SimMechanics

For Use with Simulink®

- Modeling
- Simulation
- Implementation

User's Guide

Version 2

The MathWorks

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

SimMechanics User's Guide

Revision History

December 2001	Online only	Version 1 (Release 12.1+)
July 2002	First printing	Revised for Version 1.1 (Release 13)
November 2002	Online only	Revised for Version 2.0 (Release 13+)
June 2004	Second printing	Revised for Version 2.2 (Release 14)
October 2004	Online only	Revised for Version 2.2.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.2.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.3 (Release 14SP3)
March 2006	Online only	Revised for Version 2.4 (Release 2006a)
September 2006	Third printing	Revised for Version 2.5 (Release 2006b)

Contents

1 Introducing SimMechanics

2 Building and Visualizing Simple Machines

5 Running Mechanical Models

6 Visualizing and Animating Machines

8 Analyzing Motion

9 Case Studies

Introducing SimMechanics

SimMechanics models and simulates mechanical systems, together with Simulink® and MATLAB®.

What Is SimMechanics?

SimMechanics is a block diagram modeling environment for the engineering design and simulation of rigid body machines and their motions, using the standard Newtonian dynamics of forces and torques.

With SimMechanics, you can model and simulate mechanical systems with a suite of tools to specify bodies and their mass properties, their possible motions, kinematic constraints, and coordinate systems, and to initiate and measure body motions. You represent a mechanical system by a connected block diagram, like other Simulink models, and you can incorporate hierarchical subsystems.

The visualization tools of SimMechanics display and animate simplified renderings of 3-D machines, before and during simulation, using the MATLAB Graphics system.

SimMechanics and Physical Modeling

SimMechanics is part of Simulink Physical Modeling, encompassing the modeling and design of systems according to basic physical principles. Physical Modeling runs within the Simulink environment and interfaces seamlessly with the rest of Simulink and with MATLAB. Unlike other Simulink blocks, which represent mathematical operations or operate on signals, Physical Modeling blocks represent physical components or relationships directly.

Note This SimMechanics User's Guide assumes that you already have some experience with building and running models in Simulink.

Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with SimMechanics.

Requirements for SimMechanics

You must have current versions of the following products installed to use SimMechanics:

- MATLAB
- Simulink

SimMechanics Visualization Requirements

The MATLAB Graphics-based visualization feature for SimMechanics requires Silicon Graphics OpenGL® graphics support on your system in order to render and animate machines.

If you choose to visualize your models in virtual reality, you can improve your speed and graphics resolution by adding a graphics accelerator hardware card to your system. Animation of simulations is sensitive to central processor and graphics card speed and memory. Experiment to find a reasonable compromise between quality and speed for your system.

Support for Recorded MATLAB Graphics Animations

You can record simulation animations in Microsoft Audio Video Interleave® (AVI) format using the MATLAB Graphics-based visualization feature of SimMechanics. To play back AVI files, you need an AVI-compatible media application. MATLAB has an internal movie player compatible with AVI. You can also use an external AVI-compatible player.

If you want to compress your SimMechanics AVI recordings, you need the Indeo 5 codec installed on your system to record them. Your AVI player might also require this codec to view compressed recordings.

Other Related Products

The related products listed on the SimMechanics product page at the MathWorks Web site include toolboxes and blocksets that extend the capabilities of MATLAB and Simulink. These products will enhance your use of SimMechanics in various applications.

Physical Modeling Product Family

Use the Physical Modeling product family to model physical systems in Simulink. In addition to SimMechanics, they include:

- SimDriveline, for modeling and simulating drivetrain systems
- SimHydraulics™, for modeling and simulating hydromechanical systems
- SimPowerSystems, for modeling and simulating electrical power systems

For More Information About MathWorks Products

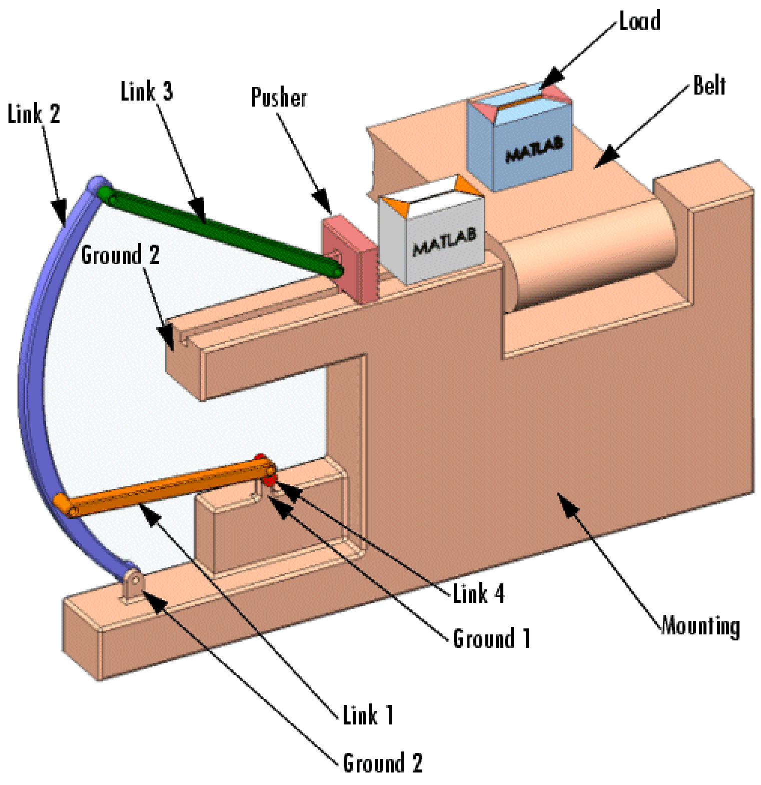
For more information about any MathWorks software products, see either

- The online documentation for that product if it is installed
- The MathWorks Web site at `www.mathworks.com`; see the "Products & Services" section.

Running a Demo Model

This demo model uses a few blocks in the library to simulate a simple machine with feedback control. You will see how SimMechanics implements the model in conjunction with standard Simulink features.

The demo model simulates a conveyor belt loading mechanism. A simple controller (not shown), with a sensor and an actuator, guides the mechanism with a saturation limit and anti-windup logic for the applied torque. The controller is user-adjustable and sets the stopping point for the pusher.



Conveyor Loader Mechanism

What This Demo Illustrates

The conveyor mechanism demo illustrates some important features of SimMechanics:

- Representing bodies and degrees of freedom with Body and Joint blocks, respectively
- Using SimMechanics blocks with normal Simulink blocks
- Feeding Simulink signals to and from SimMechanics blocks with Actuator and Sensor blocks, respectively
- Encapsulating groups of blocks into subsystems
- Visualizing and animating a machine by its component bodies

---

Caution You might want to make modifications to this demo model. To avoid errors,

- Do not attempt to connect Simulink signal lines directly to SimMechanics blocks other than Actuators and Sensors.
- Keep the collocation of the Body coordinate system origins on either side of each assembled Joint to within assembly tolerances.

  Saving modified demo models in a different directory from the demos is recommended.

Opening the Model

You can open the demo model in several ways. Here is the general procedure for starting SimMechanics demos from the **Start** button on the lower left of the MATLAB desktop:

1. Click the **Start** button.
2. In the pop-up menu, select **Simulink**, then **SimMechanics**, and then **Demos**.

This opens the MATLAB Help browser with **Demos** selected in the left **Help Navigator** pane.

3 Double-click **Conveyor Mechanism** from the list of models in the list on the left or on the right.

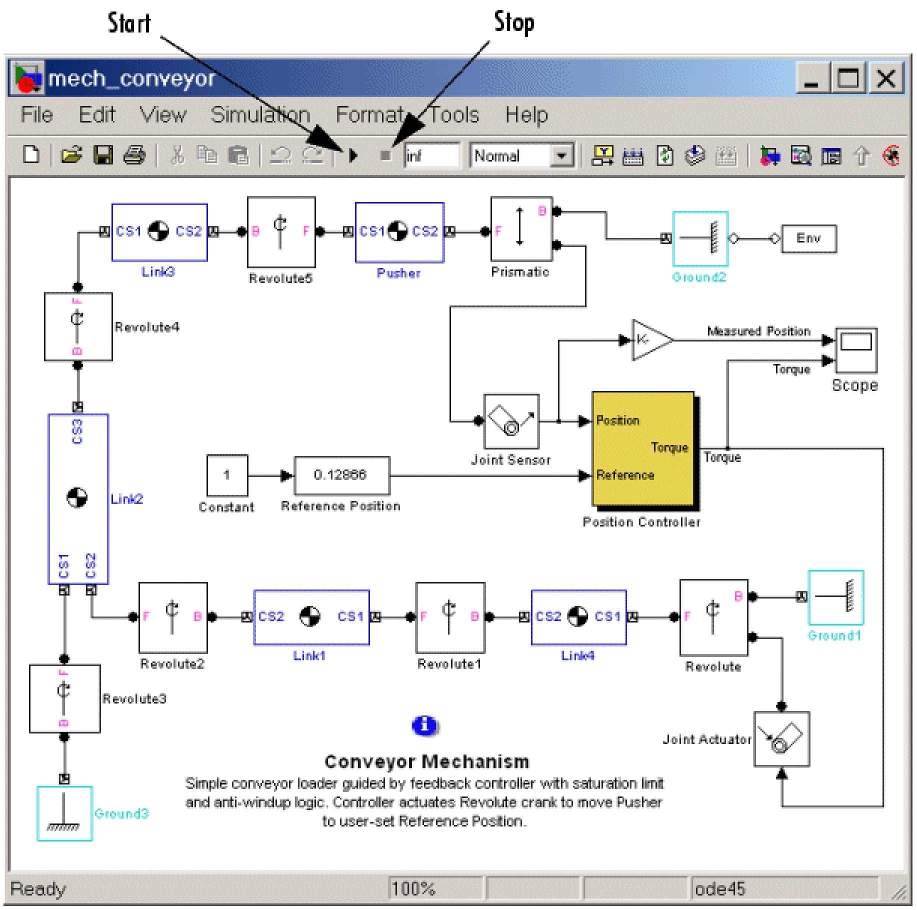
Alternatively, you can open the same MATLAB **Demos** window by entering `demos` at the MATLAB command line.

To get started quickly with this specific demo, you can use either of these steps:

- Enter `mech_conveyor` at the MATLAB command line.
- Online Help users can click the model name `mech_conveyor` here.

The Block Diagram Model

The block diagram model opens in a model window.



Conveyor Mechanism

Simple conveyor loader guided by feedback controller with saturation limit and anti-windup logic. Controller actuates Revolute crank to move Pusher to user-set Reference Position.

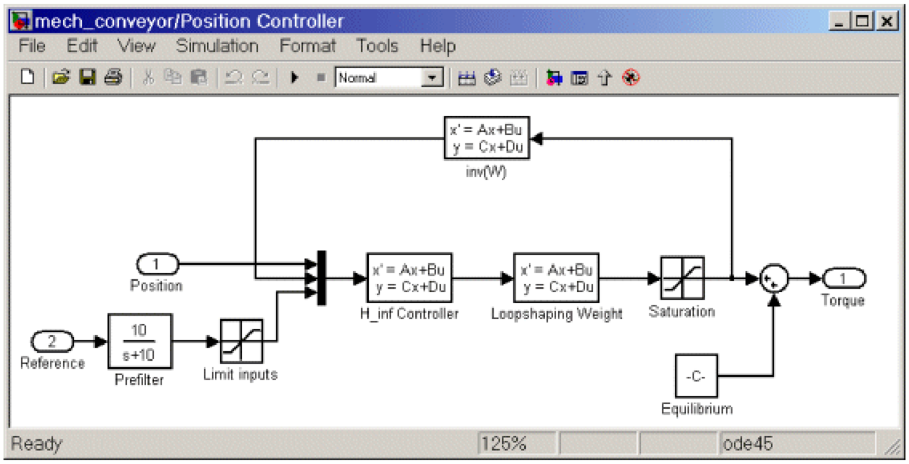
What the Model Contains

Here are some critical features of the model:

- Ignore the Position Controller, Joint Sensor, and Joint Actuator blocks for a moment. Note that the loading mechanism follows the tree of bodies and joints shown in the Conveyor Loader Mechanism on page 1-5:

  - There are four rotating link bodies and one sliding pusher body, as well as three ground points on the immobile mounting represented by Ground blocks. Double-click the Body and Ground blocks to see their dialog boxes.
  - The pusher slides and the links rotate relative to one another and to the ground points on the mounting. There are seven apparent degrees of freedom (DoFs) in the machine, represented by seven Joints, but the geometry constrains the motion to one actual DoF. Double-click the Revolute blocks to see how rotational DoFs are expressed in their dialog boxes.
  - The Prismatic block expresses the linear motion of Pusher relative to Ground_2. The Revolute block expresses the angular motion of Link4 (the crank of the whole mechanism) relative to Ground_1.

- The Joint Sensor detects the position of Pusher via the Prismatic block. The Joint Actuator applies torque to Link4 via the Revolute block. Double-click the Sensor and Actuator blocks to view how the machine motions and forces/torques are transformed into Simulink signals.
- The Position Controller subsystem converts the Pusher position information into a feedback signal to actuate Revolute and thus Link4. You can open the Position Controller block to view this subsystem, which is made of normal Simulink blocks.



- The Reference Position block gives you control over the stopping position of the pusher by modulating the control signal that actuates Revolute. Maintaining the initial pusher position requires a fixed torque on Revolute.
- Open the Scope block. You can view both the Pusher position in millimeters (`mm`) relative to Ground_2 as the Measured Position plot and the torque in newton-meters (N-m) applied to Link4 relative to Ground_1 as the Torque plot.

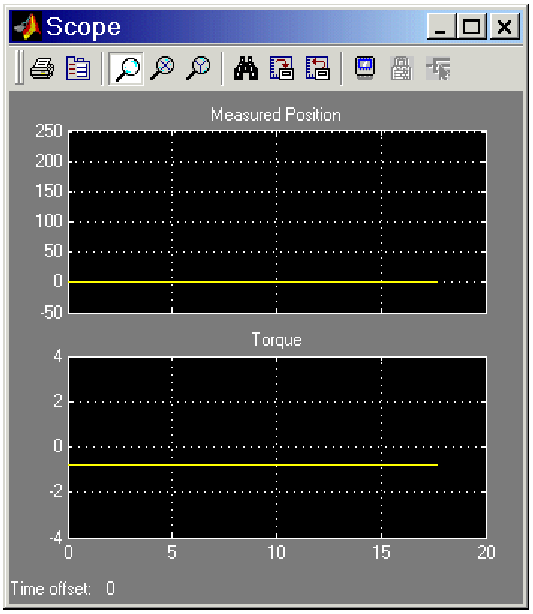
Starting the Demo

You can now run the model as it is when you first open it:

1. In the **Simulation** menu, select **Configuration Parameters**. The **Configuration Parameters** dialog box appears. Select the **Solver** node:
   a. The default **Stop time** is `inf`, so the simulation keeps running once you start it. You should leave it at `inf` and stop the simulation manually the first few times you run it.

      Later you can apply a finite stop time (in seconds) if you want.
   b. Leave the **Solver options** entries at default values and close the box.
2. From the **Simulation** menu, select **Start**. In Microsoft Windows, you can also click the **Start** button in the model window toolbar.

   The measured position of the pusher and the torque applied to maintain that position start and remain essentially constant in the Scope plots. The applied torque is adjusted to maintain the initial pusher position.



3 To see greater detail at the simulation start, stop the simulation before the time passes 20 seconds and zoom in on the Scope plots.

Modifying the Model

Here are two modifications of the demo you can try. One illustrates the simple user-driven controller you can adjust to change the motion of the pusher. The other illustrates a powerful feature of SimMechanics, visualization of a machine and animation of its simulated motion.

To make these modifications, it is best to close and restart the demo.

Changing the Pusher Reference Position

The Reference Position block is actually a Simulink Slider Gain block (from the Simulink Math Library) and controls where the pusher comes to rest.

You can adjust the Reference Position block to change where the pusher stops:

1 Open the Reference Position block. You see an adjustable slider to set the position of the pusher's rest point.

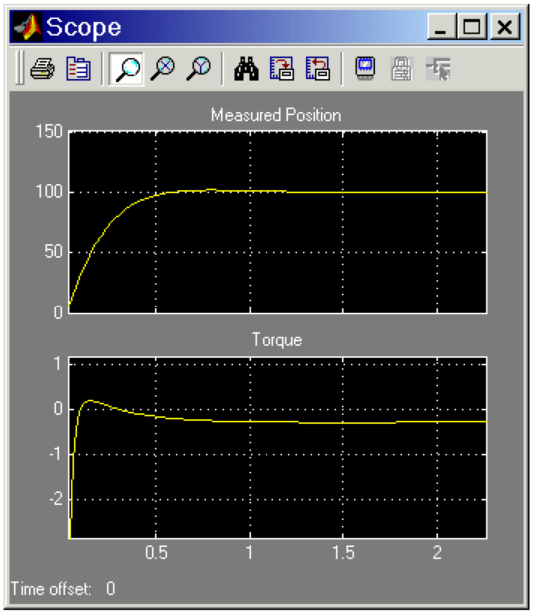
2 Enter values in the **Low** and **High** fields to set the lower and upper limits of the allowed slider range. The defaults in this demo are `0` and `0.2`, with implied units of meters (`m`).

3 Enter a value in the central field to set the pusher stopping point, which you can also adjust by clicking and dragging the slider between the lower and upper limits. The default is `0` (meters).

You can apply changes to the reference position to the simulation in two ways:

- Reset the Reference Position block first, then start the demo. You see the pusher trajectory track differently now, toward the new stopping point.

  For example, resetting the Reference Position to `0.1` and restarting the demo produces these Scope plots, with **Autoscale** and zooming applied. The asymptotic measured position now tends to 100 mm (0.1 m), and the torque applied to keep the pusher there has changed:



- Start the demo with the Reference Position block open and move the slider up and down as the simulation runs. Watch the Scope. The measured position and necessary torque change to follow the new reference position.

Visualizing and Animating the Conveyor

You can visualize the conveyor mechanism as a static machine and animate the simulation as well, by opening the special visualization window built into SimMechanics. This window is based on MATLAB Graphics.

The SimMechanics visualization window lets you display the bodies of the machine in two possible abstract renderings:

- *Equivalent ellipsoids* use the inertia tensors and masses of the bodies. Each body has a unique homogeneous ellipsoid equivalent to it in mass and inertia tensor.
- *Convex hulls* use the attached Body coordinate systems (CSs) of the bodies. A body must have at least four non-coplanar Body CS origins to enclose a convex hull with nonzero volume. If the Body has fewer than four Body CS origins or if the origins are coplanar or collinear or coincident, SimMechanics visualization renders it with simpler figures (triangle, line, or point).

Convex Hulls. First try visualizing the conveyor with bodies rendered as convex hulls:

1. From the **Simulation** menu, select **Configuration Parameters**. The **Configuration Parameters** dialog box appears.

2. Select the **SimMechanics** node. In the **Visualization** area, select the **Update machine visualization on update diagram** and **Animate machine during simulation** check boxes.

