests the need to define two frames: one at the center of the hole with its **Z**-axis along the axis of the hole, and the other at the center of the peg with its **Z**-axis along the axis of the peg. These are marked as F_{CA} and F_{CB} above.

The shape of the rigid body **C** can be approximated with a simple cuboid. In the first approximation of the rigid body, the offset between the origins of frames F_{CB} and F_{CA} can also be made zero. This results in the simplified representation of rigid body as shown below.



The function `dcrankaim_approx_body_C` shows how the **RigidBody**, **Solid** and **RigidTransform** objects have been used to define the shape, inertia and interface of the rigid body **C**.

```
brickdim = simscape.Value([10, 8, 8], 'cm');
density = simscape.Value(2700, 'kg/m^3');
color = [0 0 1];
bodyC = dcrankaim_approx_body_C(brickdim,density,color);
```

Assembling the Individual Bodies Using Joints

All the individual bodies were built in isolation. The process of assembly involves establishing relationships (using joints) between the frames attached to the rigid bodies. The following joints establish all of the necessary relationships between the frames to assemble the mechanism.

- A **Revolute Joint** between the frames F_{da} and F_{ad}
- A **Prismatic Joint** between the frames F_{ac} and F_{ca}
- A **Revolute Joint** between the frames F_{cb} and F_{bc}
- A **Revolute Joint** between the frames F_{bd} and F_{db}

The function `dcrank_aiming_mechanism_v1` shows how objects like **Multibody** and **Joints** are used to assemble individual rigid bodies. The function takes the length of each of the cylindrical links **BodyA**, **BodyC**, **BodyD** and the dimensions of the slider link **BodyB** as the input arguments and returns the multibody object as the output.

```
bodyA_l = Value(80, 'cm');
bodyB_l = Value(30, 'cm');
bodyC_dim = Value([10, 8, 8], 'cm');
bodyD_l = Value(40, 'cm');
dcrankAimMech_mb = dcrank_aiming_mechanism_v1(bodyA_l,bodyB_l,bodyC_dim,bodyD_l);
```

To understand how this function assembles the mechanism, let us look at some its implementation.

In `dcrank_aiming_mechanism_v1`, we first create a Multibody object, which acts as container for variety of other components like **RigidBodies**, **Solids**, **Joints**, **RigidTransforms**.

```
dcrankAimMech_mb = Multibody;
```

We then add the World frame to the Multibody by using its **addComponent** method.

```
addComponent(dcrankAimMech_mb, 'World', WorldFrame());
```

Then BodyA, BodyB, BodyC and BodyD are created using the RigidBody object, as discussed in the above sections.

```
% add body_a
bodyA_l = Value(80, 'cm');
bodyA_r = Simscape.Value(2, 'cm');
bodyA_color = [1 0.6 0];
bodyA_density = Simscape.Value(2700, 'kg/m^3') ;
body_a = dcrankaim_approx_body_A(bodyA_l, bodyA_r, bodyA_density, bodyA_color);

% add body_b
bodyB_l = Value(30, 'cm');
bodyB_r = Simscape.Value(2, 'cm');
bodyB_color = [0 0 1];
bodyB_density = Simscape.Value(2700, 'kg/m^3') ;
body_b = dcrankaim_approx_body_B(bodyB_l, bodyB_r, bodyB_density, bodyB_color);

% add body_c
bodyC_dim = Value([10, 8, 8], 'cm');
bodyC_color = [0 1 0];
bodyC_density = Simscape.Value(2700, 'kg/m^3') ;
body_c = dcrankaim_approx_body_C(bodyC_dim, bodyC_density, bodyC_color);

% add body_d
bodyD_l = Value(40, 'cm');
bodyD_r = Simscape.Value(2, 'cm');
bodyD_color = [0.2 0.2 0.2];
bodyD_density = Simscape.Value(2700, 'kg/m^3') ;
body_d = dcrankaim_approx_body_D(bodyD_l, bodyD_r, bodyD_density, bodyD_color);
```

Once all the 4 bodies are created, we add them to the Multibody container object using its **addComponent** method.

```
addComponent(dcrankAimMech_mb, 'body_a', body_a);
addComponent(dcrankAimMech_mb, 'body_b', body_b);
addComponent(dcrankAimMech_mb, 'body_c', body_c);
addComponent(dcrankAimMech_mb, 'body_d', body_d);
```