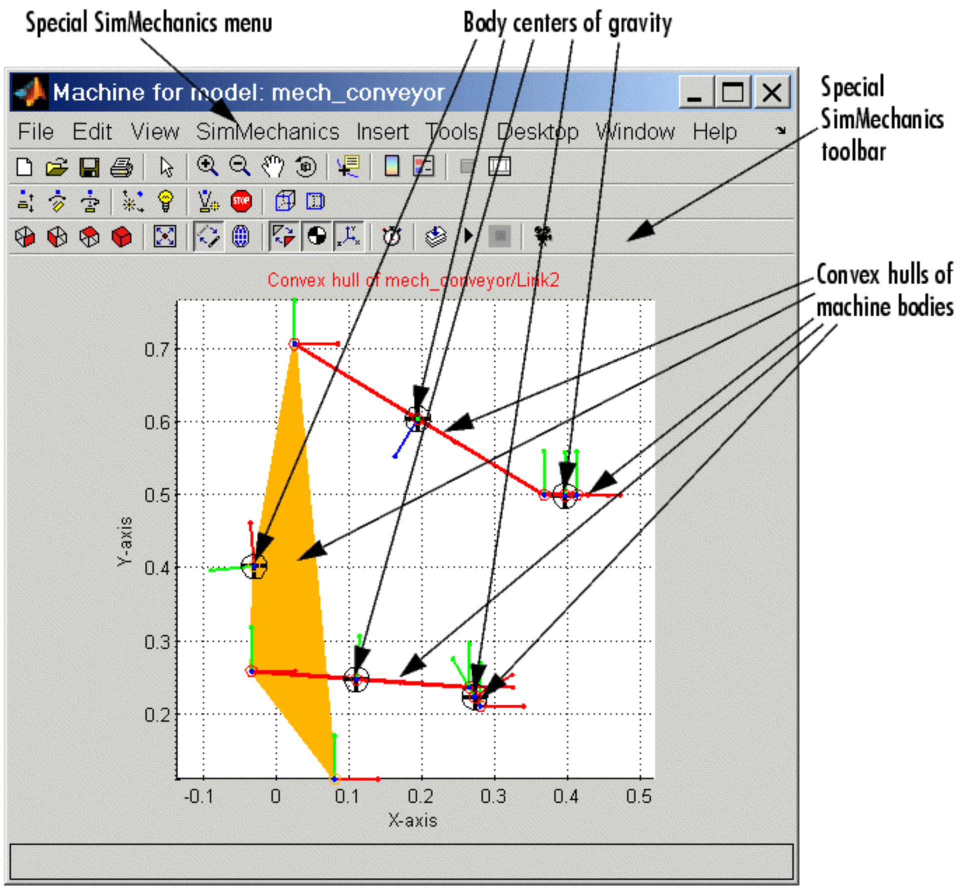
3. Leave the other defaults as they are and close the dialog. From the **Edit** menu, select **Update Diagram**.

   A MATLAB Graphics window appears, displaying the conveyor machine at rest in its initial state.

   The bodies are rendered in the default rendering, as convex hulls. The bodies and Body coordinate system axis triads are also displayed as defaults.

4. Change Reference Position to a nonzero value such as `0.1` or `0.2`.

5 Restart the simulation. The window animates the machine in motion. You can compare this motion to the plots in the scope.


Special SimMechanics menu
Body centers of gravity
Special SimMechanics toolbar
Convex hulls of machine bodies
Machine for model: mech_conveyor
File Edit View SimMechanics Insert Tools Desktop Window Help
Convex hull of mech_conveyor/Link2
Y-axis
X-axis
0.7
0.6
0.5
0.4
0.3
0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5

6 Click a body in the visualization window. The model window comes back into focus with the corresponding Body block highlighted in color.

7 Open the **SimMechanics** menu in the MATLAB Graphics menu bar.

Here, you can reconfigure the SimMechanics display properties for machines: bodies, Body CS axis triads, colored fill-in body surface patches connecting Body CSs on the same body, and user viewpoint orientation.

8 Leave the visualization window open for the next set of steps.

Equivalent ellipsoids. Now visualize the conveyor with bodies rendered as ellipsoids:

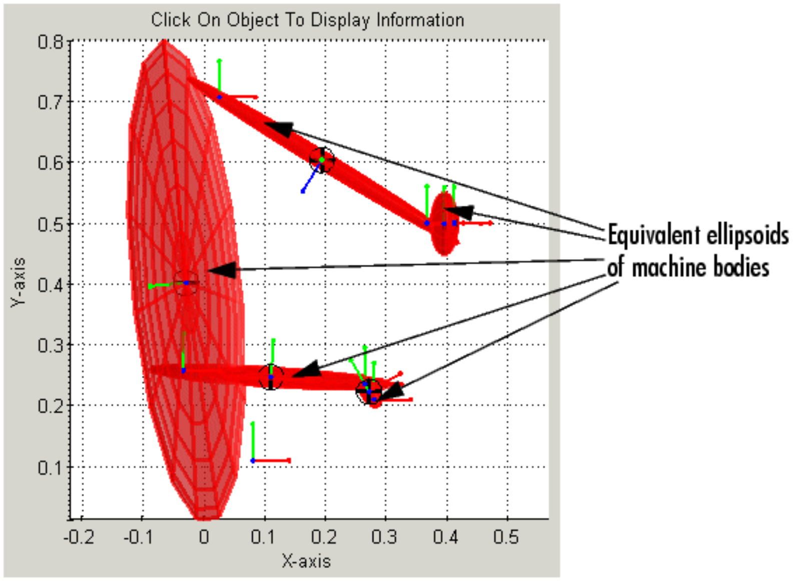
1 From the **SimMechanics** menu at the top of the visualization window, select **Ellipsoids** (so that a check mark appears beside the menu entry) and deselect **Convex Hulls** (so that the check mark beside the menu entry vanishes).

The machine display in the visualization window changes. The conveyor machine appears at rest in its initial state but with the bodies rendered as equivalent ellipsoids.

2 Restart the simulation. The viewer now animates the machine in motion.

3 Use the **SimMechanics** menu to experiment with the visualization settings. The SimMechanics toolbar contains most of these functions as well.

See Chapter 6, "Visualizing and Animating Machines" for more about how to control the machine visualization.


Click On Object To Display Information
Y-axis
X-axis
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
Equivalent ellipsoids of machine bodies

4 While the animation is running, open the Reference Position block and move the slider up and down. In addition to what you can see in the Scope plots, the window directly animates the pusher trajectory in space as the mechanism responds to your adjustment.

What Can You Do with SimMechanics?

SimMechanics is a set of block libraries and mechanical modeling and simulation features for use with Simulink. You connect SimMechanics blocks to normal Simulink blocks through Sensor and Actuator blocks.

The blocks in these libraries are the elements you need to model mechanical systems consisting of any number of rigid bodies, connected by joints representing translational and rotational degrees of freedom. SimMechanics can represent machines with components organized into hierarchical subsystems, as in normal Simulink models. You can impose kinematic constraints, apply forces/torques, integrate Newton's equations, and measure resulting motions. You can see some of these features at work in the Conveyor Loader demo model.

For More Information

See these chapters for complete details on modeling and simulating with SimMechanics:

- Chapter 4, "Modeling Mechanical Systems" for more about machines, bodies, joints, constraints, drivers, sensors, actuators, and force elements
- Chapter 5, "Running Mechanical Models" for more about simulation and code generation
- Chapter 6, "Visualizing and Animating Machines" for more about visualization and animation
- Chapter 7, "Modeling with Computer-Aided Design" for more about combining SimMechanics with computer-aided design (CAD)
- Chapter 8, "Analyzing Motion" for more about motion analysis modes

Glossary Terms For an explanation of important terms, see the Glossary.

Modeling Machines with SimMechanics

These are the major steps you follow to build and run a model representation of a machine:

1 Specify body inertial properties, degrees of freedom, and constraints, along with coordinate systems attached to bodies to measure motions and forces.

2 Set up sensors to record motions and forces, as well as actuators and force elements to initiate motions and apply forces, including continuous and discontinuous friction.

3 Start the simulation, calling the Simulink solvers to find the motions of the system, while maintaining any imposed constraints. You can also generate, compile, and run code versions of your models.

4 Visualize the machine while building the model and animate the simulation while running it, using the SimMechanics visualization window.

Bodies, Coordinate Systems, Joints, and Constraints

SimMechanics supports user-defined Body blocks specified by their masses, inertia tensors, and attached Body coordinate systems (CSs). You connect the bodies to one another with joints representing the possible motions of bodies relative to one another, the system's degrees of freedom (DoFs). You can impose kinematic constraints on the allowed relative motions of the system's bodies. These constraints restrict the DoFs or drive the DoFs as explicit functions of time.

The SimMechanics interface gives you many ways to specify CSs, constraints/drivers, and forces/torques. You can

- Attach Body CSs to different points on Body blocks to specify local axes and origins for actuating and sensing.
- Take Joint blocks from the SimMechanics library or extend the existing Joint library by constructing your own custom Joints.
- Use other Simulink tools as well as MATLAB expressions.

User-Defined Local Coordinate Systems

SimMechanics automatically sets up a single inertial reference frame and CS called *World*. You can also set up your own Local CSs:

- *Grounded CSs* attached to Ground blocks at rest in World but displaced from the World CS origin

- *Body CSs* fixed on and moving rigidly with the bodies

Kinematic Constraints

Specifying kinematic relations between any two bodies, you can constrain the motion of the system by connecting Constraint blocks to pairs of Bodies. Connecting Driver blocks applies time-dependent constraints.

Sensors, Actuators, Friction, and Force Elements

Sensors and Actuators are the blocks you use to interface between normal Simulink blocks and SimMechanics blocks. Force Elements represent internal forces that require no external input.

- Sensor blocks detect the motion of Bodies and Joints.
  - Sensor block outputs are Simulink signals that you can use like any other Simulink signal. You can connect a Sensor block to a Simulink Scope block and display the motions in a system.
  - You can feed these Sensor output signals back to a SimMechanics system via Actuator blocks, to specify forces/torques in the system.
- Actuator blocks specify the motions of Bodies or Joints.
  - They accept force/torque signals from Simulink and can apply forces/torques on a body or joint from these signals. The Simulink signals can include Sensor block outputs fed back from the system itself.
  - They detect discrete locking and unlocking of Joints to implement discontinuous static friction forces.
  - They specify the position, velocity, and acceleration of bodies or joints as explicit functions of time.
  - They prepare a system's initial kinematic state (positions and velocities) for the forward integration of Newtonian dynamics.

Force Elements model internal forces between bodies or acting on joints between bodies. Internal forces depend only on the positions and velocities of the bodies themselves, independent of external signals.

Simulating and Analyzing Mechanical Motion

SimMechanics provides four modes for analyzing the mechanical systems you simulate: Forward Dynamics, Inverse Dynamics, Kinematics, and Trimming. You can also convert any mechanical model, in any mode, to a portable code version.

Mathematical Determination of Rigid Body Motion

For the forward dynamics to be mathematically solvable, the system must satisfy certain conditions:

- The masses and inertia tensors of all bodies are known.
- All forces and torques acting on each body at each instant of time are known.
- Any kinematic constraints among DoFs are specified as constraints among positions and/or velocities alone. If the constraints are mutually consistent and are fewer in number than the DoFs, the system's motion is nontrivial and can be found by integration.
- Initial conditions (initial positions and velocities) are specified and consistent with all constraints.

In Inverse Dynamics or Kinematics analysis modes, you specify the motions instead and obtain the forces/torques needed to produce those motions.

Forward Dynamics and Linearization

In the Forward Dynamics mode, SimMechanics uses the Simulink suite of ordinary differential equation (ODE) solvers to solve Newton's equations, integrating applied forces/torques and obtaining the resulting motions. The ODE solvers project the motion of the DoFs onto the mathematical manifold of the kinematic constraints and yield the forces/torques of constraint acting within the system.

You can also use the Simulink linearization tools to linearize the forward motion of a system and obtain its response to small perturbations in forces/torques, constraints, and initial conditions.

Inverse Dynamics

SimMechanics can solve the reverse of the forward dynamics problem: the Inverse Dynamics mode determines the forces/torques needed to produce a given set of motions that you apply to the machine. This mode works only with *open* topology systems (model diagrams without closed loops).

Kinematics

You cannot analyze machines represented by model diagrams with closed topology (models with loops) using the Inverse Dynamics mode. The Kinematics mode analyzes the motion of *closed-loop* models, including the invisible constraints imposed by loop closures.

You also use the Kinematics mode to determine the forces/torques needed to produce a given set of motions applied to a closed-loop machine model.

Constraint and Driver blocks can appear only in closed loops, so you use the Kinematics mode to analyze constraint forces/torques as well.

Trimming

Finally, the Trimming mode allows you to use the Simulink trimming features to search for steady or equilibrium states in a machine's motion. These states, once found, are the starting point for linearization analysis.

Generating Code

SimMechanics is compatible with the Simulink Accelerator, Real-Time Workshop®, and xPC Target. These optional products let you generate code versions of the models you create originally in Simulink with block diagrams, enhancing simulation speed and model portability.

The presence of static friction in a mechanical model creates dynamical discontinuities and triggers mode iterations in Simulink. These discontinuities and mode iterations place certain restrictions on code generation.

Visualizing and Animating Machines

SimMechanics supports an internal visualization window as a powerful aid in building, animating, and debugging machines. (For an example of its use, see “Running a Demo Model” on page 1-5.) The machine is displayed in a MATLAB Graphics window. You can use all the standard MATLAB Graphics to change the viewer perspective.

The window also has options and features specific to SimMechanics. It displays the bodies and their Body coordinate systems (CSs) in an abstract, simplified form. You can render the bodies as convex hulls or as equivalent ellipsoids.

Visualizing Bodies During Machine Building

One way to use the visualization window is while you’re building your machine:

- You can open a MATLAB Graphics window before you start to build and then watch the bodies appear and be configured in the display as you create and configure them in your model window.

  This approach is especially useful if you’re just starting to learn how to use SimMechanics or how to model complex machines. In that case, visualization can guide you in assembling the body geometries and connections.

- You can also build a model without visualization, then open a MATLAB Graphics window when you’re done to see the completed machine.

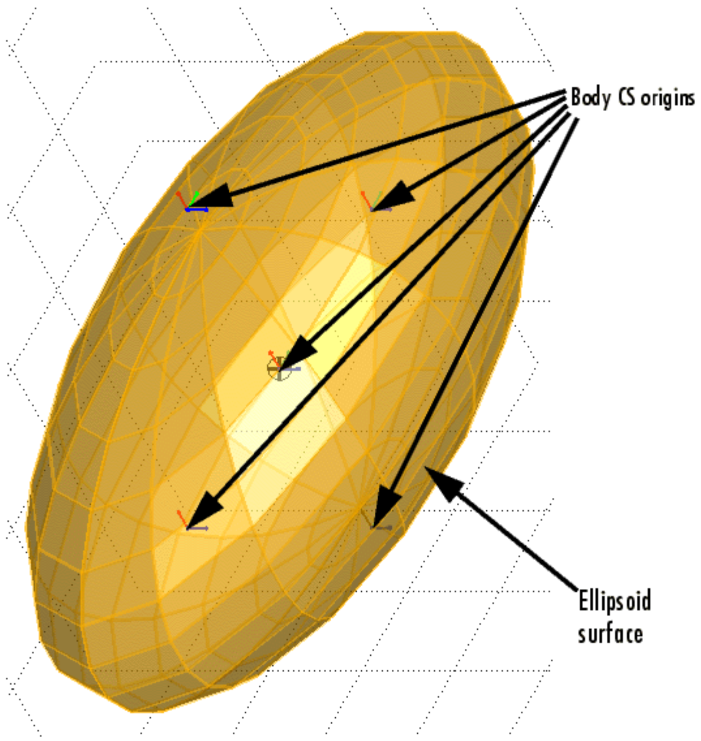
Rendering Bodies

The visualization window has two abstract shapes to render the bodies, one derived from body mass properties, the other from bodies’ attached Body coordinate systems (CSs). These shapes are geometric schematics, because SimMechanics accepts only limited body information.

Mass Properties. A rigid body’s dynamics are partly determined by the body’s total mass and how that mass is distributed in space, as encapsulated in its inertia tensor. Any rigid body has a unique corresponding homogeneous ellipsoid with the same mass and inertia tensor.

Using these *equivalent ellipsoids* is one visualization mode of rendering a body in space. The relative sizes of the ellipsoid axes indicate the relative inertial moments about each axis.

Here is a rigid body rendered by its equivalent ellipsoid.

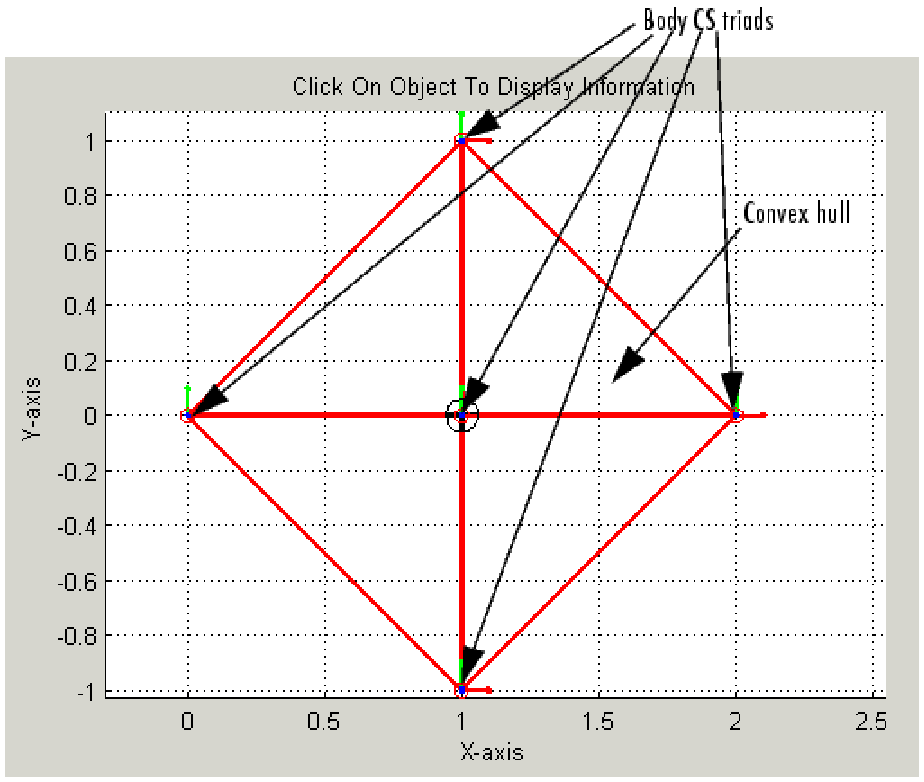


Geometric Properties. In SimMechanics, every body is represented by a Body block with at least one attached Body CS. The minimum Body CS origin is located at the body's center of gravity (CG).

You can also create other Body CSs on a Body. Any Joint, Constraint/Driver, Actuator, or Sensor attached to a Body must be attached at a Body CS origin.

The set of Body CS origins can be enveloped by a surface; if there are more than three non-coplanar origins, the surface encloses a volume. The minimal surface with outward-bending curvature enveloping this set is the *convex hull*, which is the other abstract shape available for visualizing a body in space. Fewer than four CS origins produce simpler Body figures.

Here is the same body as a convex hull. The Body CS origins are coplanar in this case, and the hull is two triangles. The hull body surfaces are turned off to emphasize the hull outline.



Animating Machine Motion During Simulation

Besides rendering your machine bodies either while you build a model or as a completed model, you can also keep the visualization window open while a model is running in the Simulink model window. The MATLAB Graphics window animates the simulation of the bodies' motions, whether you choose to render the bodies as ellipsoids or as convex hulls, and moves in parallel with model changes on the Simulink side.

Learning More

You can get help online in a number of ways to assist you while using SimMechanics.

Using the MATLAB Help System for Documentation and Demos

The MATLAB Help browser allows you to access the documentation and demo models for all the MATLAB and Simulink-based products that you have installed. The online help includes an online index and search system.

Consult the “Help for Using MATLAB” section of the Using MATLAB documentation for more about the MATLAB help system.

Finding Special SimMechanics Help

This user’s guide also includes reference chapters for use with SimMechanics.

- “Technical Conventions” explains mechanical conventions, abbreviations, and units.
- The “Bibliography” lists external references on mechanics, mechanical simulation, and related topics.
- The Glossary explains special terms and phrases used in this guide.

In addition, many SimMechanics demos have help links represented by the information symbol ⓘ. Click this symbol to open that demo’s documentation in the Help browser.

Building and Visualizing Simple Machines

Constructing simple mechanical models with SimMechanics is easy to learn if you already know how to make Simulink models. If you are not already familiar with Simulink, see the Simulink documentation.

Introducing the SimMechanics Block Libraries

SimMechanics is organized into hierarchical libraries of related blocks. The next section, "Viewing the Blocks" on page 2-2 shows how to view these libraries and gives you a summary of what they contain.