The mechanism has 4 joints - 3 revolute and 1 prismatic. We create and add them using the **RevoluteJoint** and **PrismaticJoint** objects.

```
rJoint = RevoluteJoint;
pJoint = PrismaticJoint;

addComponent(dcrankAimMech_mb, 'Ro', rJoint);
addComponent(dcrankAimMech_mb, 'Ri', rJoint);
addComponent(dcrankAimMech_mb, 'Rg', rJoint);
addComponent(dcrankAimMech_mb, 'P', pJoint);
```

Now, we have all the required components for the mechanism. However, we still need to connect these components. We add these connections using the **connect** and **connectVia** methods of the **Multibody** object by connecting the appropriate RigidBody frames to the Base and Follower frames of the appropriate Joints.

```

connect(dcrankAimMech_mb, 'World/W', 'body_d/Fdw');
connectVia(dcrankAimMech_mb, 'Ro', 'body_d/Fda', 'body_a/Fad');
connectVia(dcrankAimMech_mb, 'P', 'body_a/Fac', 'body_c/Fca');
connectVia(dcrankAimMech_mb, 'Rg', 'body_b/Fbc', 'body_c/Fcb');
connectVia(dcrankAimMech_mb, 'Ri', 'body_d/Fdb', 'body_b/Fbd');

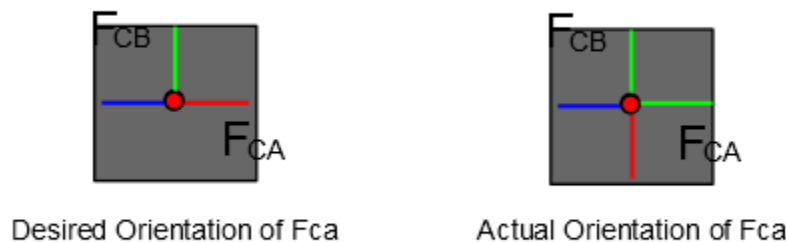
```

The effort that went into carefully defining the interfaces of all of the rigid bodies (i.e. the frames attached to them) made it very easy to complete the mechanism by simply adding and connecting the joints between the appropriate frames. However, at this point the resulting assembly may or may not be in the desired configuration since the mechanism can be assembled into multiple configurations. The function **dcrankaim_assembly_failure** shows the assembled mechanism.

```
dcrankAimMech_mb = dcrankaim_assembly_failure(bodyA_l, bodyB_l, bodyC_dim, bodyD_l);
```

Using computeState method to identify problems

In the function above, an intentional mistake has been made in the definition of the frame F_{CA} attached to rigid body C. This causes the assembly to fail. The figure below shows the desired and actual orientations of the frame F_{CA} .



The orientation of F_{CA} has to be corrected by a rotation of 90 deg about the **Z**-axis. To identify this issue, we first compile the multibody using the **compile** method of the **Multibody** object.

```
compiled_mb = compile(dcrankAimMech_mb);
```

Then we use the **computeState** method of the compiled multibody object for a default OperatingPoint.

```
state = computeState(compiled_mb, simscape.op.OperatingPoint)
```

```

state =
  State:

  Status: PositionViolation

  Assembly diagnostics:
  x
    Ro
      Joint successfully assembled
    Rz
      Free position value: +0.000377073 (deg)
      Free velocity value: +0 (deg/s)

```

```
P
  Joint successfully assembled
  Pz
    Free position value: +0.3 (m)
    Free velocity value: +0 (m/s)
Ri
  Joint successfully assembled
  Rz
    Free position value: +0.00087955 (deg)
    Free velocity value: +0 (deg/s)
Rg
  Joint not assembled due to a position violation.
  Rz
    Free position value: N/A (deg)
    Free velocity value: N/A (deg/s)
```

The **State** object returned by the `computeState` method indicates if the attempt to compute a state was successful or not. In this case, it reported indicating that there is a **PositionViolation** and that the joint **Rg** is not assembled due to a position violation. In this example it is in fact true that an error was made in the specification of the frame F_{CA} which would cause a position violation.

Changing the parameters of the rigid transform `ax_fca.BaseAxis2`, from **Axis.NegZ** to **Axis.PosZ** in `dcrankaim_approx_body_C_assembly_failure` fixes the problem and allows the assembly to succeed.

Zero Configuration of Joints

The **Zero Configuration** of a joint is defined as the relative position and orientation between the base and follower frames when all of the joint angles are zero. For almost all of the joints in Simscape Multibody, the base and follower frames are identical in the zero configuration: their origins are coincident, and their axes are aligned. One defines the relative position and orientation between two bodies connected by a joint when the joint angles are zero by adjusting the positions and orientations of the base and follower frames on their respective bodies.