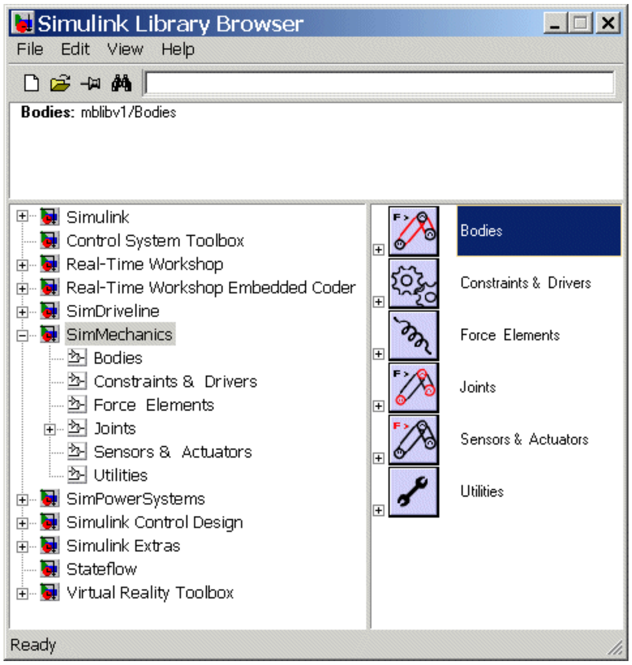
- "Bodies Library" on page 2-4
- "Joints Library" on page 2-5
- "Constraints & Drivers Library" on page 2-5
- "Sensors & Actuators Library" on page 2-5
- "Force Elements Library" on page 2-5
- "Utilities Library" on page 2-6

Viewing the Blocks

There are several ways to get to the top-level SimMechanics library on Microsoft Windows and UNIX platforms.

Microsoft Windows Platforms

Microsoft Windows users can access the blocks through the Simulink Library Browser. Expand the SimMechanics entry in the contents tree.



You can also access the blocks directly inside the SimMechanics library in several ways:

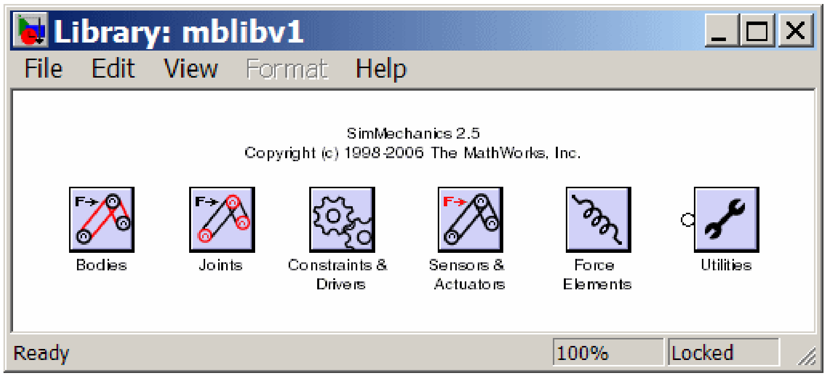
- In the Simulink Library Browser, right-click the SimMechanics entry and select **Open the SimMechanics Library**. The library appears.
- Click the **Start** button in the lower left corner of your MATLAB desktop. In the pop-up menu, select **Simulink**, then **SimMechanics**, then **Block Library**.
- Enter `mechlib` at the MATLAB command line prompt.

UNIX Platforms

UNIX users can click the Simulink icon on the MATLAB menu bar, open the Blocksets & Toolboxes library and then SimMechanics. You can also enter `mechlib` at the command line.

The SimMechanics Library

Once you perform one of these steps, the SimMechanics library opens.



Note This library displays six top-level block groups. You can expand each library by double-clicking its icon. The Joints library contains two second-level sublibraries.

The following sections summarize the blocks in each library. For an explanation of special terms, see the Glossary. You can also consult the SimMechanics block reference.

Bodies Library

The Bodies library provides the Body block for representing user-defined bodies by their mass properties (masses and inertia tensors), their positions and orientations, and their attached Body coordinate systems (CSs). This library also contains the Ground block representing immobile ground points, which have their own Grounded CSs, and the Machine Environment block, for configuring the mechanical settings of a SimMechanics block diagram.

Joints Library

The Joints library provides blocks to represent the relative motions between bodies as degrees of freedom (DoFs). The library is made up of assembled Joints listed individually and two sublibraries of specialized Joint blocks.

An assembled joint restricts the Body CSs on the two bodies to which it is connected. The assembled Joints are the primitive Prismatic, Revolute, and Spherical blocks and ready-made composite Joints. Unless it is explicitly labeled as disassembled, you can assume a generic Joint block is assembled.

Joints/Disassembled Joints Sublibrary. The Disassembled Joints sublibrary provides blocks for disassembled joints, modified joints that do not restrict the Body CSs on the two connected bodies or the DoF axes of the two bodies. You can only use Disassembled Joints to close a loop in your machine. You cannot sense or actuate Disassembled Joints.

Joints/Massless Connectors Sublibrary. The Massless Connectors sublibrary provides blocks for massless connectors, composite joints whose DoFs are separated by a fixed distance. You cannot actuate or sense Massless Connectors.

Constraints & Drivers Library

The Constraints & Drivers library provides blocks to specify prior restrictions on DoFs between Bodies. These restrictions can be time-independent constraints or time-dependent driving of DoFs with Simulink signals.

Sensors & Actuators Library

The Sensors & Actuators library provides blocks for sensing and initiating the motions of joints and bodies. These blocks play a special role in connecting SimMechanics blocks to other Simulink blocks, as described in “Connecting SimMechanics Blocks” on page 4-5, “Modeling Actuators” on page 4-45, and “Modeling Sensors” on page 4-63.

Force Elements Library

The Force Elements library provides blocks for creating forces or torques between bodies. These blocks model forces internal to your machine.

Utilities Library

The Utilities library contains miscellaneous blocks useful in building models.

Creating SimMechanics Models

To become comfortable building mechanical models, you might find it helpful to work through the guided examples in subsequent sections of how to configure and put together elements of SimMechanics to simulate simple machines. This section gives you an overview of the model-building process before you start:

- "Essential Steps to Build a Model" on page 2-7
- "Essential Steps to Configure and Run a Model" on page 2-9

The special terms used in this guide are summarized in the "Glossary" on page Glossary-1.

Essential Steps to Build a Model

You use the same basic procedure for building a SimMechanics model regardless of its complexity. The steps are similar to those for building a regular Simulink model. More complex models add steps without changing these basics.

1. *Select Ground, Body, and Joint blocks.* From the Bodies and Joints libraries, drag and drop the Body and Joint blocks needed to represent your machine, including a Machine Environment block and at least one Ground block, into a Simulink model window.

   The Machine Environment block represents your machine's mechanical settings.

   Ground blocks represent immobile ground points at rest in absolute (inertial) space.

   Body blocks represent rigid bodies.

   Joint blocks represent relative motions between the Body blocks to which they are connected.

2. *Position and connect blocks.* Place Joint and Body blocks in proper relative position in the model window and connect them in the proper order. The essential result of this step is creation of a *valid tree* block diagram made of

Machine Env — Ground — Joint — Body — Joint — Body — ... — Body

with an open or closed topology and where at least one of the bodies is a Ground block. Connect exactly one environment block to a Ground.

A Body can have more than two Joints attached, marking a branching of the sequence. But Joints must be attached to two and only two Bodies.

3 *Configure Body blocks.* Click the Body blocks to open their dialog boxes; specify their mass properties (masses and moments of inertia), then position and orient the Bodies and Grounds relative to the World coordinate system (CS) or to other CSs. You set up Body CSs here.

   Look for intensive explanation and examples of positioning and orienting bodies in Chapter 3, "Representing Motion".

4 *Configure Joint blocks.* Click each of the Joint blocks to open its dialog box and set translation and rotation axes and spherical pivot points.

5 *Select, connect, and configure Constraint and Driver blocks.* From the Constraints & Drivers library, drag, drop, and connect Constraint and Driver blocks in between pairs of Body blocks. Open and configure each Constraint/Driver's dialog box to restrict or drive the relative motion between the two respective bodies of each constrained/driven pair.

6 *Select, connect, and configure Actuator and Sensor blocks.* From the Sensors & Actuators library, drag and drop the Actuator and Sensor blocks that you need to impart and sense motion. Reconfigure Body, Joint, and Constraint/Driver blocks to accept Sensor and Actuator connections. Connect Sensor and Actuator blocks. Specify control signals (applied forces/torques or motions) through Actuators and measure motions through Sensors.

   Actuator and Sensor blocks connect SimMechanics blocks to non-SimMechanics Simulink blocks. You cannot connect SimMechanics blocks to regular Simulink blocks otherwise. Actuator blocks take inport signals from normal Simulink blocks (for example, from the Simulink Sources library) to actuate motion. Sensor block output ports generate Simulink signals that you can feed to normal Simulink blocks (for example, from the Simulink Sinks library).

In the most straightforward model of a machine, you apply forces/torques and initial conditions, then start the simulation in the Forward Dynamics mode to obtain the resulting motions. In the Kinematics and Inverse Dynamics modes, you apply motions to all independent degrees of freedom. With these modes, you can find the forces/torques needed to produce these imposed motions.

7. *Encapsulate subsystems.* Systems made from SimMechanics blocks can function as subsystems of larger models, like subsystems in normal Simulink models. You can connect an entire SimMechanics model as a subsystem to a larger model by using the Connection Port block in the Utilities library.

Essential Steps to Configure and Run a Model

After you've built your model as a connected block diagram, you need to decide how you want to run your machine, configure SimMechanics and Simulink settings, and set up visualization.

- SimMechanics offers four analysis modes for running a machine model. The mode you will probably use most often is Forward Dynamics.

  But a more complete analysis of a machine makes use of the Kinematics, Inverse Dynamics, and Trimming modes as well. You can create multiple versions of the model, each with the same underlying machine, but connected to Sensors and Actuators and configured differently for different modes.

- You can also use the powerful visualization and animation features of SimMechanics. You can visualize your machine as you build it or after you are finished but before you start the simulation, as a tool for debugging the machine geometry. You can also animate the machine model as you simulate.

- Choose the analysis mode, as well as other important mechanical settings, in your Machine Environment dialog. Start visualization and adjust Simulink settings in the Simulink **Configuration Parameters** dialog. See "Four Bar Mechanism" on page 2-36 for an example.

The tutorials of this chapter introduce you to most of these steps.

Caution You might want to make modifications to these tutorial models. To avoid errors,

- Do not attempt to connect Simulink signal lines directly to SimMechanics blocks other than Actuators and Sensors
- Keep the collocation of the Body coordinate system origins on either side of each assembled Joint to within assembly tolerances

  You should save multiple versions of models as you try different analysis modes and configurations.

The first tutorial in the next section shows you how to configure the most basic blocks in any model: Machine Environment, Ground, Body, and a Joint, in order to create a simple pendulum model. The second tutorial explains how to visualize and animate the pendulum.

Building a Simple Pendulum

In this first tutorial, you drag, drop, and configure the most basic blocks needed for any mechanical model, as well as add some sensors to measure motion. The tutorial guides you through these aspects of model-building:

- "The World Coordinate System and Gravity" on page 2-12
- "Configuring a Ground Block" on page 2-13
- "Configuring a Body Block" on page 2-14
- "Configuring a Joint Block" on page 2-20
- "Adding a Sensor and Starting the Simulation" on page 2-24

The end result is a model of a simple pendulum. The pendulum is a swinging steel rod. We strongly recommend that users work through this tutorial first before moving on to, "Visualizing a Simple Pendulum" on page 2-30.

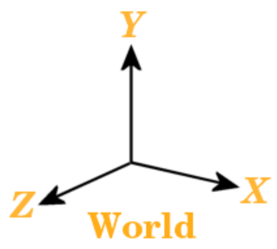
A Simple Pendulum: A Swinging Steel Rod

Opening the SimMechanics Block Library

Following one of the ways described earlier in the "Viewing the Blocks" on page 2-2 section in this chapter, open the SimMechanics library. Then from the SimMechanics library, open a new, empty Simulink model window.

The World Coordinate System and Gravity

Before you configure a Ground block, you need to understand SimMechanics' internally defined fixed or "absolute" coordinate system (CS) called *World*. The World CS sits at rest in the inertial reference frame also called World. The World CS has an origin (0,0,0) and a triad of right-handed, orthogonal coordinate axes.



The default World coordinate axes are defined so that

+*x* points right

+*y* points up (gravity in -*y* direction)

+*z* points out of the screen, in three dimensions

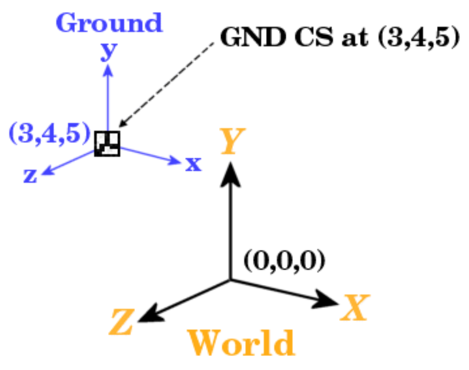
The vertical direction or up-and-down is determined by the gravity vector direction (acceleration ***g***) relative to the World axes. Gravity is a background property of a model that you can reset before starting a simulation, but does not dynamically change during a simulation.

See "Running SimMechanics Models in Simulink" on page 5-2 for displaying global mechanical properties of SimMechanics models.

Configuring a Ground Block

World serves as the single absolute CS that defines all other CSs. But you can create additional *ground points* at rest in World, at positions other than the World origin, by using Ground blocks. Ground blocks, representing ground points, play a dynamical role in machine models. They function as immobile bodies and also serve to implement a machine's mechanical environment.

Minimum Ground Blocks Every machine model must have *at least one* Ground block. Exactly one Ground block in every machine must be connected to a Machine Environment block.

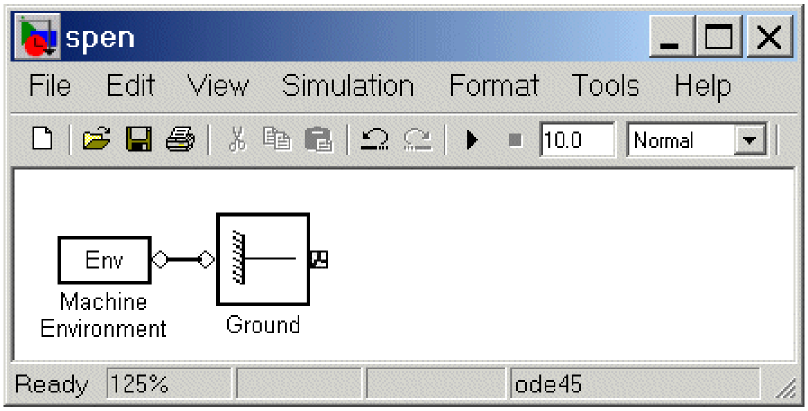


A Ground Point Relative to World

Steps to Configuring the Ground Block

Now place a fixed ground point at position (3,4,5) in the World CS:

1. In the SimMechanics library, open the Bodies library.
2. Drag and drop a Ground and a Machine Environment block from the Bodies library into the model window. Close the Bodies library.
3. Open the Ground block dialog box. Into the **Location [x y z]** field, enter the vector `[3 4 5]`. Select the **Show Machine Environment port** check box. Click **OK** to close the dialog. Connect the environment block.



Properties of Grounds

At every ground point, a *Grounded CS* is automatically created:

- The origin of each Grounded CS is the ground point itself.
- The Grounded CS axes are always fixed to be parallel to the World CS axes, as shown in the figure A Ground Point Relative to World on page 2-13.

Configuring a Body Block

While you need one Machine Environment and at least one Ground block to make a machine model, a real machine consists of one or more rigid bodies. So you need to translate the components of a real machine into block representations. This section explains how you use a Body block to represent each rigid body in your machine:

- "Characteristics of a Body Block" on page 2-15
- "Properties of the Simple Pendulum Body" on page 2-16
- "Configuring the Body Dialog" on page 2-17

Although the body is the most complicated component of a machine, SimMechanics does not use the full geometric shape and mass distribution of the body. SimMechanics only needs certain mass properties and simplified geometric information about the body's center of gravity, its orientation, and the coordinate systems attached to the body. Chapter 3, “Representing Motion” explains in detail how to orient bodies and their coordinate systems.

Setting these properties sets the body’s initial conditions of motion, if you do nothing else to the Body block or its connected Joints before simulating.

Characteristics of a Body Block

The main characteristics of a Body block are its *mass properties*, its *position* and *orientation* in space, and its attached *Body coordinate systems* (CSs).

The mass properties include the *mass* and *inertia tensor*. The mass is a real, positive scalar. The inertia tensor is a real, symmetric 3-by-3 matrix. It does not have to be diagonal.

The position of the body’s *center of gravity* (CG) and *orientation* relative to some coordinate system axes indicate where the body is and how it is rotated. These are the body’s initial conditions during construction of the model and remain so when you start the simulation, unless you change them before starting.

The attached *Body CSs* (their origins and coordinate axes) are fixed rigidly in the body and move with it. The minimum CS set is one, the CS at the CG (the CG CS), with its CS origin at the center of gravity of the body. The default CS set is three, the CG CS and two additional CSs called CS1 and CS2 for connecting to Joints on either side. See the next section, “Configuring a Joint Block” on page 2-20.

Beyond the minimum CS at the CG, you can attach as many Body CSs on one Body as you want. You need a separate CS for each connected Joint, Constraint, or Driver and for each attached Actuator and Sensor.

The inertia tensor components are interpreted in the CG CS, setting the orientation of the body relative to the CG CS axes. The orientation of the CG CS axes relative to the World axes then determines the absolute initial orientation of the body. Since the CG CS axes remain rigidly fixed in the body during the simulation, this relative orientation of the CG CS axes and the body does not change during motion. The inertia tensor components in the CG CS also do not change. As the body rotates in inertial space, however, the CG CS axes rotate with it, measured with respect to the absolute World axes.

Properties of the Simple Pendulum Body

The simple pendulum is a uniform, cylindrical steel rod of length 1 meter and diameter 2 cm. The initial condition is the rod lying parallel to the negative x-axis, horizontal in the gravity field. One end of the rod, the fixed pivot for the rod to swing, is located at the ground point (3,4,5). Its coordinate system is called CS1. The center of gravity and the origin of the CG CS is the geometric center of the rod. Take the CG CS axes to be parallel to the World axes as you set up the pendulum.

Uniform steel has density ρ = 7.93 gm/cc (grams per cubic centimeter). In the CG CS here, the inertia tensor $\boldsymbol{I}$ is diagonal, and I_{zz} controls the swinging about the z-axis, in the x-y plane. The inertia tensor is always evaluated with the origin of coordinates at the CG. For a rod of length L = 1 m and radius r = 1 cm, the mass $m = \rho\pi r^2 L$ = 2490 gm (grams), and the inertia tensor $\boldsymbol{I}$ reads

$$\begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} = \begin{pmatrix} \frac{mr^2}{2} & 0 & 0 \\ 0 & \frac{mL^2}{12} & 0 \\ 0 & 0 & \frac{mL^2}{12} \end{pmatrix} = \begin{pmatrix} 1250 & 0 & 0 \\ 0 & 2.08\times10^6 & 0 \\ 0 & 0 & 2.08\times10^6 \end{pmatrix}$$

in gm-cm^2 (gram-centimeters^2). The x-axis is the cylinder's symmetry axis. Thus $I_{yy} = I_{zz}$.

The mass and geometric properties of the body are summarized in the following table and depicted in the figure Equivalent Ellipsoid of Simple Pendulum with Body Coordinate Systems on page 2-17.

Body Data for the Simple Pendulum

Property	Value
Mass (gm)	`2490`

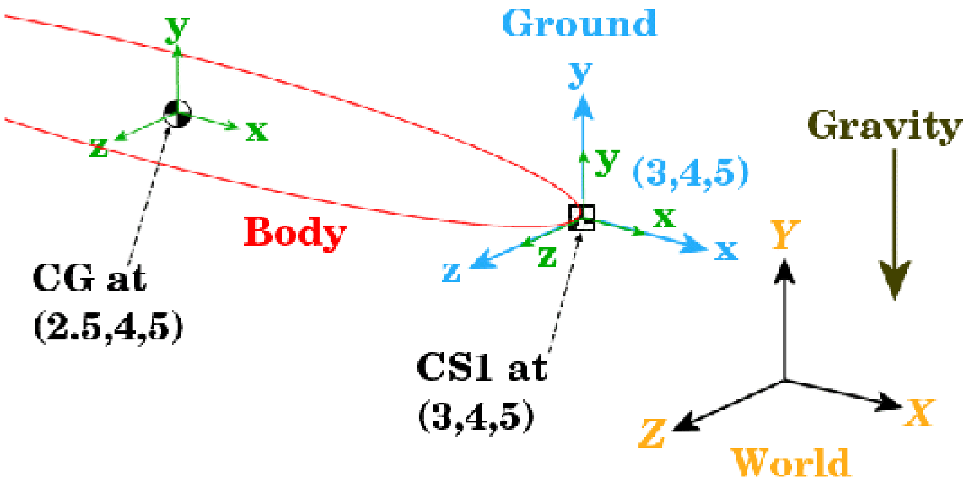
Inertia tensor (kg-m^2)	`[1.25e-4 0 0;`
	`0 0.208 0;`
	`0 0 0.208]`

Body Data for the Simple Pendulum (Continued)

Property	Value
CG Position/Origin (m)	`[ 2.5 4 5 ]`
CS1 Origin (m)	`[ 3 4 5 ]`

Configuring the Body Dialog

Take the steps to configuring a Body block in several stages.



Equivalent Ellipsoid of Simple Pendulum with Body Coordinate Systems

Adding the Body Block. To start working with the Body block:

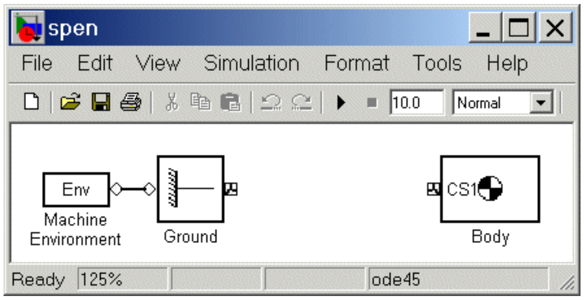
1 Open the Bodies library in the SimMechanics library.

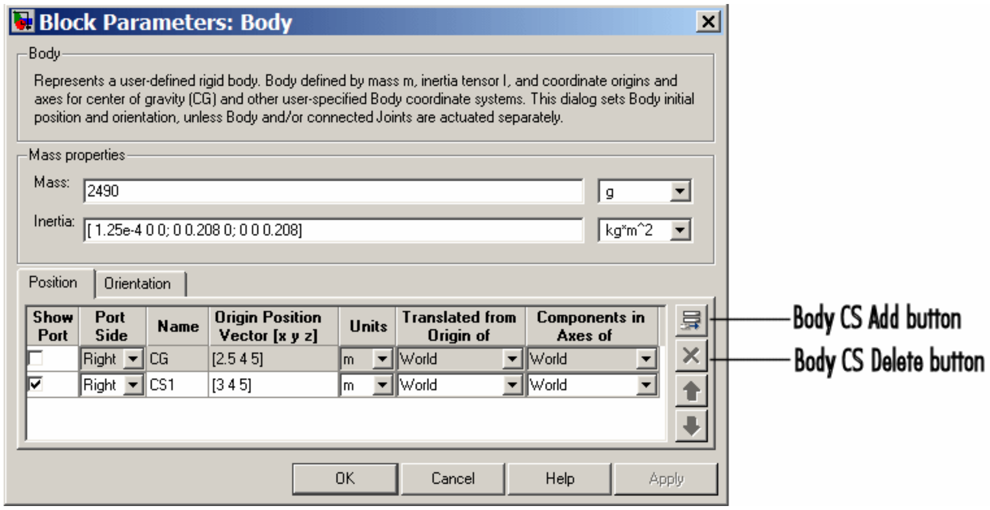
2 Drag and drop a Body block into your model window.

3 Open the Body block dialog box. Note the two main areas you need to configure:

- **Mass properties** — These are the mass and inertia tensor.
- Body coordinate systems — These are the details about the position and orientation of the Body CSs.

Note To apply your dialog entries at any time, click **Apply**. To apply your entries and close a dialog, click **OK**.





Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties

Mass: 2490 g

Inertia: [1.25e-4 0 0; 0 0.208 0; 0 0 0.208] kg*m^2

Position | Orientation

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Right	CG	[2.5 4 5]	m	World	World
✓	Right	CS1	[3 4 5]	m	World	World

Specifying the Body's Mass Properties. Now enter the body's mass and inertia tensor:

1 Use the data from the table Body Data for the Simple Pendulum on page 2-16.

   In the **Mass** field, enter `2490` and change the units to `g` (grams).

2 In the **Inertia** field, enter `[1.25e-4 0 0; 0 0.208 0; 0 0 0.208]` and leave the default units as `kg-m`^{2}.

Specifying Body Coordinate Systems (Position). Enter the translational position of the body and its Body CS origins in space:

1 Use the data from the table Body Data for the Simple Pendulum on page 2-16, and work on the **Position** pane. Vectors are assumed translated from the World origin and oriented to the World axes.

2 Note the three default CSs in the Body dialog box. The CS at the CG is necessary for any Body, and you will connect CS1 to the Ground with a Joint shortly.

   Delete CS2 by selecting its line in the Body CS list and clicking the **Delete** button in the Body CS controls.

   You have two already existing CSs not on this Body that you can use to specify the positions of the Body CS origins that are on this Body:

   - Preexisting World origin at `[0 0 0]`
   - The Adjoining CS on the neighboring body, in this case the Grounded CS origin at `[3 4 5]`

3 Specify the CG and CS1 origins relative to World:

   a In the pull-down menu under **Translated from Origin of**, choose `World` for both coordinate systems, CG and CS1.

   b Under **Origin Position Vector**, specify the position of the origin of each CS, translated from the World origin:

      `[3 4 5]` for CS1

      `[2.5 4 5]` for CG

4 Select a CS relative to whose coordinate axes the components of the vectors in the last step are measured. You choose these CS axes in the **Components in Axes of** menu. Select `World` for both CSs. Leave the units as `m` (meters).

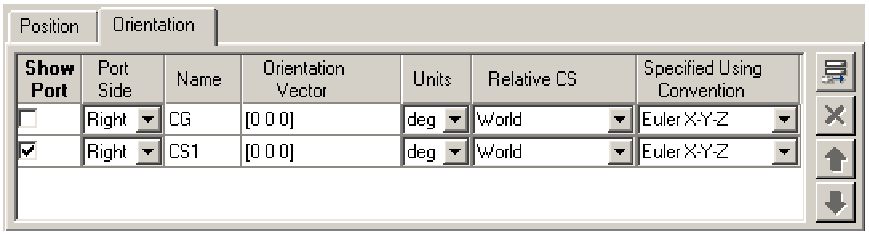
Specifying Body Coordinate Systems (Orientation). Enter the rotational orientation of the body and its Body CS axes in space:

1 Work on the **Orientation** pane. The default orientation for all CS axes is parallel to World. The sign of all rotations is determined by the *right-hand rule.*

   In the figure Equivalent Ellipsoid of Simple Pendulum with Body Coordinate Systems on page 2-17, the CS1 and CG axes are oriented parallel to the World axes, so the CS1 and CG axes need no rotation.

2 For both CSs, set the **Relative to CS** menu to `World`.

3 For CG and CS1, leave the **Orientation Vector** at default `[0 0 0]` and the **Specified Using Convention** at default `Euler X-Y-Z`. Close the Body dialog.



Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☐	Right	CG	[0 0 0]	deg	World	Euler X-Y-Z
✓	Right	CS1	[0 0 0]	deg	World	Euler X-Y-Z

Configuring a Joint Block

A machine is made up of Bodies with geometric and mass information. But Bodies carry no information of how they move. The possible directions of motion that a Body can take are called its *degrees of freedom* (DoFs), and this section explains how you represent these DoFs by Joint blocks:

- "How to Connect a Joint Between Two Bodies" on page 2-21
- "Revolute Joint for the Simple Pendulum" on page 2-21

DoFs Are Relative In SimMechanics, DoFs and the Joints that represent them are *relative* DoFs. That is, DoFs represent the possible motions between one body and another. So a DoF is defined by a pair of bodies, and you must connect every Joint to two and only two Bodies.

One (but not both) of the members of such a pair of Bodies can be a Ground. The other member Body of such a pair then has its motion defined relative to a fixed ground point. This fixed ground point does not have to be the same as the World origin. A machine can have many such Ground-Body pairs and *must have at least one.*

How to Connect a Joint Between Two Bodies

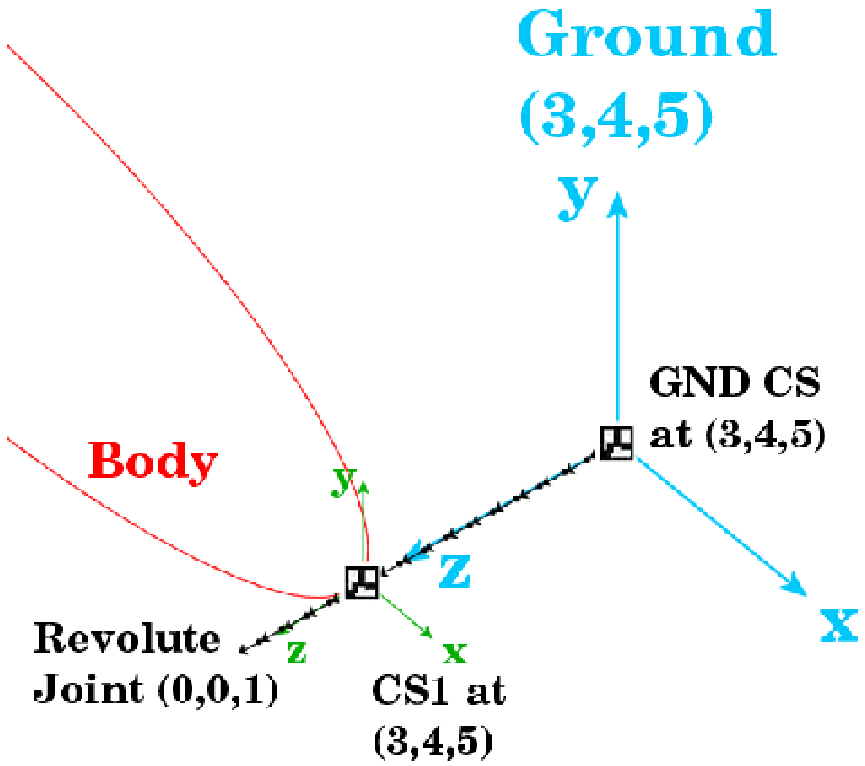
You represent relative motion of bodies with respect to one another by connecting their Body blocks with Joints. You can connect a Body to one or more Joints.

A Joint block is always connected to a specific point on the Body on either side of the Joint. The specific point for anchoring a Joint on a Body is the origin of a Body CS, and a Joint is therefore connected on one side to one Body at a Body CS origin, and on the other side to the other Body at a Body CS origin.

Usually a Body is connected to a Joint on either side, so the default you saw earlier in this tutorial for Body CSs in the Body dialog box is three Body CSs: the CS at the center of gravity (CG) and two other CSs (CS1 and CS2). But a Body at the end of a Body — Joint — ... — Body chain is connected to only one Joint.

Revolute Joint for the Simple Pendulum

In spite of the complexity of the concepts implicit in a Joint, the actual configuration of a Joint block is fairly simple. Here you insert and configure one revolute Joint block between the Ground and Body blocks you've already put into the model window.



A Simple Pendulum Connected to Ground by a Revolute

Configuring the Revolute Joint Block. The geometry of the Joint connection is shown in the figure preceding. The ground point at (3,4,5) and the CS1 at (3,4,5) are actually the same point in space, but have been separated in the figure for clarity. The revolute rotation axis is along the +*z* direction:

1. Open the Joints library in the block library.

2. Drag and drop a Revolute block into your model window.

3. Rotate the Revolute block so that you can connect the base (B) side of the Joint to the Ground block and the follower (F) side of the Joint to the Body block. Make the two connections.

4 Open the Revolute dialog box. In the **Parameters** area, on the **Axes** pane, configure the rotation axis to the World *z*-axis:

   a Enter `[0 0 1]` under **Axis of Action [x y z]**.

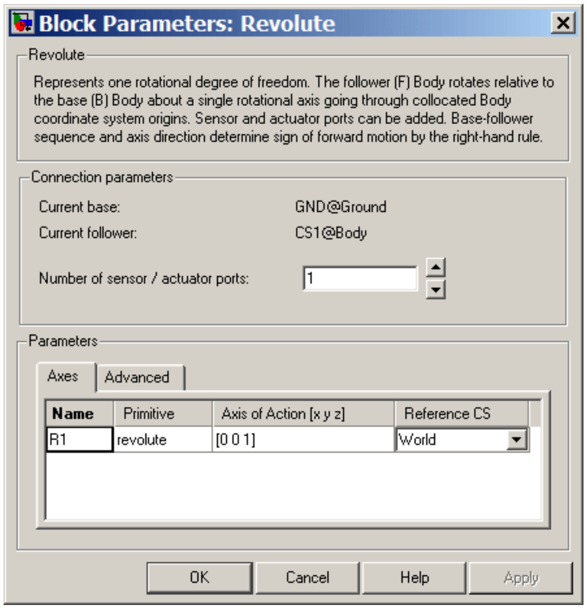
   b Leave the **Reference CS** at `WORLD`.

   c Ignore the **Advanced** tab.

   Note several important things:

   - Under **Connection parameters**, the **Current base** is located at `GND@Ground`, which is the Grounded CS associated with the Ground block located at (3,4,5) in World.
   - Under **Connection parameters**, the **Current follower** is located at `CS1@Body`, which is the CS1 on Body1 located at (3,4,5) in World.
   - This Joint's directionality runs from Ground to Body along the +*z*-axis.

5 Close the Revolute dialog box.


Block Parameters: Revolute

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base: GND@Ground

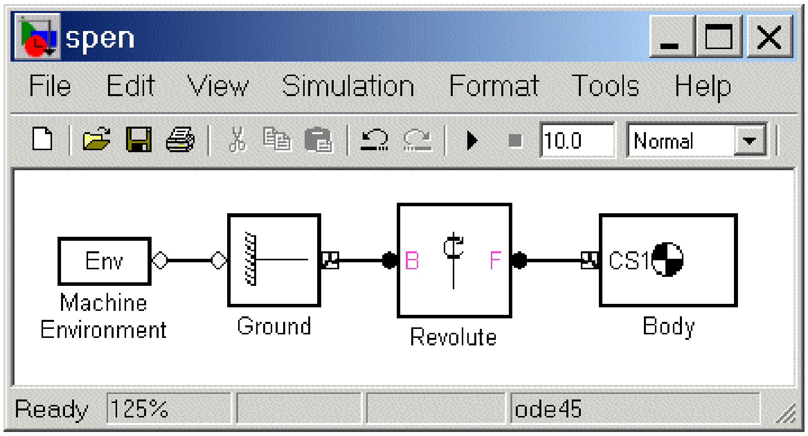
Current follower: CS1@Body

Number of sensor / actuator ports: 1

Parameters

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Congratulations — you have now finished the simplest possible model of a machine: a connected block diagram of Ground–Joint–Body. Your model window should look like this.



Adding a Sensor and Starting the Simulation

To measure the motion of the pendulum as it swings, you need to connect one or more Simulink Scope blocks to your model. The SimMechanics library of Actuators and Sensor blocks gives you the means to input and output Simulink signals to and from SimMechanics models. Sensors allow you to watch the mechanical motion once you start the simulation, as the following explain:

- "Connecting and Configuring the Pendulum Sensor" on page 2-24
- "Configuring the Machine Environment and Configuration Parameters" on page 2-26
- "Starting and Interpreting the Motion" on page 2-27

Connecting and Configuring the Pendulum Sensor

In this example, you measure the angular motion of the revolute joint:

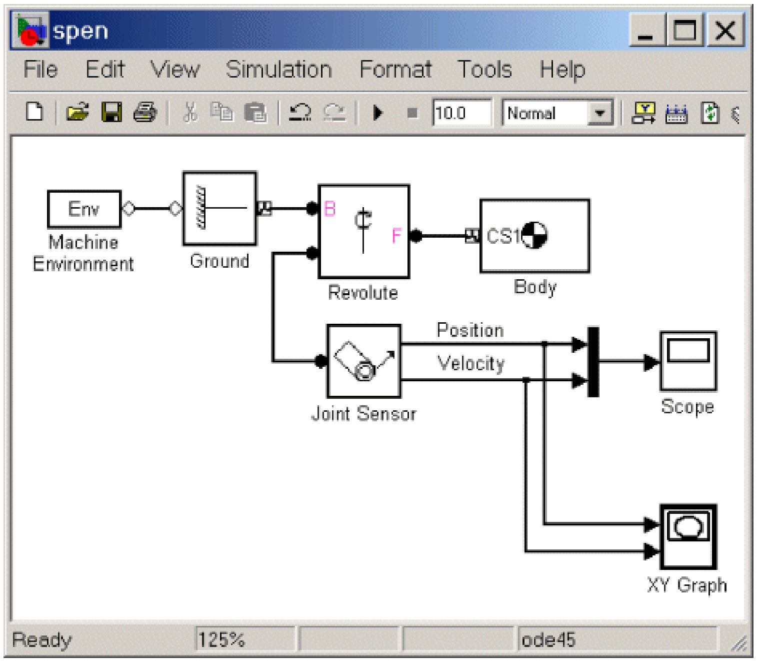
1 In the block library, open the Sensors and Actuators library. Drag and drop a Joint Sensor block into your model window.

2 Open the Revolute block. Change **Number of sensor/actuator ports** from 0 to 1 using the spinner menu. An open connector port ○ appears on the side of Revolute. Close Revolute.

3 Connect this connector port to the connector port on the Joint Sensor block. The open connector port changes to solid ●.

4 Open Joint Sensor. Select the **Angle** and the **Angular velocity** check boxes. Leave the other defaults. Close the Joint Sensor block.

5 Open the Simulink Library Browser. From the Sinks library, drag and drop a Scope block and an XY Graph block into your model window. From the Signal Routing library, drag and drop a Mux block as well. Connect the Simulink outports > on the Joint Sensor block to the Scope and XY Graph blocks as shown.



The lines from the outports > to the Scope and XY Graph blocks are normal Simulink signal lines and can be branched. You cannot branch the lines connecting SimMechanics blocks to each another at the round connector ports ●.

6 Save your model for future reference as `spen.mdl`.

You now need to configure the global parameters of your model before you can run it.

Configuring the Machine Environment and Configuration Parameters

The **Configuration Parameters** dialog box is a standard feature of Simulink. Reset its entries for this model to obtain more accurate simulation results.

1 In the Simulink menu bar, open the **Simulation** menu and click **Configuration Parameters** to open the **Configuration Parameters** dialog.

2 Select the **Solver** node of the dialog. Under **Solver options**, change **Relative tolerance** to `1e-6` and **Absolute tolerance** to `1e-4`.

   If you want the simulation to stop after a finite time, change **Stop time** to a finite number. The pendulum period is approximately 1.6 seconds.

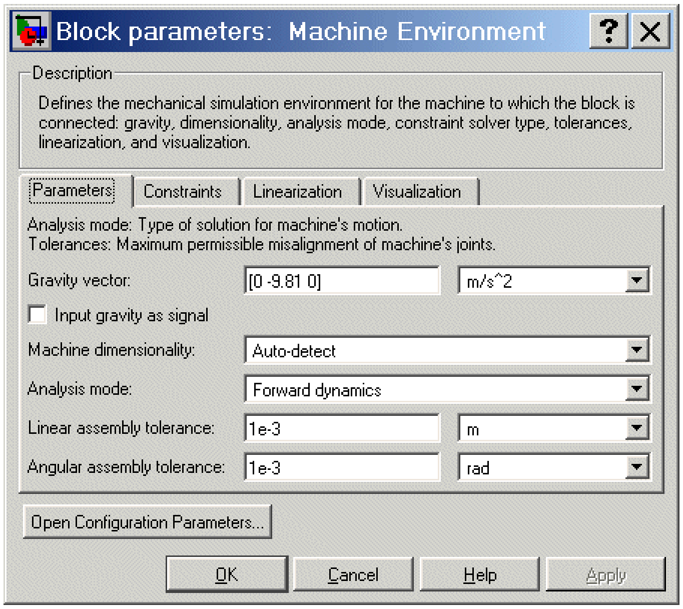
3 Close the **Configuration Parameters** dialog box.