Consider, for example, the rigid bodies **B** and **C** and the joint **Rg** connecting them. The frames F_{CB} and F_{BC} are the base and follower frames of the joint **Rg**. The figure below shows how different choices of orientations for the frame F_{CB} attached to rigid body **C** result in different assembled configurations when the joint angle is zero. The choice of orientation of the frames must be made with the desired zero configuration in mind.



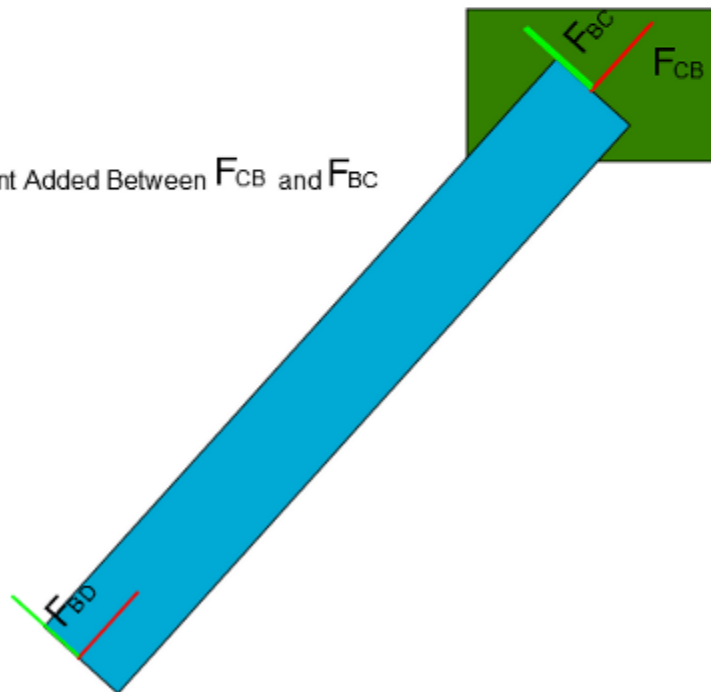
Revolute joint Added Between F_{CB} and F_{BC}



Assembled Zero Configuration



Revolute joint Added Between F_{CB} and F_{BC}



Assembled Zero Configuration

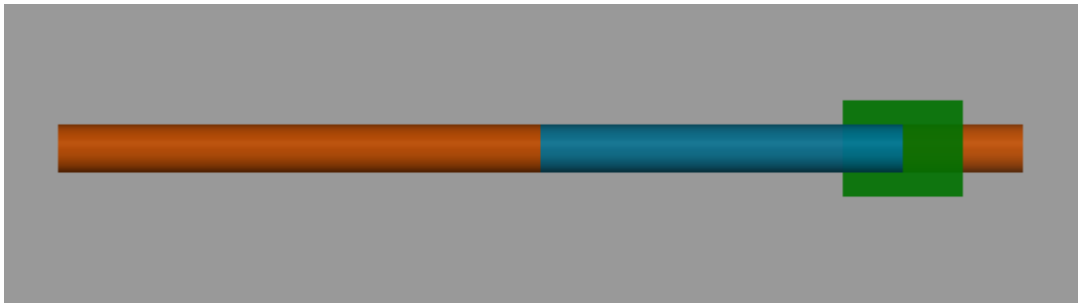
In the aiming mechanism, the choice of frame orientations leads to a default assembled configuration in which the central axes of all of the bodies lie along the same line.

Guiding Assembly Using Operating Points

Visualize the mechanism created using `dcrank_aiming_mechanism_v1` (contains the fix for the errors found in `dcrankaim_approx_body_C_assembly_failure`) using the following code.

```
dcrankAimMech_mb = dcrank_aiming_mechanism_v1(bodyA_l,bodyB_l,bodyC_dim,bodyD_l);
cmb = compile(dcrankAimMech_mb);
visualize(cmb,computeState(cmb,simscape.op.OperatingPoint),'vizAssembly');
```

It can be seen that all of the bodies are collapsed onto a common line; this is the default assembly configuration. In this configuration, all of the revolute joint angles are zero. Thus, the base and follower frames of each revolute joint are coincident and aligned with each other; the corresponding frame pairs are: F_{DA} and F_{AD} , F_{CB} and F_{BC} and F_{BD} and F_{DB} . In contrast, the frame F_{CA} is translated from frame F_{AC} , thus the joint **Pg** is not in its zero state. We can view the `computeState` result to view the values of the joint positions in this assembled configuration. This is currently not a desirable assembly configuration.