A special feature of SimMechanics models is the Machine Environment block.

1 Open your block diagram's Machine Environment block dialog.

   Note the default **Gravity vector**, `[0 -9.81 0] m/s`^{2}, which points in the -*y* direction, as shown in the figure Equivalent Ellipsoid of Simple Pendulum with Body Coordinate Systems on page 2-17. The gravitational acceleration g = 9.81 m/s^2.

2 Close the Machine Environment dialog.



Block parameters: Machine Environment

Description

Defines the mechanical simulation environment for the machine to which the block is connected: gravity, dimensionality, analysis mode, constraint solver type, tolerances, linearization, and visualization.

Parameters | Constraints | Linearization | Visualization

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 -9.81 0] m/s^2

☐ Input gravity as signal

Machine dimensionality: Auto-detect

Analysis mode: Forward dynamics

Linear assembly tolerance: 1e-3 m

Angular assembly tolerance: 1e-3 rad

Open Configuration Parameters...

OK | Cancel | Help | Apply

Starting and Interpreting the Motion

You can now start your simulation and watch the pendulum motion via the Scope and XY Graph blocks:

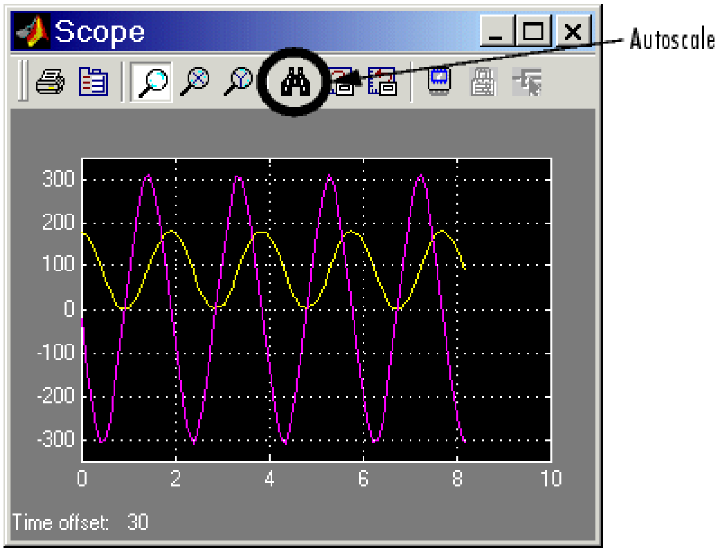
1 Open the XY Graph block dialog box. Set the following parameters.

Parameter	Value
`x-min`	0
`x-max`	200
`y-min`	-500
`y-max`	500

Leave **Sample time** at default and close the dialog.

2 Open the Scope block and start the model. The XY Graph opens automatically when you start the simulation.

3 View the full motion of both angle and angular velocity (in degrees and degrees per second, respectively) as functions of time in Scope. Click **Autoscale** if the motion is not fully visible.

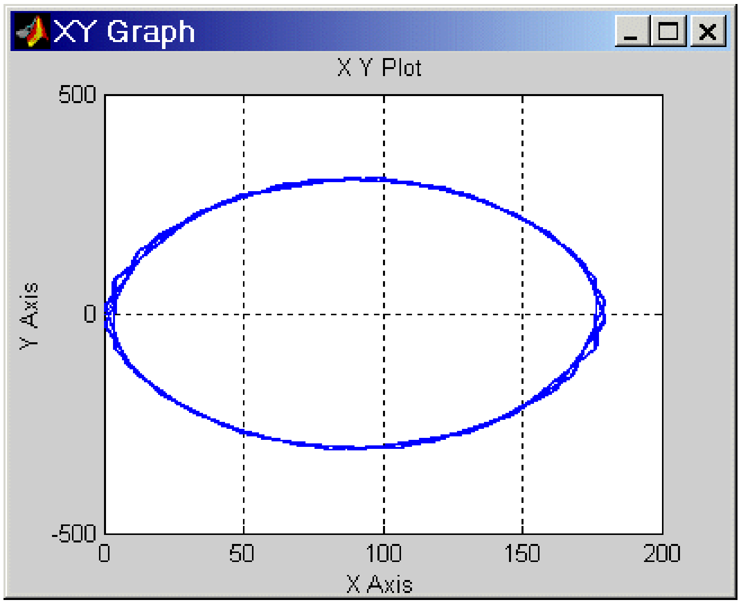


The motion is periodic but not simple harmonic (sinusoidal), because the amplitude of the swing is so large (180 degrees from one turning point to the other). Note that the zero of angle is the initial horizontal angle, not the vertical. The zeros of motion are always the initial conditions.

The XY Graph shows the angle versus angular velocity, with no explicit time axis. These two variables trace out a figure similar to an ellipse, because of the conservation of total energy E:

$$\frac{1}{2}J\left(\frac{d\theta}{dt}\right)^2 + mgh * (1 - \sin\theta) = E = \text{constant}$$

where $J = I_{zz} + mL^2/4$ is the inertial moment of the rod about its pivot point (not the center of gravity). The two terms on the left side of this equation are the kinetic and potential energies, respectively. The coordinate-velocity space is the *phase space* of the system.



Phase Space Plot of Simple Pendulum Motion: Angular Velocity Versus Angle

The directionality of the Revolute Joint assumes that the rotation axis lies in the +*z* direction. Looking at the pendulum from the front, follow the figures

- A Ground Point Relative to World on page 2-13
- Equivalent Ellipsoid of Simple Pendulum with Body Coordinate Systems on page 2-17
- A Simple Pendulum Connected to Ground by a Revolute on page 2-22

Positive angular motion from this perspective is counterclockwise, following the right-hand rule.

The next tutorial walks you through visualizing and animating this same simple pendulum model.

Visualizing a Simple Pendulum

In this section, you learn how to view the swinging pendulum rod of the model introduced in the last section using the SimMechanics visualization window. Use your saved `spen.mdl` model, or use the mech_spen model in the Demos library.

SimMechanics supports a customized MATLAB Graphics-based window for visualizing machines. This tool displays a machine by rendering its bodies. The bodies can be displayed in two ways, by equivalent ellipsoids and by closed surfaces (convex hulls) enveloping the bodies' coordinate systems.

Note You can find more on visualizing and animating machine models in Chapter 6, "Visualizing and Animating Machines".

Consult the MATLAB Graphics documentation for more about its standard features.

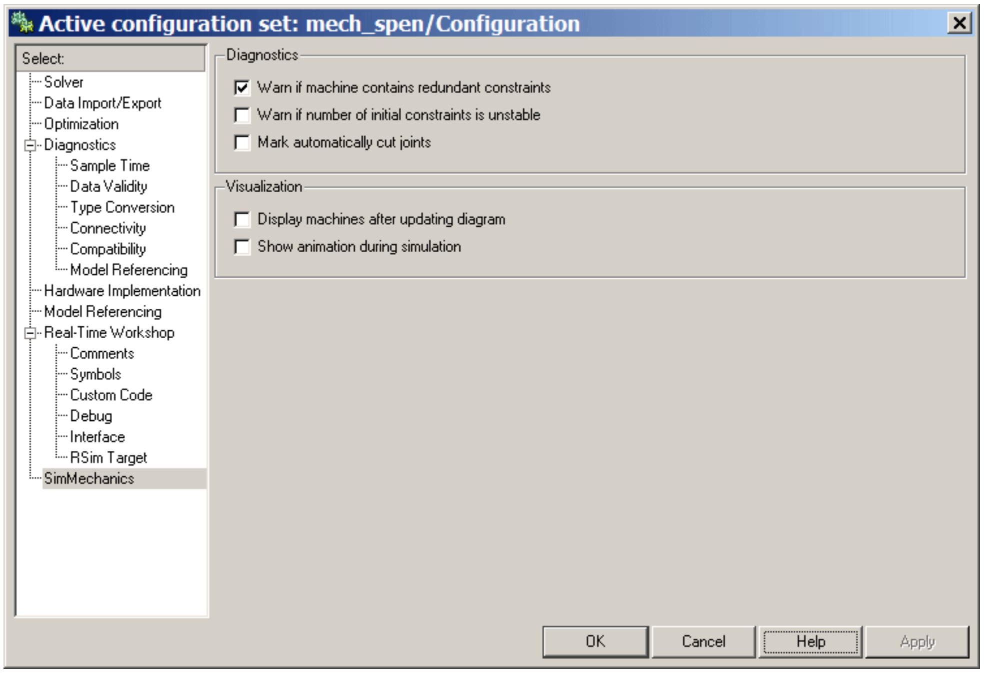
This section explains how to visualize your pendulum using either body rendering. You can view the pendulum before you start and, separately, choose to animate it during simulation as well:

- "Rendering the Bodies" on page 2-31
- "Visualizing with MATLAB Graphics" on page 2-32

Starting Visualization

The first step is to open the **Configuration Parameters** dialog from the **Simulation** menu of your model window:

1 On the Simulink menu bar, open the **Simulation** menu and select the **Configuration Parameters** entry. The **Configuration Parameters** dialog appears. Select the **SimMechanics** node at the lower left.



Diagnostics

- [✓] Warn if machine contains redundant constraints
- [ ] Warn if number of initial constraints is unstable
- [ ] Mark automatically cut joints

Visualization

- [ ] Display machines after updating diagram
- [ ] Show animation during simulation

2 To view the pendulum in its static initial state, select the **Display machines after updating diagram** check box.

To animate the pendulum visualization while the simulation is running, select the **Show animation during simulation** check box as well.

3 Click **OK**. Select **Update Diagram** from the **Edit** menu to open the visualization window.

Rendering the Bodies

The information that you use to specify body properties in a SimMechanics model is enough to render each body in certain simplified ways. SimMechanics does not have the information about the bodies needed to render their full geometries.

Equivalent Ellipsoids

A rigid body has a unique equivalent ellipsoid, a homogeneous solid ellipsoid with the same inertia tensor. For more about this rendering for rigid bodies, see “Rendering Body Shapes in SimMechanics” on page 6-5.

Because the rod has an axis of symmetry, the *x*-axis in this case, two of its three generalized radii are equal: $a_y = a_z$. The generalized radii of the equivalent ellipsoid are $a_x = \sqrt{5/3}(L/2) = 0.646$ m and $a_y = a_z = \sqrt{5}(r/2) =$ 1.12 cm.

Convex Hulls

Every Body has at least one Body coordinate system (CS) at the CG. A Body also has one or more extra Body CSs for the attached Joints, as well as possible Actuators and Sensors. Each Body CS has an origin point, and the collection of all these points, in general, defines a volume in space. The minimum outward-bending surface enclosing such a volume is the convex hull of the Body CSs, and this is the alternative way that SimMechanics has to render a body.

You created the pendulum body with only two Body CSs, CG and CS1. The convex hull for the pendulum rod is thus a line joining the two Body CS origins, the minimum connecting figure.

Choosing the Body Rendering

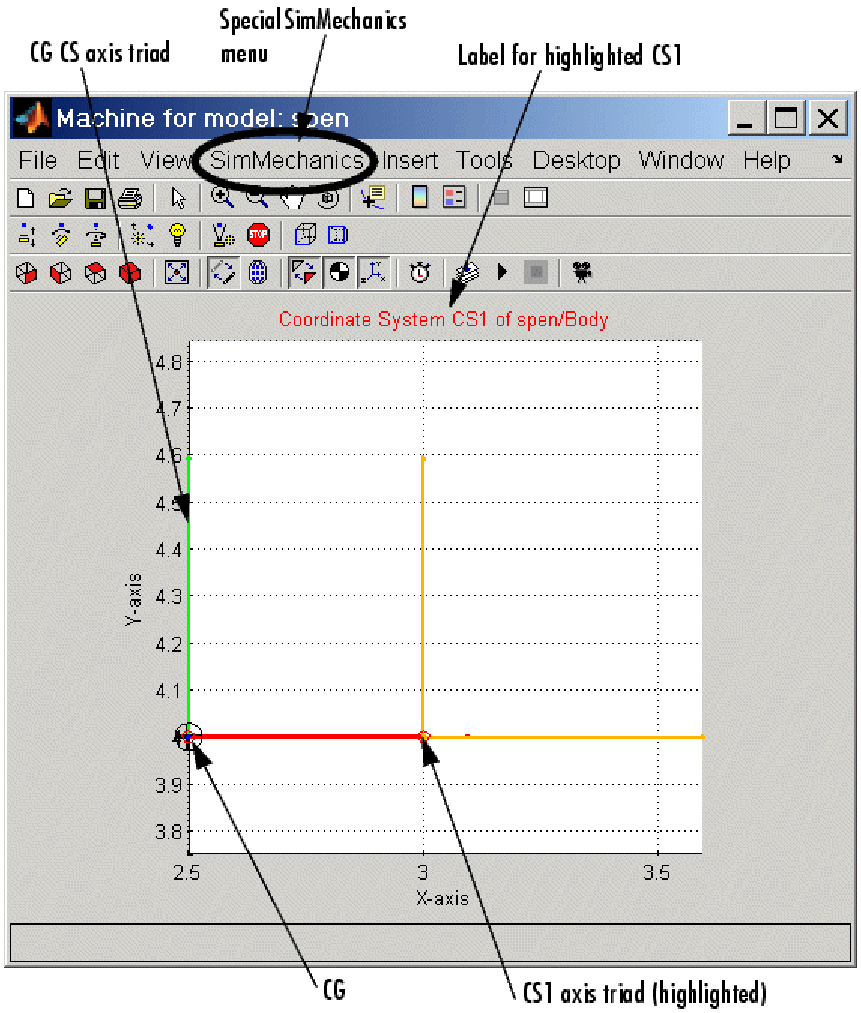
You can choose which displayed rendering of the pendulum rod or any machine bodies SimMechanics uses in the special **SimMechanics** menu of the visualization window. In the **Machine Display** submenu, choose **Convex Hulls** or **Ellipsoids**.

Visualizing with MATLAB Graphics

The MATLAB Graphics-based visualization tool is built into SimMechanics. To open it or to synchronize it at any time with your model, select **Update Diagram** in your model window’s **Edit** menu.

Rendering the Pendulum as a Convex Hull

The displayed figure depends on the body rendering you choose. If you chose **Convex Hulls** in the **SimMechanics > Machine Display** menu, a convex hull appears.


CG CS axis triad
Special SimMechanics menu
Label for highlighted CS1
Machine for model: spen
File Edit View SimMechanics Insert Tools Desktop Window Help
Coordinate System CS1 of spen/Body
4.8
4.7
4.6
4.5
4.4
4.3
4.2
4.1
3.9
3.8
Y-axis
2.5
3
3.5
X-axis
CG
CS1 axis triad (highlighted)

Pendulum Rod Rendered as a Convex Hull

You can change the viewpoint and manipulate the image using standard MATLAB Graphics techniques. Consult “Introducing the SimMechanics Visualization Window” on page 6-11 as well. Experiment with the **SimMechanics** menu’s settings to see various ways of displaying the pendulum.

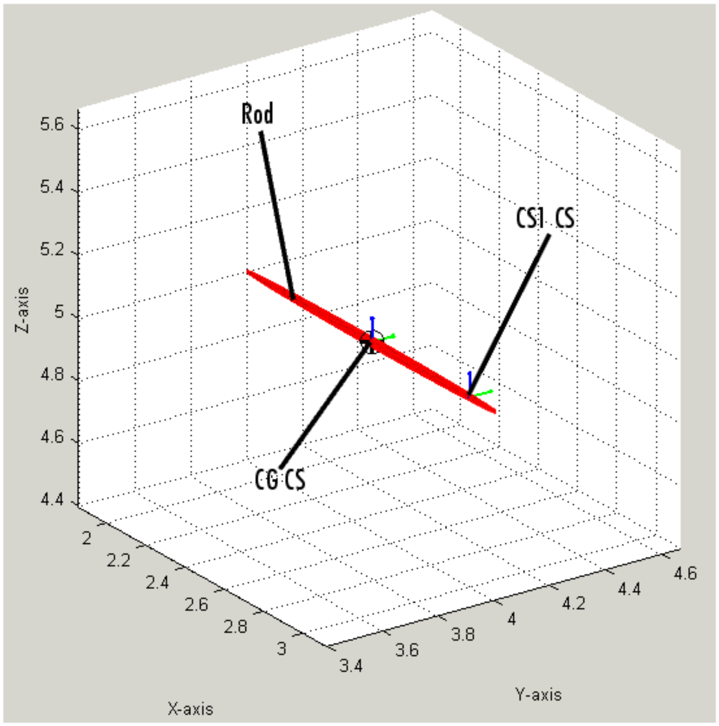
When you start the model, the body in the graphics window moves in step with the simulation.

Rendering the Pendulum as an Equivalent Ellipsoid

To render the pendulum as an equivalent ellipsoid, follow the previous steps, but change the **Machine Display** choice:

1. Open the **SimMechanics** menu and select **Machine Display**.
2. In the submenu, select **Ellipsoids**.

   The SimMechanics machine display changes. The equivalent ellipsoid looks like this.



Pendulum Rod Rendered as an Equivalent Ellipsoid

Modeling and Visualizing More Complex Machines

The next tutorial shows how to create, run, and visualize a model for a more complex machine, a four bar mechanism. To configure Ground, Body, and Joint blocks now means repeating and expanding upon the three blocks of the first two tutorials.

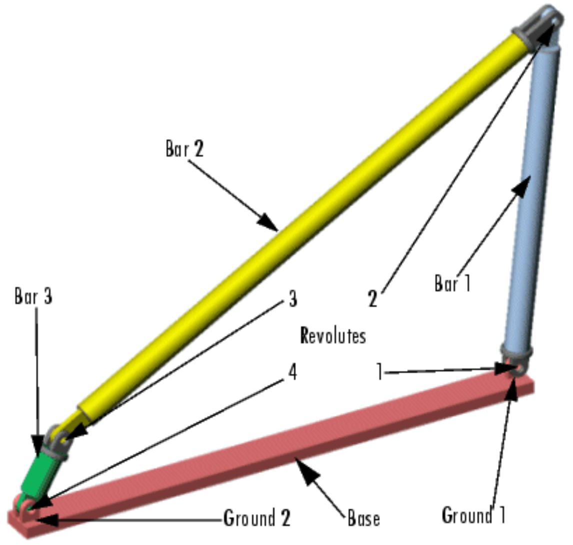
Four Bar Mechanism

In this tutorial, you build a model of a planar, four bar mechanism and practice using some of the important SimMechanics features:

- "Configuring the Mechanical Environment" on page 2-38
- "Setting Up the Block Diagram" on page 2-40
- "Configuring the Ground and Joint Blocks" on page 2-43
- "Configuring the Body Blocks" on page 2-47
- "Sensing Motion and Running the Model" on page 2-52

You are urged to work through "Building a Simple Pendulum" on page 2-11 and "Visualizing a Simple Pendulum" on page 2-30 before proceeding with this section. Learn more about how to position and orient bodies in Chapter 3, "Representing Motion".

The machine consists of three moving bars of homogeneous steel, two connected at one end each to ground points and a third crossbar connecting the first two. The base acts as an immobile fourth bar, with a Ground at each end. The machine forms a single closed loop, and its motion is confined to two dimensions.



A Four Bar Mechanism

The elementary parts of the machine are the bodies, while the revolute joints are the idealized rotational degrees of freedom (DoFs) at each body-to-body contact point. The bodies and the joints expressing the bodies' relative motions must be translated into corresponding SimMechanics blocks. If you want, you can add elaborations such as Constraints, Drivers, Sensors, and Actuators to this essential block diagram.

Counting the Degrees of Freedom

The three moving bars are constrained to move in a plane. So each bar has two translational and one rotational DoFs, and the total number of machine DoFs, before counting constraints, is 3*(2+1) = 9.

Because the motion of the bars is constrained, however, not all of these nine DoFs are *independent*:

- In two dimensions, each connection of a body with another body or with a ground point imposes two restrictions (one for each coordinate direction).

Such a restriction effectively eliminates one of the two body ends as independently moving points, because its motion is determined by the next body's end.

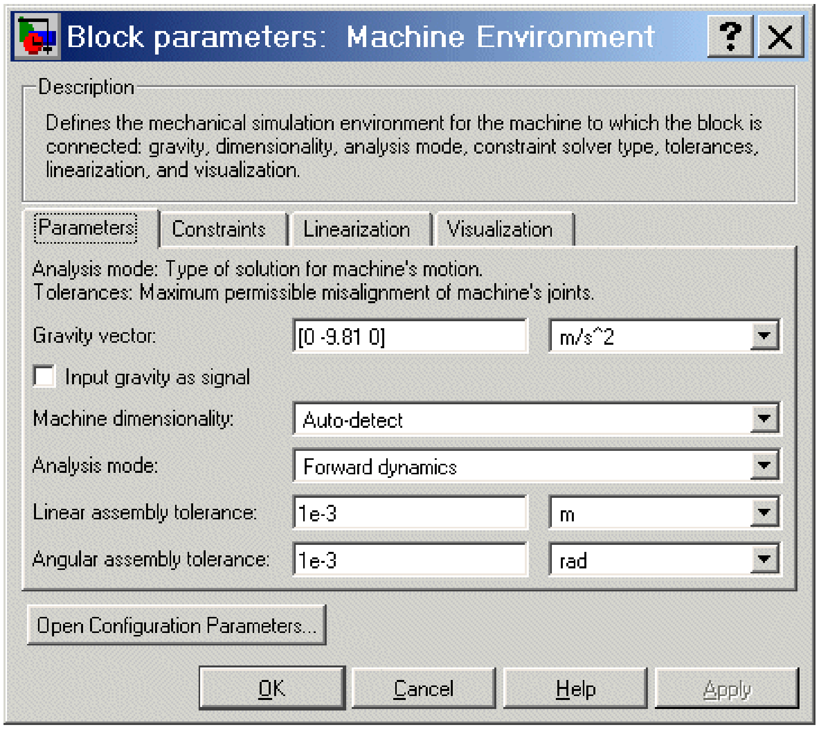
- There are four such body-body or body-ground connections and therefore eight restrictions implicit in the machine's geometry.

The eight restrictions on the nine apparent DoFs reduce the DoFs to one, 9 - 8 = 1. There are four rotational DoFs between bars or between bars and grounds. But three of these are dependent. Specifying the state of one rotational DoF fully specifies the other three.

Configuring the Mechanical Environment

Open a new blank model window from the SimMechanics library. From the Bodies library of SimMechanics, drag in and drop a Machine Environment block and a Ground block. Enable the Ground's Machine Environment port and connect the environment block to the Ground.

First you need to configure the machine's mechanical settings. Open the Machine Environment block. The block dialog box appears.


Block parameters: Machine Environment

Description

Defines the mechanical simulation environment for the machine to which the block is connected: gravity, dimensionality, analysis mode, constraint solver type, tolerances, linearization, and visualization.

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 -9.81 0] m/s^2

Input gravity as signal

Machine dimensionality: Auto-detect

Analysis mode: Forward dynamics

Linear assembly tolerance: 1e-3 m

Angular assembly tolerance: 1e-3 rad

The Machine Environment Dialog Box Panes

Click the four tabs in succession to display each pane.

Pane	Function
Parameters	Controls general settings for mechanical simulations
Constraints	Sets constraint tolerances and how constraints are interpreted
Linearization	Controls how SimMechanics models are linearized with Simulink
Visualization	Chooses whether or not to visualize the machine

Note some important features of this dialog box:

- The **Gravity vector** field specifies the magnitude and direction of gravitational acceleration and sets the vertical or up-down direction.
- The **Linear** and **Angular assembly tolerance** fields are also set here. Change **Angular assembly tolerance** to `1e-3 deg` (degrees). (See "Setting Assembly Tolerances" on page 5-5.)
- Leave the other defaults, except **Visualization**. In that pane, select the **Visualize machine** check box.

Close the dialog by clicking **OK**.

Starting Visualization

Note If possible, open the visualization window before building a model. With it, you can keep track of your model components and how they are connected, as well as correct mistakes.

To visualize the bodies as you build the machine, go to the **SimMechanics** node of the **Configuration Parameters** dialog:

1. Select the **Update machine visualization on update diagram** check box. If you want to animate the simulation later when you run the model, select the **Animate machine during simulation** check box as well. Click **OK** or **Apply**.

   Then select **Update Diagram** from the **Edit** menu. The window opens.

2. In the **SimMechanics** menu, select **Machine Display**, then **Ellipsoids**.

   As you add and change bodies in your model, you can update the machine display in your window at any time by updating your diagram.

Setting Up the Block Diagram

In this set of steps, you create Bodies, position them, connect them with Joints, then configure the Body and Joint properties. The Body dialog boxes give you many ways to represent the same machine in the same physical state. This section explains one way.

Alternative, equivalent ways of configuring Bodies are discussed in "Body Coordinate Systems" on page 4-13.

MAT-File Data Entry

The geometric and mass properties you need to specify for the Grounds and Bodies in this model are listed in the tables of the following two sections, "Configuring the Ground and Joint Blocks" on page 2-43 and "Configuring the Body Blocks" on page 2-47.

Instead of typing the numerical values of these properties into the dialog boxes, you can load the variable set you need into the workspace by entering

```
load fourbar_data
```

at the MATLAB command line. The variable name for each property is given in the tables. Just enter the appropriate variable names in the appropriate fields as you come to them in the dialog boxes.

Block Diagram Setup

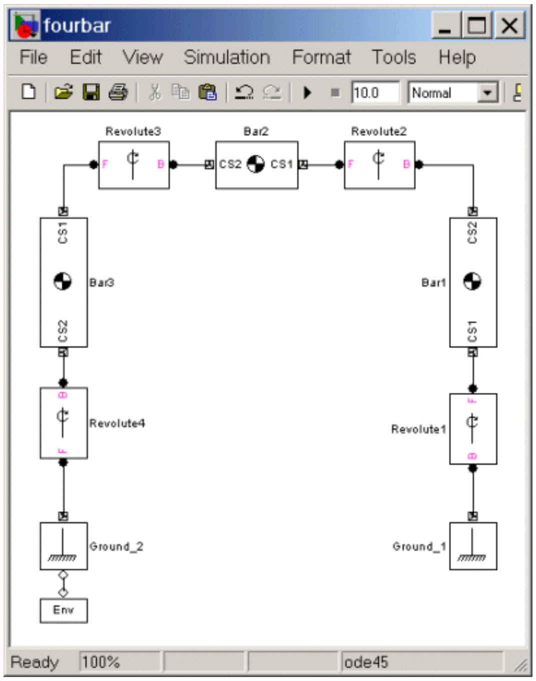
Your model already has one environment block and one ground block. Assemble the full model with these steps:

1. In the block library, open the Bodies library. Drag and drop another Ground block and three Body blocks into the new model window. Close the Bodies library.

2. From the Joints library, drag and drop four Revolute blocks into the model window.

3. Rotate and connect the blocks in the pattern shown in the following figure or with an equivalent block diagram topology.

   Use the block names shown in this figure for later consistency.



Connected Environment, Ground, Body, and Joint Blocks for the Four Bar

Block Diagram Topology. The topology of the block diagram is the connectivity of its elements. The elements are the Bodies and Grounds, connected by the Joints. Unlike the model of "Building a Simple Pendulum" on page 2-11, the four bar mechanism is a closed-loop machine. The two Ground blocks represent points on the same absolute, immobile body, and they close the loop of blocks. The simple pendulum has only one ground and does not close its block connections.

To maintain consistent Body motion direction, make sure the Body coordinate system (CS) port ⊠ pairs on each Body follow the sequence `CS1-CS2`, `CS1-CS2`, etc., for each bar, moving from Ground_1 to Ground_2, from right to left, as shown. To make the Joints consistent with the Body motion, the base-follower pairs `B-F`, `B-F`, etc., should follow the same right-to-left sequence.

Configuring the Ground and Joint Blocks

Now configure the Ground blocks with the data from the following table. Grounded coordinate systems (CSs) are automatically created.

Geometry of the Four Bar Base

This table summarizes the geometry of ground points.

Geometric Properties of the Four Bar Grounds

Property	Value	MAT-File Variable
Ground_1 point (m)	`[ 0.434 0 0.04 ]`	`gpoint_1`
Ground_2 point (m)	`[-0.433 0 0.04 ]`	`gpoint_2`

The base of the mechanism has these measurements:

- The base is horizontal, with length 86.7 cm.
- Ground_1 represents the ground point 43.3 cm to the right of the World CS origin.
- Ground_2 represents the ground point 43.4 cm to the left of the World CS origin.
- The bottom revolutes are 4 cm above the origin (x-z) plane.

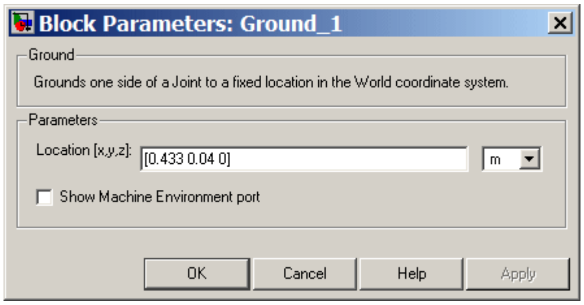
Setting Up the Grounds

To represent ground points on the immobile base, you need to configure the Ground blocks. Use the variable names if you've loaded `fourbar_data.mat` into your workspace:

1. Open Ground_1 and enter `[ 0.434 0 0.04 ]` or `gpoint_1` in the **Location** field.

2. Open Ground_2 and enter `[-0.433 0 0.04 ]` or `gpoint_2` in the **Location** field.

3. Leave both pull-down menus for units at default `m` (meters).


Block Parameters: Ground_1

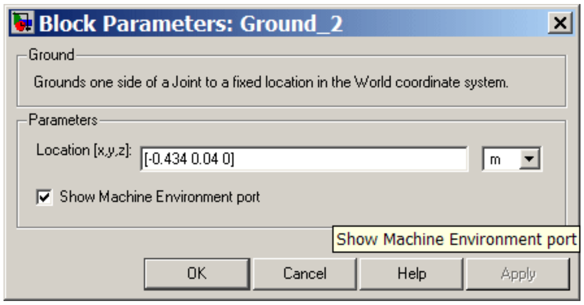
Ground

Grounds one side of a Joint to a fixed location in the World coordinate system.

Parameters

Location [x,y,z]: [0.433 0.04 0] m

☐ Show Machine Environment port


Block Parameters: Ground_2

Ground

Grounds one side of a Joint to a fixed location in the World coordinate system.

Parameters

Location [x,y,z]: [-0.434 0.04 0] m

☑ Show Machine Environment port

Configuring the Revolute Joints

The three nongrounded bars move in the plane of your screen (*x*-*y* plane), so you need to make all the Revolute axes the *z*-axis (out of the screen):

1 Open each Revolute's dialog box in turn. In its **Parameters** area, note on the **Axes** pane that the *z*-axis is the default: **Axis of Action** is set to `[0 0 1]` in each, relative to **Reference CS** `WORLD`. Leave these defaults.

   A Revolute block contains only one primitive joint, a single revolute DoF. So the **Primitive** is automatically `revolute`. Its name within the block is `R1`.

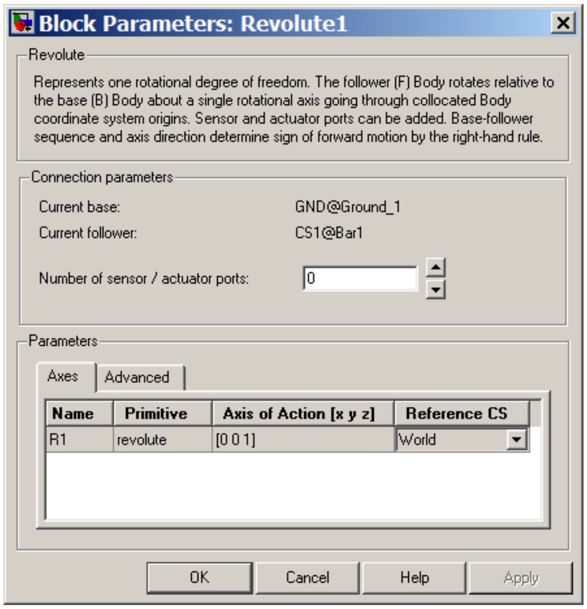
2 Leave these Revolute joint block defaults and ignore the **Advanced** tab.

The Body CS and base-follower joint directionality should be set up as shown in the block diagram of the figure Connected Environment, Ground, Body, and Joint Blocks for the Four Bar on page 2-42. In the **Connection parameters** area, the default Joint directionality for each Revolute automatically follows the right-to-left sequence of Grounded and Body CSs:

- Revolute1: Base to follower: `GND@Gound_1` to `CS1@Bar1`
- Revolute2: Base to follower: `CS2@Bar1` to `CS1@Bar2`
- Revolute3: Base to follower: `CS2@Bar2` to `CS1@Bar3`
- Revolute4: Base to follower: `CS2@Bar3` to `GND@Ground_2`

In this Joint directionality convention,

- At each Joint, the leftward Body moves relative to the rightward Body.
- The rotation axis points in the +*z* direction (out of the screen).
- Looking at the mechanism from the front in the figure, A Four Bar Mechanism on page 2-37, the positive rotational sense is counterclockwise. All Joint Sensor and Actuator data are interpreted in this sense.



Block Parameters: Revolute1

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

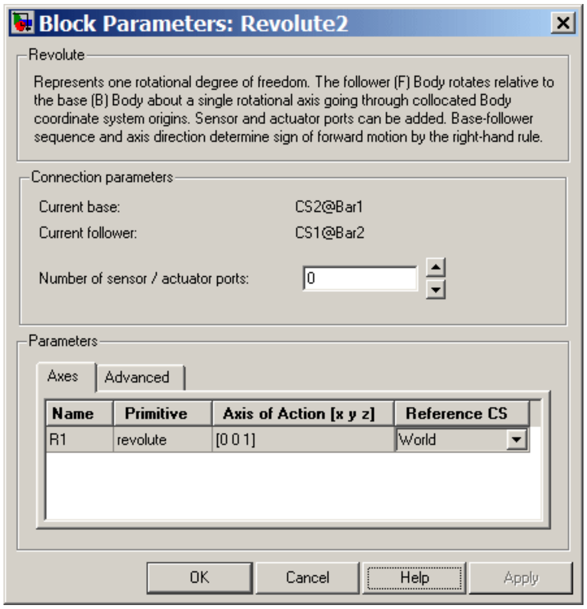
Connection parameters

Current base:	GND@Ground_1
Current follower:	CS1@Bar1
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World


Block Parameters: Revolute2

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base: CS2@Bar1

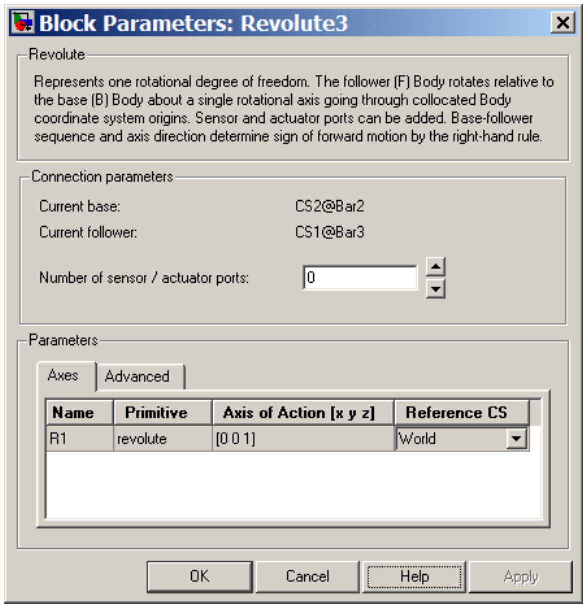
Current follower: CS1@Bar2

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World


Block Parameters: Revolute3

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection parameters

Current base: CS2@Bar2

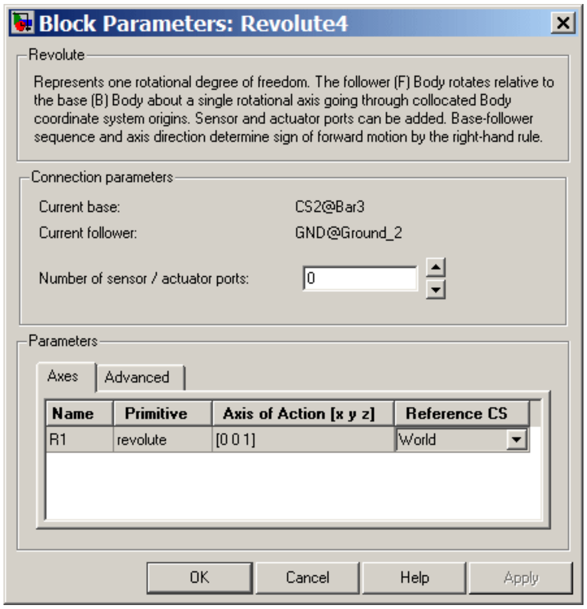
Current follower: CS1@Bar3

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World



Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Current base: CS2@Bar3
Current follower: GND@Ground_2
Number of sensor / actuator ports: 0

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Configuring the Body Blocks

Setting the Body properties is similar for each bar, but with different parameter values entered into each dialog box:

- Mass properties
- Lengths and orientations
- Center of gravity (CG) positions
- Body coordinate systems (CSs)

In contrast to the first tutorial, where you specify Body CS properties with respect to the absolute World CS, in this tutorial, you specify Body CS origins on the bars in relative coordinates, displacing Bar1's CS1 relative to Ground_1, Bar2's CS1 relative to Bar1, and so on, around the machine loop. You can refer the definition of a Body CS to three types of coordinate systems:

- To World
- To the other Body CSs on the same Body

- To the Adjoining CS (the coordinate system on a neighboring body or ground directly connected by a Joint to the selected Body CS).

The components of the displacement vectors for each Body CS origin continue to be oriented with respect to the World axes. The rotation of each Body's CG CS axes is also with respect to the World axes, in the Euler X-Y-Z convention.

The following three tables summarize the body properties for the three bars.

Bar1 Mass and Body CS Data (MKS Units)

Property	Value	Variable Name
Mass	`5.357`	`m_1`

Inertia tensor	`[1.07e-3 0 0;`
	`0 0.143 0;`
	`0 0 0.143]`
		`inertia_1`

CG Origin	`[0.03 0.282 0]` from `World` in axes of `CS1`	`cg_1`
CS1 Origin	`[0 0 0]` from `World` in axes of `Adjoining`	`cs1_1`
CS2 Origin	`[0.063 0.597 0]` from `World` in axes of `CS1`	`cs2_1`
CG Orientation	`[0 0 83.1]` from `WORLD` in convention `Euler X-Y-Z`	`orientcg_1`

Bar2 Mass and Body CS Data (MKS Units)

Property	Value	Variable Name
Mass	`9.028`	`m_2`

Inertia tensor	`[1.8e-3 0 0;`
	`0 0.678 0;`
	`0 0 0.678]`
		`inertia_2`

CG Origin	`[-0.427 0.242 0]` from `World` in axes of `CS1`	`cg_2`

Bar2 Mass and Body CS Data (MKS Units) (Continued)

Property	Value	Variable Name
CS1 Origin	`[0 0 0]` from `World` in axes of `Adjoining`	`cs1_2`
CS2 Origin	`[-0.87 0.493 0]` from `World` in axes of `CS1`	`cs2_2`
CG Orientation	`[0 0 29.5]` from `WORLD` in convention `Euler X-Y-Z`	`orientcg_2`

Bar3 Mass and Body CS Data (MKS Units)

Property	Value	Variable Name
Mass	`0.991`	`m_3`

Inertia tensor	`[2.06e-4 0 0;`
	`0 1.1e-3 0;`
	`0 0 1.1e-3]`
		`inertia_3`

CG Origin	`[-0.027 -0.048 0]` from `World` in axes of `CS1`	`cg_3`
CS1 Origin	`[0 0 0]` from `World` in axes of `Adjoining`	`cs1_3`
CS2 Origin	`[0 0 0]` from `World` in axes of `Adjoining`	`cs2_3`
CG Orientation	`[0 0 60]` from `World` in convention `Euler X-Y-Z`	`orientcg_3`

Configuring the Bodies

Here are the common steps for configuring the Body dialogs of all three bars. See the three preceding tables for Body dialog box mass property (mass and inertia tensor) entries. The units are MKS: lengths in meters (m), masses in kilograms (kg), and inertia tensors in kilogram-meters^2 (kg-m^2).