The configuration depicted in the schematic diagram of the mechanism is the desired initial assembly configuration. From the schematic diagram we can see that in the initial configuration the angle β is about 35 deg. The assembly algorithm can be guided by specifying joint position and velocity targets using the **OperatingPoint**. In this example, the position target for the joint **Ro** can be set to guide assembly into the desired initial configuration using operating points. The target priority has been set as **High**. Since this is the only target for the system that `computeState` is able to achieve exactly.

```
op = simscape.op.OperatingPoint;
op('Ro/Rz/q') = simscape.op.Target(35, 'deg', 'high') ;
% Note: As an aid to obtain the joint primitive paths needed by the operating point (i.e
% 'Ro/Rz/q'), use jointPrimitivePaths method of the Multibody object.
% For example, in this case we can view all the joint primitive paths using
% >> dcrankAimMech_mb.jointPrimitivePaths;
```

Now compute the state for the above operating point and check the **State** object to see that the joint target for **Ro** has been met exactly.

```
compiled_mb = dcrankAimMech_mb.compile();
state = compiled_mb.computeState(op)
```

```
state =
  State:

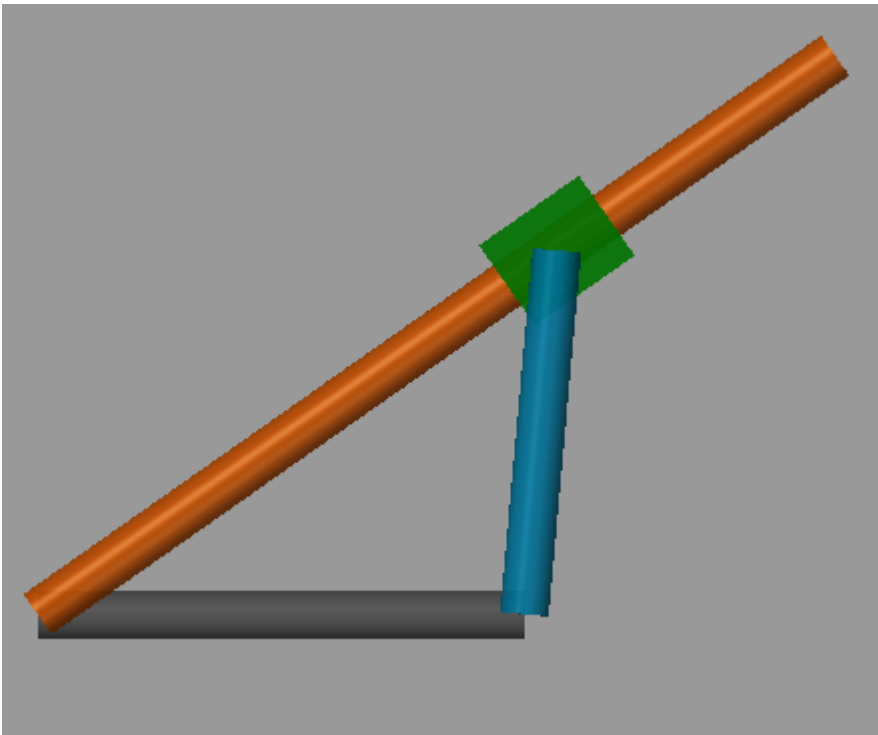
  Status: Valid
```

Assembly diagnostics:

```
x
Ro
  Joint successfully assembled
Rz
  High priority position target +35 (deg) achieved
  Free velocity value: +0 (deg/s)
P
  Joint successfully assembled
Pz
  Free position value: +0.120952 (m)
  Free velocity value: +0 (m/s)
Ri
  Joint successfully assembled
Rz
  Free position value: +84.8864 (deg)
  Free velocity value: +0 (deg/s)
Rg
  Joint successfully assembled
Rz
  Free position value: -49.8864 (deg)
  Free velocity value: +0 (deg/s)
```

Visualize it to view the new configuration

```
compiled_mb.visualize(state, 'vizAimMech');
```