1 Open all three Body dialogs for each bar. Enter the mass properties for each from the tables in the **Mass** and **Inertia** fields.

2 Now work in the Body coordinate systems area, the **Position** pane:

   a Set the **Translated from Origin of** menu, for each Body CS on each bar, to `WORLD`.

   b Leave units as default `m` (meters).

3 Set the Body CS properties for each Body CS on each bar from the data of the preceding tables:

   a Enter the Body CS origin position data for CG, CS1, and CS2 on each bar from the tables or from the corresponding MAT-file variables.

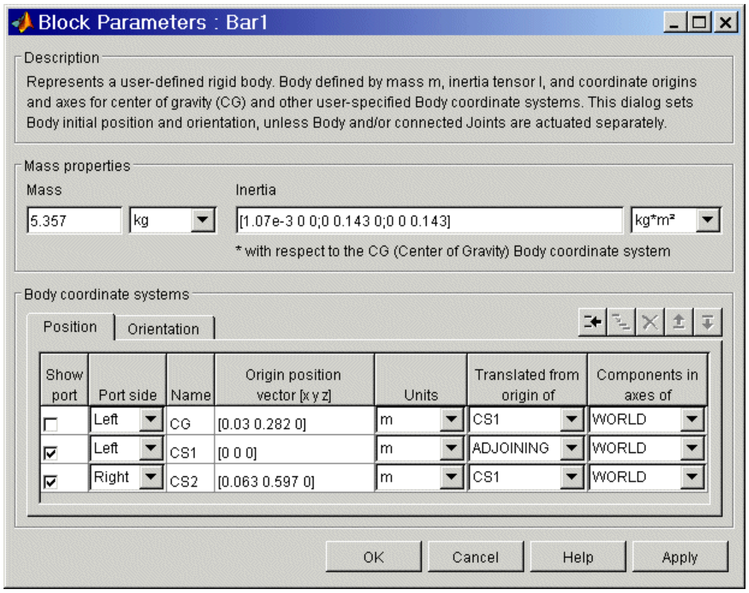
   b Set the **Components in Axes of** menu entries for each Body CS on each bar according to the values in the tables.

4 Select the **Orientation** pane by clicking its tab:

   a Enter the **Orientation Vector** for the CG on each bar from the tables or from the corresponding MAT-file variables.

   b Choose `WORLD` for **Relative CS** in each case.

   c Leave the other fields in their default values.


Block Parameters : Bar1

Description

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties

Mass: 5.357 kg

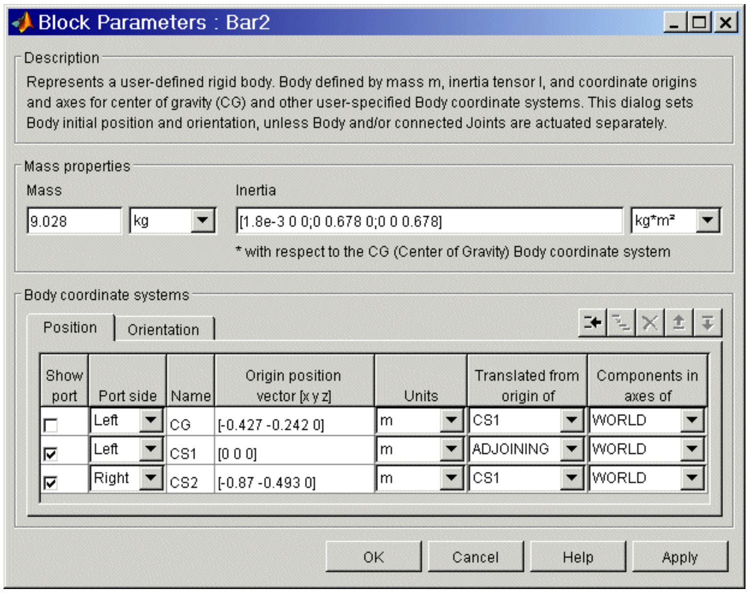
Inertia: [1.07e-3 0 0;0 0.143 0;0 0 0.143] kg*m²

* with respect to the CG (Center of Gravity) Body coordinate system

Body coordinate systems

Position | Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
☐	Left	CG	[0.03 0.282 0]	m	CS1	WORLD
☑	Left	CS1	[0 0 0]	m	ADJOINING	WORLD
☑	Right	CS2	[0.063 0.597 0]	m	CS1	WORLD


Block Parameters : Bar2

Description

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties

Mass: 9.028 kg

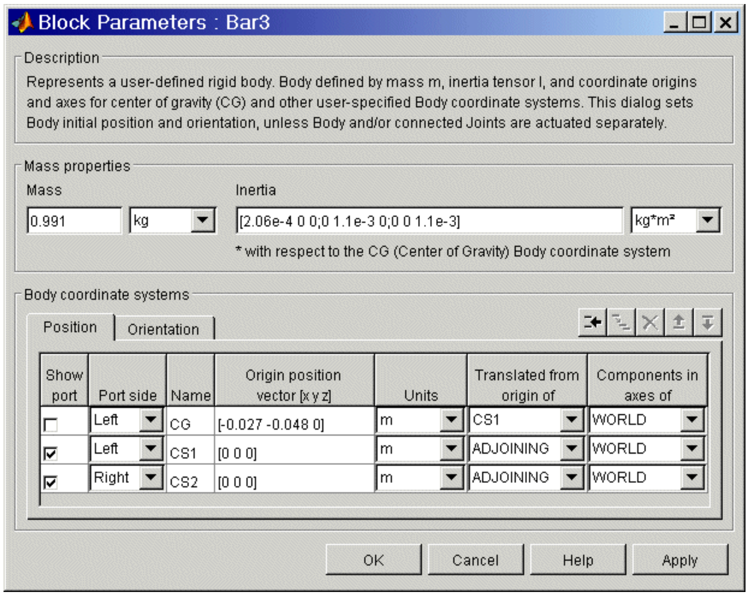
Inertia: [1.8e-3 0 0;0 0.678 0;0 0 0.678] kg*m²

* with respect to the CG (Center of Gravity) Body coordinate system

Body coordinate systems

Position | Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
☐	Left	CG	[-0.427 -0.242 0]	m	CS1	WORLD
☑	Left	CS1	[0 0 0]	m	ADJOINING	WORLD
☑	Right	CS2	[-0.87 -0.493 0]	m	CS1	WORLD


Block Parameters : Bar3

Description

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties

Mass: 0.991 kg

Inertia: [2.06e-4 0 0;0 1.1e-3 0;0 0 1.1e-3] kg*m²

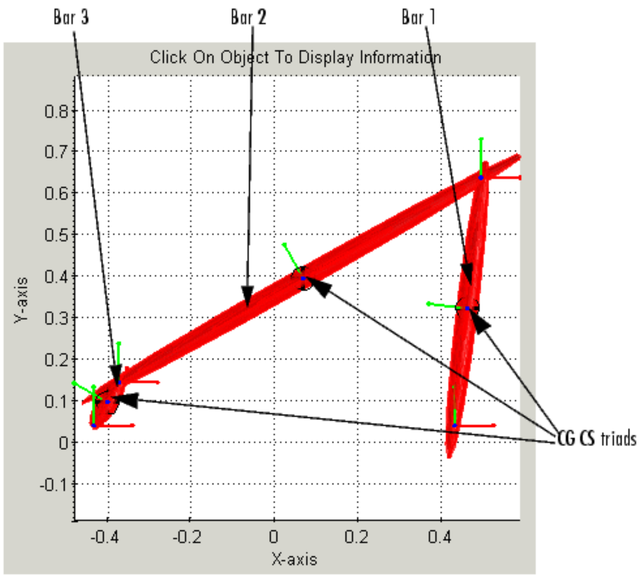
* with respect to the CG (Center of Gravity) Body coordinate system

Body coordinate systems

Position | Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
☐	Left	CG	[-0.027 -0.048 0]	m	CS1	WORLD
☑	Left	CS1	[0 0 0]	m	ADJOINING	WORLD
☑	Right	CS2	[0 0 0]	m	ADJOINING	WORLD

The front view of the four bar mechanism, with the bodies rendered as equivalent ellipsoids, looks like this:



Sensing Motion and Running the Model

You finish building your model by setting initial conditions and inserting Sensors.

Before you start a machine simulation, you need to set its kinematic state or initial conditions. These include positions/angles and linear/angular velocities. This information, the machine's initial kinematic state, is discussed further in "Kinematics and the Machine's State of Motion" on page 3-2 and "Modeling Actuators" on page 4-45.

You can sense motion in any model in two basic ways: sensing bodies or sensing joints. Here you sense Joint motion, using Joint Sensor blocks and feeding their Simulink signal outputs to Scope blocks.

Caution Because they are immobile, ground points cannot be moved, nor do they have any motion to measure.

Therefore, you cannot connect Ground blocks to Actuator or Sensor blocks.

Connecting the Joint Sensors

To sense the motion of the Revolute2 and Revolute3 blocks,

1. From the Sensors & Actuators library, drag and drop two Joint Sensor blocks into the model window. Drag Joint Sensor next to Revolute2 and Joint Sensor1 next to Revolute3.
2. Before you can attach a Joint Sensor block to a Revolute block, you need to create a new open round connector port ○ on the Revolute. Open Revolute2's dialog box:
   a. In the **Connection parameters** area in the middle, adjust the spinner menu **Number of sensor/actuator ports** to the value 1. Click **OK**.

      A new connector port ○ appears on Revolute2.
   b. Connect this connector port to the open round connector port on Joint Sensor.
3. Now repeat the same steps with Revolute3:
   a. Create one new connector port ○ on Revolute 3.
   b. Connect this port to Joint Sensor1.
4. Be sure to connect the outports > of the Sensor blocks to a Simulink Sink block. These outports are normal Simulink signals.

Graphical Plot of Joint Motion with a Scope Block

Here you can view the Joint Sensor measurements of Revolute2 and Revolute3's motions using a Scope block from the Simulink Sinks library:

1. Open the Simulink Library Browser. From the Sinks library, drag and drop a Scope block into your model window in between Joint Sensor and Joint Sensor1 blocks. Rename the Scope block “Angle.”

2. Open the Angle block. In this scope window’s toolbar, open the **Parameters** box. Under **Axes**, reset **Number of axes** to `2`. Click **OK**. A second inport > appears on the Angle block.

3. Expand the scope window for ease of viewing.

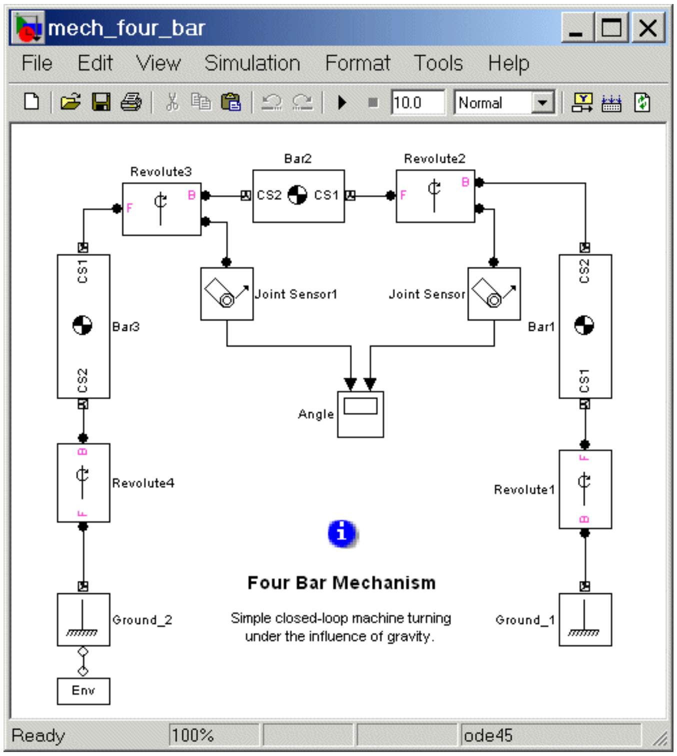
4. Connect the Joint Sensor and Joint Sensor1 block outports > to the Angle block inports >.

5. Open Joint Sensor and Joint Sensor1:

   a. In the **Measurements** area, **Connected to primitive** is set to `R1` in both blocks, indicating the first and only primitive revolute inside Revolute2 and Revolute3 to which each Sensor can be connected.

   b. Select the **Angle** check box to measure just the angle. Leave the units in default as `deg` (degrees). The Simulink line will contain one scalar.

Your completed model should look similar to the `mech_four_bar` demo model.



Caution Sensor and Actuator blocks are the *only* blocks that can connect SimMechanics blocks to normal Simulink blocks.

Configuring and Running the Simulation

Now take the final steps to prepare and start the model:

1 In the model window **Simulation** menu, select **Configuration Parameters**:

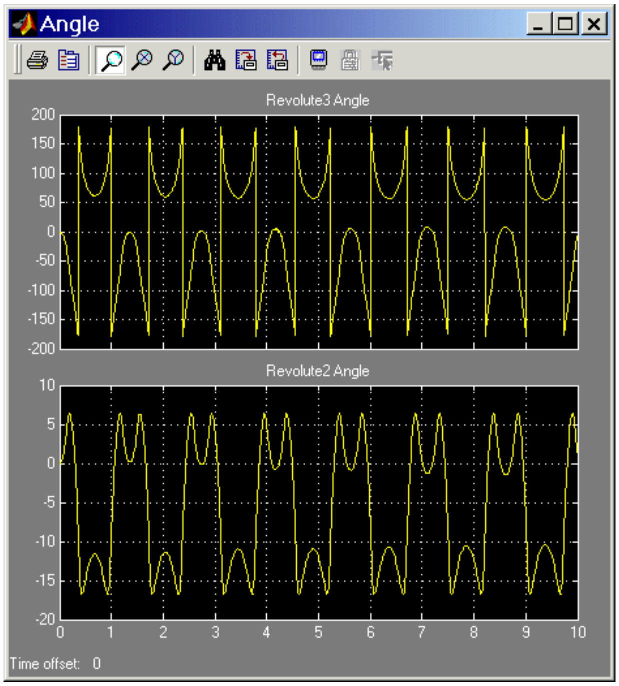
   a In the **Solver** node, change **Absolute tolerance** to `1e-6`.

   b Leave the other defaults and click **OK**.

2 Now run the model by clicking **Start** in the Simulink toolbar. The four bar machine will fall under the influence of gravity.

Note some features of the simulation:

- In this example, the machine starts from rest, with the initial velocities at zero. Thus the initial state of the four bar machine is just the geometric state that you set up in “Setting Up the Block Diagram” on page 2-40.
- The assembly at first falls over to the right, and the Revolute2 angle decreases.
- Bar3 turns all the way around, and Bar2 and Bar1 turn back to the left. The Revolute2 angle reverses direction. Revolute3 sweeps through a complete turn. Angles are mapped to the interval (-180º,+180º] and exhibit discontinuities.
- The motion repeats periodically, as there is no friction.



Animation

If you leave your visualization window open at the time you start the simulation and select the **Animate machine during simulation** check box in the **SimMechanics** node of the **Configuration Parameters** dialog, the visualized machine moves in step with the simulation.

You can now compare the animated motion with the Scope plots of the Revolute2 and Revolute3 angles.

For More About the Four Bar Machine

The four bar system is also discussed in the context of advanced SimMechanics features and methods: “Modeling Joints” on page 4-20, “Checking Model Validity” on page 4-74, “Finding Forces from Motions” on page 8-7, “Trimming Mechanical Models” on page 8-18, and “Linearizing Mechanical Models” on page 8-32.

Representing Motion

This chapter explains how SimMechanics represents body position, orientation, and motion. It connects mechanics concepts commonly used in physics and engineering with specific SimMechanics implementations. The last section is a case study on configuring a SimMechanics Body block to represent position and orientation.

This chapter assumes some familiarity with mechanics and vector analysis. You should work through it as a single unit. Consult “References” on page 3-4 for more.

Kinematics and the Machine's State of Motion

Kinematics is the description of a machine's motion without regard to forces, torques, and the mass properties of bodies. Because accelerations are proportional to forces and torques, if you know the mass properties of the bodies and the forces and torques applied to them, you need only the initial positions and their first derivatives (velocities) to integrate a machine's motion.

Degrees of Freedom

The relative position and orientation of a body with respect to a neighbor constitute up to six *degrees of freedom* (DoFs). The fundamental DoFs are translational (one body sliding relative to another along a prismatic axis) and rotational (one body rotating relative to another about a revolute axis, or one body pivoting relative to another about a spherical pivot point).

SimMechanics represents DoFs by Joint blocks connected between Body blocks. Bodies without Joints have no DoFs in SimMechanics and acquire DoFs only by having Joints connected to them. SimMechanics represents the machine's motion state by the positions (prismatics), angles (revolutes or sphericals), and their first derivatives with respect to time (velocities).

The State of Motion

The *state of motion* of a mechanical system is the set of the instantaneous positions and orientations of all its bodies and their linear and angular velocities. In SimMechanics, body positions/orientations are *relative*: one body's state is specified with respect to its neighbors. The absolute positions and velocities of the bodies' states are determined via the machine's connections to one or more grounds. These grounds are at rest in World, although they do not have to coincide with the World origin.

Home, Initial, and Assembled Configurations

When you start your model, SimMechanics configures your machines in preparation for motion by stepping them sequentially through three states.

- SimMechanics starts by analyzing the machines in their *home configurations*. A machine in its home configuration has all its bodies positioned and oriented purely according to the Body block dialog data. All body velocities are zero.

- From the model's initial condition actuators, SimMechanics then applies initial condition (position, orientation, and velocity) data to the joints of the model, changing its machines to their *initial configurations*.
- Finally, SimMechanics assembles any disassembled joints in the model, transforming the machines to their *assembled configurations*. While doing this, it holds fixed any positions and orientations specified by initial condition actuators.

  The assembled configuration is the final premotion machine state.

Updating your SimMechanics diagram (from the **Edit** menu or by pressing **Ctrl+D**) resets the model to its currently valid home configuration.

For More Information

For a detailed explanation of how to represent body motions, see "Body Motion in SimMechanics" on page 3-4.

Chapter 4, "Modeling Mechanical Systems" contains complete information on machine modeling in SimMechanics.

- "Modeling Rigid Bodies" on page 4-12
- "Modeling Joints" on page 4-20
- "Specifying Initial Positions and Velocities" on page 4-57
- "Counting Degrees of Freedom" on page 4-77

"How SimMechanics Works" on page 5-15 enumerates the complete SimMechanics simulation steps.

Refer also to the `mech_stateVectorMgr` command reference for identification of DoFs in SimMechanics and elaboration of the machine state.

Body Motion in SimMechanics

This section summarizes observer coordinate systems, measuring body motion, and how SimMechanics represents the state of a body's motion. It assumes a basic knowledge of vector algebra and analysis. Goldstein [2]; Murray, Li, and Sastry [3]; and Shuster [4] present coordinate transformations, rotations, and rigid body kinematics in detail. The preceding section, "Kinematics and the Machine's State of Motion" on page 3-2, should also be helpful.

How to Read This Section

Each topic in this section builds on the preceding one. Therefore, you should scan linearly through the whole section, then read it in detail.

- "Reference Frames and Coordinate Systems" on page 3-5
- "Relating Coordinate Systems in Relative Motion" on page 3-6
- "Observing Body Motion in Different Coordinate Systems" on page 3-7
- "Representing Body Translations and Rotations" on page 3-9

References

[1] Bell, E. T., "An Irish Tragedy: Hamilton (1805-1865)," in *Men of Mathematics,* New York, Simon & Schuster, 1937.

[2] Goldstein, H., *Classical Mechanics,* Second Edition, Reading, Massachusetts, Addison-Wesley, 1980.

[3] Murray, R. M., Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation,* Boca Raton, Florida, CRC Press, 1994.