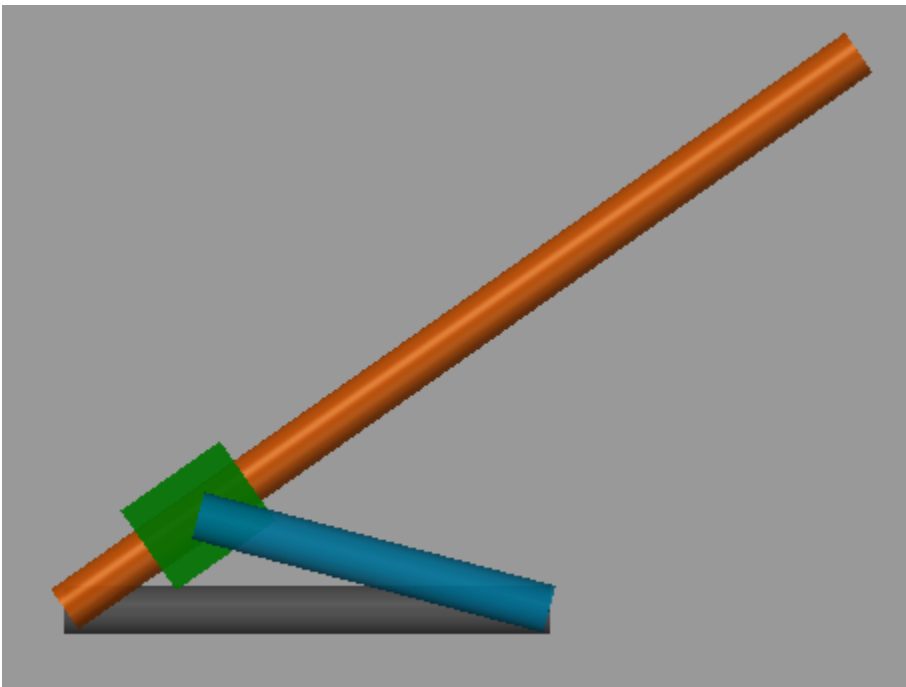
Unfortunately, the assembled configuration is not the intended one because the rigid body **B** is not aligned as indicated in the schematic diagram. Attempting to specify the joint angles of both *Ro* and *Ri* exactly is an over-specification for this one degree-of-freedom mechanism. This is not prohibited,

but if there is a conflict, neither target may be met. Moreover, the desired angle of joint **Ri** is not even known exactly.

In this situation, a convenient approach is to leave the high-priority target of 35 deg on **Ro** but to specify the angle of **Ri** through a low-priority position target. The latter provides an approximate value, or hint, for the desired joint angle. In this case, it is obvious that the angle θ should be obtuse; 150 deg is a rough estimate of its desired value. This target is set for joint **Ri** with a priority of **Low**.

```
op('Ri/Rz/q') = simscape.op.Target(150, 'deg', 'low') ;
compiled_mb = compile(dcrankAimMech_mb);
state = computeState(compiled_mb,op);
visualize(compiled_mb,state,'vizAimMech');
```

The assembled configuration after setting the new target is shown below.



To simulate the mechanism, we can create a simulink model by using the **makeBlockDiagram** method of the **Multibody** object.

```
makeBlockDiagram(dcrankAimMech_mb,op,'dcrankAimMech_model');
```

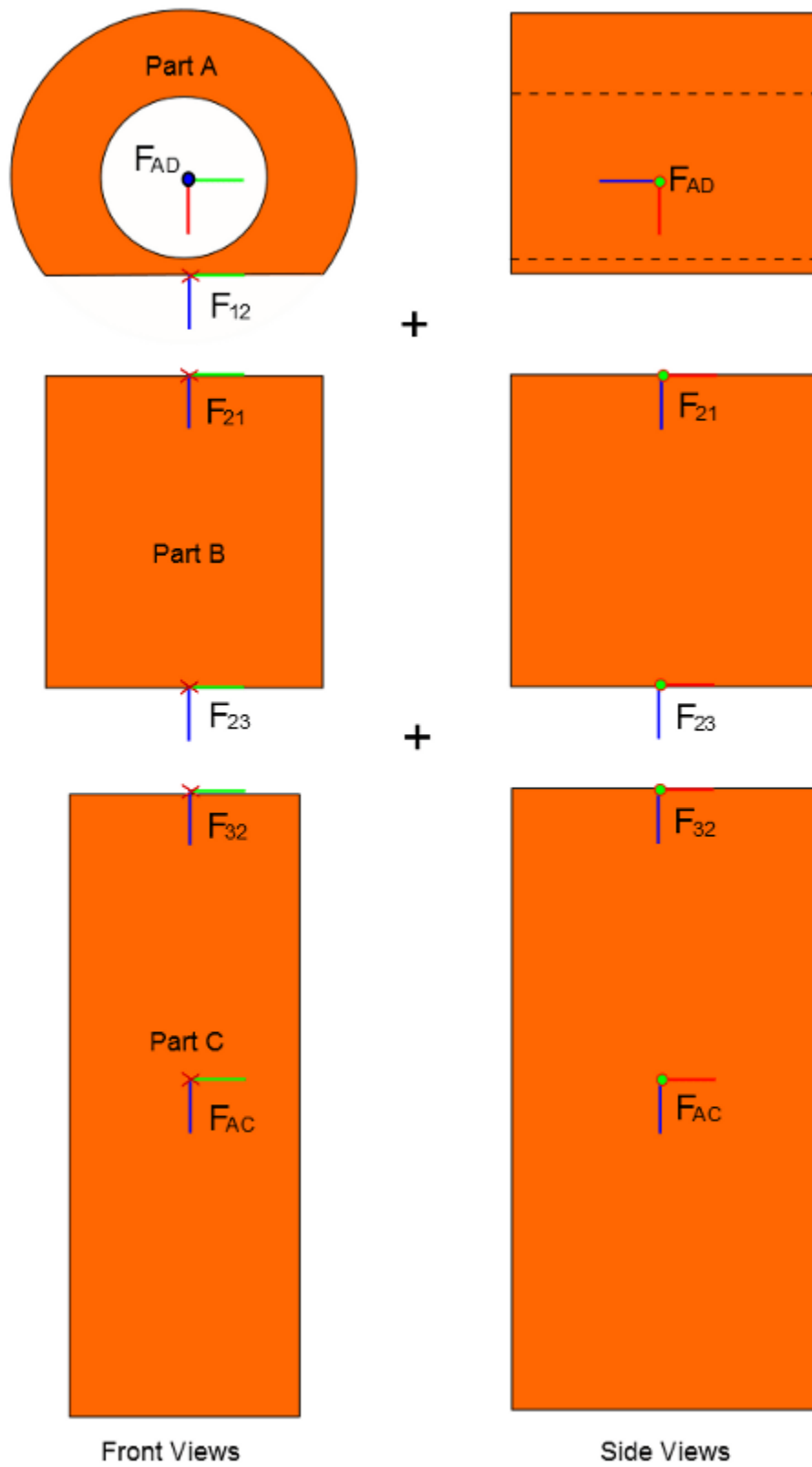
Once the model is created, simulate the model (Ctrl-T) to view the motion of the mechanism under gravity.

Adding Detail to the Rigid Bodies

Now that the basic model is working, the next step is to add detail to make the model more realistic and accurate. Perhaps the first version of the model was created when detailed information about the geometry of the rigid bodies was not yet available. Having carefully established the interfaces of the rigid bodies, it is fairly easy to add detail to each of the rigid bodies without affecting/changing the rest of the model.

As an example, consider adding detail to the rigid body **A** while keeping its interface unchanged. The figure below shows rigid body **A** as a composition of simpler bodies. The interface exposed by rigid

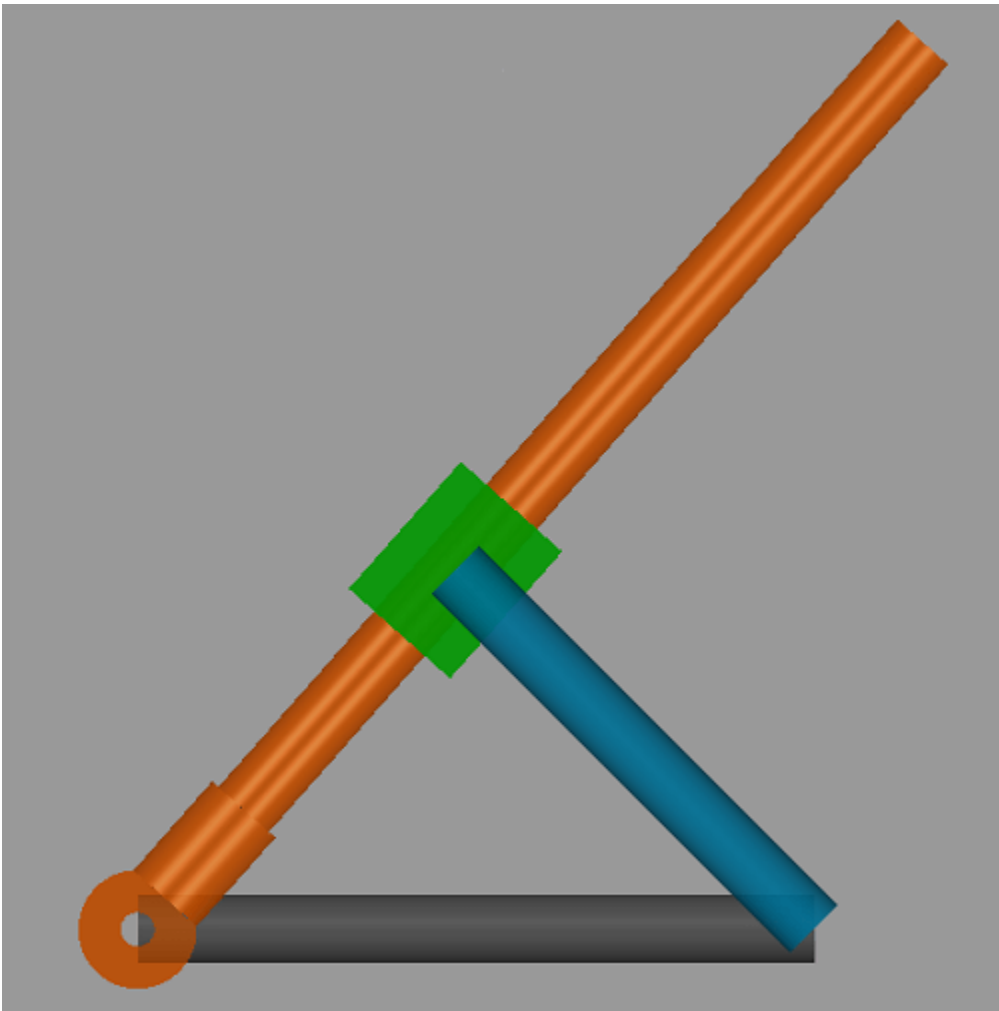
body **A** is still the pair of frames F_{AD} and F_{AC} . Their positions and orientations remain unchanged. The frames F_{12} , F_{21} , F_{23} and F_{32} are internal to the rigid body and should be created to assemble the individual pieces of the rigid body into a whole. The function **dcrankaim_detailed_body_A** shows the construction of the complex version of the rigid body **A**.



The second version function **dcrank_aiming_mechanism_v2** was obtained from **dcrank_aiming_mechanism_v1** by just replacing the call to the function **dcrankaim_approx_body_A** to **dcrankaim_detailed_body_A** to create complex version of rigid body **A**. Because the interface remained constant, it was a simple operation. Create and view the updated mechanism using the following code.

```
dcrankAimMech_mb = dcrank_aiming_mechanism_v2(simscape.Value(80, 'cm'),simscape.Value(30, 'cm'),sin
cmb = compile(dcrankAimMech_mb);
op = OperatingPoint;
op('Ro/Rz/q') = Target(35, 'deg', 'high') ;
op('Ri/Rz/q') = Target(150, 'deg', 'low') ;
visualize(cmb,computeState(cmb,op), 'vizMechDetailed');
```

Similarly, details can be added to the other rigid bodies by following the above discussed process.



Summary

In summary, we took the following steps to build a multibody system in MATLAB :

- Started with a schematic diagram of the mechanism and identified the rigid bodies and joints in the mechanism.

- Built a first approximation of each rigid body in isolation using the **RigidBody** object.
- Assembled the rigid bodies together using joints to achieve the first version of the assembled mechanism using various objects like **Multibody**, **RevoluteJoint**, **PristmaticJoint** etc.
- Used the **computeState** method of **Multibody** object to identify problems with the assembly.
- Used **OperatingPoint** to guide the assembly into a desirable configuration.
- Once a full first version of the model was complete, added details to one of the rigid bodies without changing the interface of the rigid body. Details could be added to other rigid bodies as well.

See Also

[addComponent](#) | [addConnector](#) | [addFrame](#) | [compile](#) | [simscape.multibody.Solid](#) | [simscape.op.OperatingPoint](#) | [simscape.Value](#)

Vehicle Dynamics - Car with Heave and Roll

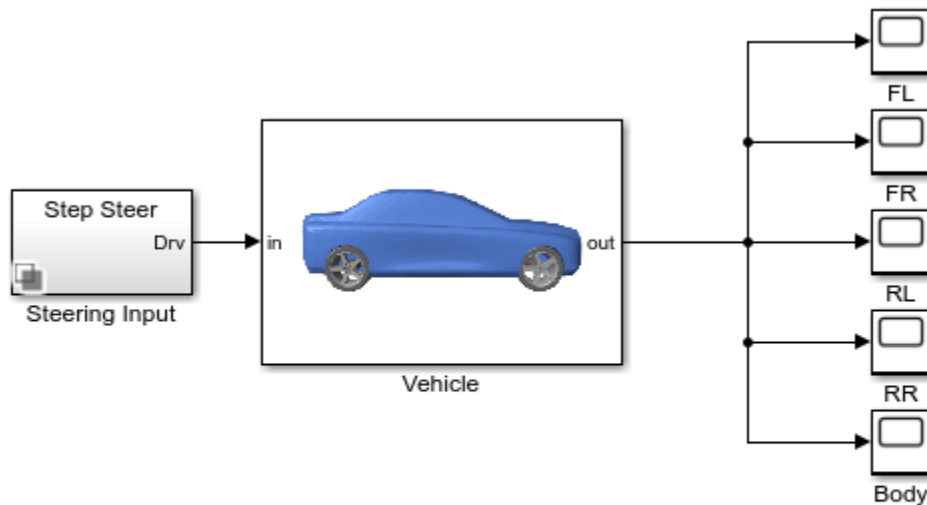
This example models vehicle dynamics using a vehicle model that has 14 degrees of freedom. The driver inputs can be configured as you select one of the maneuvers.

The vehicle model includes a six degree-of-freedom body model, two axles each with heave and roll degrees of freedom, and four wheels that rotate. The front wheels are steered using the Ackermann steering equation. Many of the vehicle parameters can be modified using MATLAB.

The tire model is the Magic Formula Tire Force and Torque block from Simscape Multibody. You can plot the forces and torques at the contact patch from the simulation results.

Explore the Simscape Vehicle Templates for more customizable models of battery-electric vehicles, hybrid-electric, and multi-axle vehicles.

Acknowledgements: MathWorks would like to thank M V Krishna Teja, PhD, Virtual Proving Ground and Simulation Lab, Raghupati Singhanian Centre of Excellence at the Indian Institute of Technology, Madras for contributions to this example, including the tire parameters.



Vehicle Dynamics - Car with Heave and Roll

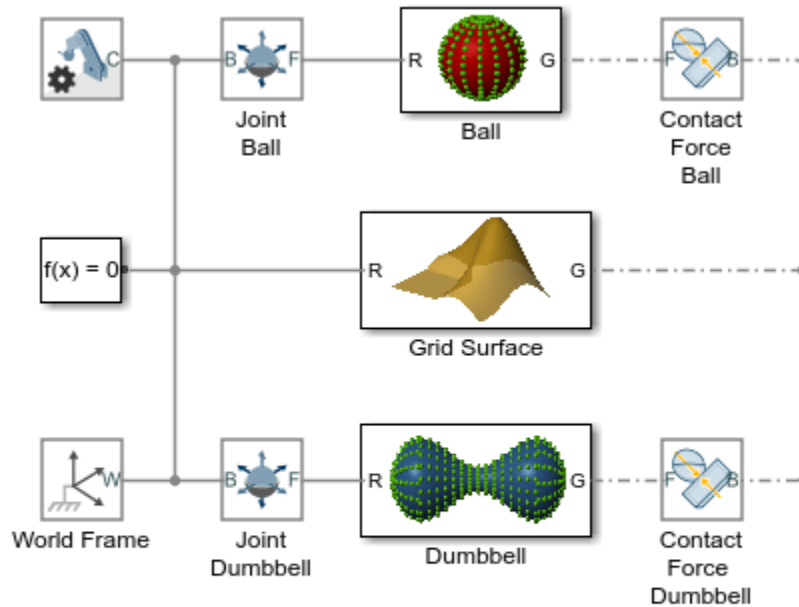
1. Select maneuver: [Slalom](#), [Parking](#), [Sine with Dwell](#), [Step Steer](#), ([see code](#))
2. [Explore simulation results](#) using [Simscape Results Explorer](#)
3. [Learn more](#) about this example

See Also

Magic Formula Tire Force and Torque

Contact Modeling - Ball on Grid Surface

This example shows a ball and dumbbell rolling down on an L-shaped surface. Grid Surface block is used to generate the L-shaped membrane. Point Cloud blocks are used to model the contact between the ball/dumbbell and the surface. The number of point blocks can be controlled by varying the point cloud density. This example demonstrates the capabilities of Grid Surface and Point Cloud blocks to model contacts between complex shaped bodies.



Ball on Grid Surface

1. [Explore simulation results](#) using [Simscape Results Explorer](#)
2. [Learn more](#) about this example

See Also

Grid Surface | Point Cloud