[4] Shuster, M. D., "Spacecraft Attitude Determination and Control," in V. L. Piscane and R. C. Moore, eds., *Fundamentals of Space Systems,* New York, Oxford University Press, 1994.

Overview of Machine Motion

Machines are composed of bodies, which have relative degrees of freedom (DoFs). Bodies have positions, orientations, mass properties, and sets of Body coordinate systems. Joints represent the motions of the bodies.

- A machine's *geometry* consists of its static body features before starting a simulation: positions, orientations, and Body coordinate systems.
- A machine's *kinematics* consist of all degrees of freedom (DoFs) of all bodies: the positions/orientations and their derivatives of at any instant during the machine's motion.

The full description of a machine's motion includes not only its kinematics, but also specification of its observers, who define reference frames (RFs) and coordinate systems (CSs) for measuring the machine motion.

All vectors and tensors, unless otherwise noted, are represented by Cartesian matrices with three and nine, respectively, spatial components measured by rectangular coordinate axes.

Reference Frames and Coordinate Systems

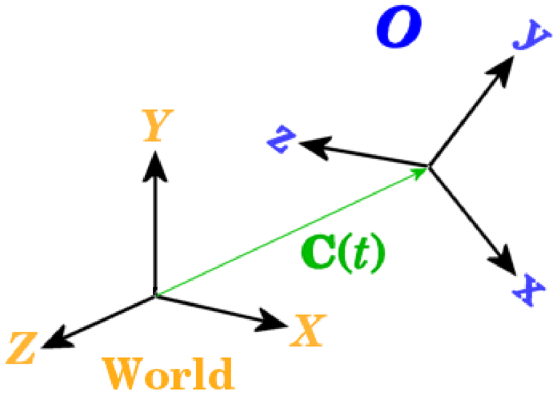
The *reference frame* of an observer is an observer's state of motion, which has to be measured by other observers. SimMechanics simulates a machine's motion using its Newtonian dynamics, which takes its simplest form in the special set of *inertial* RFs, the set of all frames unaccelerated with respect to inertial space. Within an RF, you can pick any point as a *coordinate system* origin, then set up Cartesian (orthogonal) axes there.

SimMechanics uses a master inertial RF called *World*. A CS origin and axis triad are also defined in World. World can mean either the RF or the CS, although in most contexts, it means the World coordinate system. For SimMechanics, World defines absolute rest and a universal coordinate origin and axes independent of any bodies and grounds in a machine.

A common synonym for coordinate system is *working frame*.

Relating Coordinate Systems in Relative Motion

Now add a second CS, called *O*, whose origin is translating with respect to the World origin and whose axes are rotating with respect to the World axes. Later in this section, this second CS is identified with a CS fixed in a moving body. (See "Representing Body Translations and Rotations" on page 3-9.)



A vector ***C*** represents the origin of *O*. Its head is at the *O* origin and its tail is at the World origin. The *O* origin moves as an arbitrary function of time ***C****(t)*.

The orthogonal unit vectors {***u****(x)*, ***u****(y)*, ***u****(z)*} define the coordinate axes of *O*.

- This set is oriented with respect to the World coordinate axes *X*, *Y*, *Z*, with unit vectors {***e****(x)*, ***e****(y)*, ***e****(z)*}. The orientation changes with time.
- You can express the set {***u****(x)*, ***u****(y)*, ***u****(z)*} as a linear combination of the basis {***e****(x)*, ***e****(y)*, ***e****(z)*} in terms of nine coefficients. These are relationships between *vectors* (not vector *components*) and are independent of the reference frame and coordinate system.

$$\mathbf{u}(\mathrm{x}) = R_{\mathrm{xx}}\mathbf{e}(\mathrm{x}) + R_{\mathrm{yx}}\mathbf{e}(\mathrm{y}) + R_{\mathrm{zx}}\mathbf{e}(\mathrm{z})$$
$$\mathbf{u}(\mathrm{y}) = R_{\mathrm{xy}}\mathbf{e}(\mathrm{x}) + R_{\mathrm{yy}}\mathbf{e}(\mathrm{y}) + R_{\mathrm{zy}}\mathbf{e}(\mathrm{z})$$
$$\mathbf{u}(\mathrm{z}) = R_{\mathrm{xz}}\mathbf{e}(\mathrm{x}) + R_{\mathrm{yz}}\mathbf{e}(\mathrm{y}) + R_{\mathrm{zz}}\mathbf{e}(\mathrm{z})$$

- You obtain the *components* of the ***u***'s in World by projecting the ***u***'s on to the ***e***'s by scalar products. The time-dependent *R* coefficients represent the orientation of the ***u***'s with respect to the ***e***'s. You can use the labels (1,2,3) as equivalents for *(x,y,z)*.

$$u_x(x) = R_{xx}\ ,\ u_y(x) = R_{yx}\ ,\ u_z(x) = R_{zx}$$
$$u_x(y) = R_{xy}\ ,\ u_y(y) = R_{yy}\ ,\ u_z(y) = R_{zy}$$
$$u_x(z) = R_{xz}\ ,\ u_y(z) = R_{yz}\ ,\ u_z(z) = R_{zz}$$

- The *components* of any vector ***v*** measured in World are ***e***(i)·***v***. Represent them by a column vector, $\boldsymbol{v}_{\mathrm{World}}$. The components of ***v*** in *O* are ***u***(i)·***v***. Represent them by a column vector, $\boldsymbol{v}_{\mathrm{O}}$. The two sets of components are related by the matrix transformation $\boldsymbol{v}_{\mathrm{World}} = R_{\mathrm{WO}}\cdot\boldsymbol{v}_{\mathrm{O}}$. The coefficients *R* form a matrix whose *columns* are the components of the ***u***'s in World:

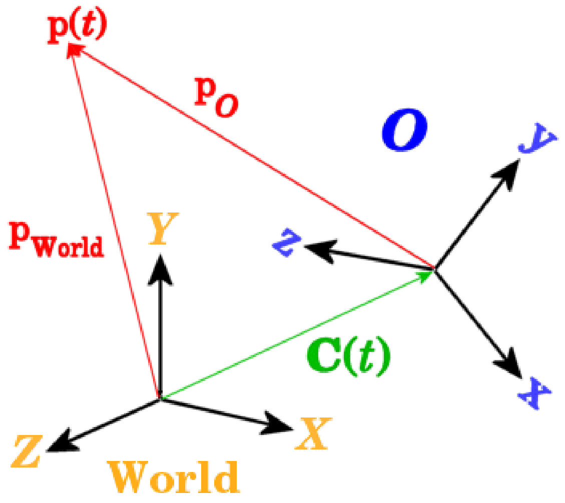
$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

  The orthogonality and unit length of the ***u***'s guarantee that *R* is an orthogonal rotation matrix satisfying $RR^T = R^TR = I$, the identity matrix. R^T is the transpose of *R* (switch rows and columns). Thus $R^{-1} = R^T$.

- Rotations always follow the *right-hand rule*, so that det(*R*) = +1.
- You use rotation matrices in general to transform the components of any vector from one CS representation to another, rotated CS representation.

Observing Body Motion in Different Coordinate Systems

To the two observer CSs, World and *O*, now add a third point ***p*** in arbitrary motion. ***p*** could represent a point mass, the center of gravity (CG) of an extended body, or a point fixed in a moving rigid body, for example. The two observers describe the motion of this point in different ways, related to one another by time-dependent World-to-*O* coordinate transformations.



The components of $\boldsymbol{p}$ are given by projecting it on to some CS axes. The components of $\boldsymbol{p}$ as measured in World are a column vector $\boldsymbol{p}_{\mathrm{World}}$ and, measured in O, are a column vector p_{O}. The two descriptions are related by

$$\boldsymbol{p}_{\mathrm{World}} = \boldsymbol{C}_{\mathrm{World}} + R \cdot \boldsymbol{p}_{\mathrm{O}}$$

Thus the motion as measured by $\boldsymbol{p}_{\mathrm{World}}$, when transformed and observed by O as p_{O}, has additional time dependence arising from the motion of C and R.

Relating Velocities Observed in Different Coordinate Systems

Differentiate the relationship between $\boldsymbol{p}_{\mathrm{World}}$ and $\boldsymbol{p}_{\mathrm{O}}$ once with respect to time. The result relates the velocity of $\boldsymbol{p}$ as measured by O to the velocity as measured in World.

$$\mathrm{d}\boldsymbol{p}_{\mathrm{World}}/\mathrm{dt} \;=\; \mathrm{d}\boldsymbol{C}_{\mathrm{World}}/\mathrm{dt} + R \cdot (\mathrm{d}\boldsymbol{p}_{\mathrm{O}}/\mathrm{dt}) + (\mathrm{d}R/\mathrm{dt}) \cdot \boldsymbol{p}_{\mathrm{O}}$$

The section “The Angular Velocity of a Body from Its Rotation Matrix” on page 3-9 explains how to express the third term in a simpler form.

Representing Body Translations and Rotations

Next consider the special case essential for describing the rigid body motions simulated by SimMechanics: the moving point $\boldsymbol{p}$ is fixed in the body itself. Let O be the center of gravity coordinate system (CG CS) of an extended rigid body (the origin of O at the CG itself) and let $\boldsymbol{p}$ be a point fixed somewhere in the same body. This body-fixed point is denoted by $\boldsymbol{b}$ in this special case. Because a moving body in general accelerates both translationally and rotationally, the CG CS is noninertial.

The rotation matrix R now describes the rotational motion of the body in terms of the rotation of the CG CS axes with respect to the World axes. Furthermore, because $\boldsymbol{b}$ is now fixed in the body itself, it does not move in O: $d\boldsymbol{b}_{\mathrm{O}}/dt = 0$. All of its motion as seen by World is due implicitly to the motion of R and C.

The Angular Velocity of a Body from Its Rotation Matrix

Continue to identify O with the body CG CS and $\boldsymbol{b}$ as a point fixed in the body. The vector components of $\boldsymbol{b}$ are observed by World as $\boldsymbol{b}_{\mathrm{World}}$ and by the CG CS as $\boldsymbol{b}_{\mathrm{Body}}$. In the body, the point is immobile: $d\boldsymbol{b}_{\mathrm{Body}}/dt = 0$. Its velocity observed by World is composed of the translational and rotational motion of the entire rigid body.

$$\mathrm{d}\boldsymbol{b}_{\mathrm{World}}/\mathrm{dt} = \mathrm{d}\boldsymbol{C}_{\mathrm{World}}/\mathrm{dt} + (\mathrm{d}R/\mathrm{dt})\cdot\boldsymbol{b}_{\mathrm{Body}}$$

Because $RR^{\mathrm{T}} = I$, $(dR/dt)*R^{\mathrm{T}} + R(*dR^{\mathrm{T}}/dt) = 0$. Insert $R^{\mathrm{T}}R = I$ to the left of $\boldsymbol{b}_{\mathrm{Body}}$ and define an antisymmetric matrix $\Omega = +(dR/dt)*R^{\mathrm{T}} = -R*(dR^{\mathrm{T}}/dt)$. Its components are $\Omega_{\mathrm{ik}} = +\Sigma_{\mathrm{j}}\ \varepsilon_{\mathrm{ijk}}\omega_{\mathrm{j}}$.

$$\Omega = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

where ω is the body's angular velocity in the World CS.

$$\begin{aligned}\mathrm{d}\boldsymbol{b}_{\mathrm{World}}/\mathrm{dt} &= \mathrm{d}\boldsymbol{C}_{\mathrm{World}}/\mathrm{dt} + \Omega\cdot R\cdot\boldsymbol{b}_{\mathrm{Body}} \\ &= \mathrm{d}\boldsymbol{C}_{\mathrm{World}}/\mathrm{dt} + \boldsymbol{\omega}\times(R\cdot\boldsymbol{b}_{\mathrm{Body}})\end{aligned}$$

The motion of $\boldsymbol{b}_{\text{Body}}$ decomposes into the motion of the body's CG plus the angular rotation of $\boldsymbol{b}_{\text{Body}}$ relative to the CG, all measured in World.

The relationship between time derivatives of a vector measured in World and measured in the body holds generally. For any vector V,

$$(\mathrm{d}\boldsymbol{V}/\mathrm{dt})_{\text{World}} = (\mathrm{d}\boldsymbol{V}/\mathrm{dt})_{\text{Body}} + \boldsymbol{\omega}_{\text{World}} \times \boldsymbol{V}$$

The derivative of the angular velocity ω is the angular acceleration. It is the same, whether measured in World or in the body, because $\boldsymbol{\omega} \times \boldsymbol{\omega} = 0$.

The Permutation Symbol ε and the Vector Cross Product

The permutation symbol ε_{ijk} is defined by

ε_{ijk}= +1 if *ijk* is an even permutation (123 or any cyclic permutation thereof)

ε_{ijk}= -1 if *ijk* is an odd permutation (321 or any cyclic permutation thereof)

ε_{ijk} changes sign upon switching any two indices and vanishes if any two indices are equal. The components of the cross (vector) product $\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}$ of two vectors ***a*** and ***b*** are

$$c_{\text{i}} = \Sigma_{\text{jk}}\, \varepsilon_{\text{ijk}} a_{\text{j}} b_{\text{k}}$$

How SimMechanics Represents Body Orientation

In SimMechanics, you represent a body's orientation by specifying the orientation of its center of gravity coordinate system (CG CS) axes relative to some other set of axes, either the CS axes of an adjoining body or the World CS axes. No reorientation is represented by "no rotation" or the rotational identity.

A general rotation of a body in three dimensions has three independent degrees of freedom. There are many ways to represent these degrees of freedom. SimMechanics uses the following representations in the Body and related Body Sensor and RotationMatrix2VR blocks. The block reference pages for these blocks discuss block-specific details.

- "Axis-Angle Representation" on page 3-11
- "Quaternion Representation" on page 3-12
- "Rotation Matrix Representation" on page 3-12
- "Euler Angle Representation" on page 3-13

The rotation representations are equivalent and convertible to one another:

- "Converting Rotation Representations" on page 3-14
- "Converting the Angular Velocity" on page 3-17

Axis-Angle Representation

The *axis-angle* representation of a rotation is the most fundamental form. Specify a rotation axis $\boldsymbol{n}$, then rotate by the right-hand rule about that axis by some angle θ. The vector $\boldsymbol{n} = (n_x,n_y,n_z)$ is a three-component unit vector, where $\boldsymbol{n}\cdot\boldsymbol{n} = 1$. The axis $\boldsymbol{n}$ is sometimes called the *eigenaxis*.

SimMechanics does not make direct use of the axis-angle representation, but it is the starting point for deriving other forms. It is also used extensively in mechanical applications such as computer-aided design and robotics.

The axis-angle representation is usually written as a 4-vector: $[n_x\ n_y\ n_z\ \theta]$. Of the four numbers, three are independent, because $\boldsymbol{n}\cdot\boldsymbol{n} = n_x^2 + n_y^2 + n_z^2 = 1$. That is, $\boldsymbol{n}$ specifies only a direction, not a length.

To describe continuous rotation in time, treat ***n*** and θ as functions of time.

Quaternion Representation

A *quaternion* represents a three-dimensional rotation as a four-component row vector of unit length:

$$q = \left[n_x \sin(\theta/2),\ n_y \sin(\theta/2),\ n_z \sin(\theta/2),\ \cos(\theta/2) \right] = [\boldsymbol{q}_{\mathrm{v}}, q_{\mathrm{s}}]$$

with $q{*}q = \boldsymbol{q}_{\mathrm{v}} \cdot \boldsymbol{q}_{\mathrm{v}} + q_{\mathrm{s}}^2 = 1$. This definition uses the axis-angle representation defined above. The rotation angle about that axis is θ. To describe continuous rotation in time, treat ***n*** and θ as functions of time. Unlike some rotation representations, quaternions never become singular.

See Bell [1] and Shuster [4] for more about quaternions.

Rotation Matrix Representation

The axis-angle representation also defines the *rotation matrix R* in exponential form $R = \exp(\theta\, \boldsymbol{n}\cdot\boldsymbol{J})$, where the J^k are real, antisymmetric matrices, and $\boldsymbol{n}\cdot\boldsymbol{J} = n_x J^1 + n_y J^2 + n_z J^3$. The rotation matrix R is orthogonal: $RR^T = R^T R = I$.

The J matrices are related to the antisymmetric permutation symbol ε_{ijk}.

$$\left(J^j\right)_{ik} = \varepsilon_{ijk}$$

The exponential R is reduced to closed form by the *Rodrigues identity*:

$$R = \exp(\theta \boldsymbol{n} \cdot \boldsymbol{J}) \ = \ I \ + \ (\boldsymbol{n} \cdot \boldsymbol{J}) \sin\theta \ + \ (\boldsymbol{n} \cdot \boldsymbol{J})^2 (1 - \cos\theta)$$

where I is the identity matrix, and $\boldsymbol{n}\cdot\boldsymbol{J}$ is given by

$$\mathbf{n} \cdot \boldsymbol{J} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

The inverse of R is identical to its transpose R^{T}. You can also obtain the inverse by replacing θ with θ or by reversing the direction of ***n***.

To describe continuous rotation in time, treat ***n*** and θ as functions of time.

Euler Angle Representation

An alternative representation for R is to rotate, in succession, about three independent axes, by three independent *Euler angles*. A full rotation R starting in *World* composes by multiplying the matrices successively on the *left*:

$$R_{\mathrm{BW}} = R_3 * R_2 * R_1$$

A full rotation R starting in a *body CS* composes by multiplying the matrices successively on the *right*:

$$R_{\mathrm{WB}} = R_1 * R_2 * R_3$$

The Euler angle convention is to

1. Rotate about one body coordinate axis (which rotates the other two).
2. Then rotate about a second body coordinate axis (rotated from its original direction) not identical to the first.
3. Lastly, rotate about another body coordinate axis not identical to the second.

Thus there are 3*2*2 = 12 possible Euler angle rotation sequences. The rotation axis sequences *Z-X-Z* and *Z-Y-X* are common. Rotation angles are often labeled as θ_1, θ_2, θ_3 or Φ, θ, Ψ as the first, second, and third angles, respectively. For example,

$$R_{\mathrm{BW}} = R_{\mathrm{X}}(\theta_1) * R_{\mathrm{Y}}(\theta_2) * R_{\mathrm{Z}}(\theta_3)$$

$$R_{\mathrm{WB}} = R_{\mathrm{Z}}(\Phi) * R_{\mathrm{X}}(\theta) * R_{\mathrm{Z}}(\Psi)$$

A two-dimensional rotation about a fixed axis requires one angle. For example, rotating the x- and y-axes about the z-axis by Φ is represented by

$$R_Z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

To describe continuous rotation in time, treat the Euler angles as functions of time. The Euler angle representation is singular in certain limiting situations. Such singularities are artifacts of the Euler angle form and have no geometric or physical significance.

Converting Rotation Representations

Certain SimMechanics blocks make use of different rotation representations.

- The Body block makes direct use of the Euler angle, rotation matrix, and quaternion representations.
- The Body Sensor block makes use of the rotation matrix.
- The RotationMatrix2VR block uses the rotation matrix and axis-angle forms.

The four rotation representations presented in this section are equivalent. You can represent a rotation equally well with any one of them. Some applications, however, tend to favor one representation over the others, and certain representations are singular in certain limits. It is helpful to know how to convert the various rotation representations into one another. The following summaries group the conversion formulas into one place.

Transforming the Axis-Angle Representation

The rotation axis unit vector ***n*** and the rotation angle θ define this representation, which is discussed in detail in "Axis-Angle Representation" on page 3-11. This representation defines the quaternion and rotation matrix representations:

$$q = \left[n_x \sin(\theta/2),\ n_y \sin(\theta/2),\ n_z \sin(\theta/2),\ \cos(\theta/2)\right] = [\boldsymbol{q}_\mathrm{v}, q_\mathrm{s}]$$

$$R = \exp(\theta \boldsymbol{n}\cdot\boldsymbol{J}) = I + (\boldsymbol{n}\cdot\boldsymbol{J})\sin\theta + (\boldsymbol{n}\cdot\boldsymbol{J})^2(1-\cos\theta)$$

$$\mathbf{n}\cdot\boldsymbol{J} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

Transforming the Quaternion Representation

The quaternion is a vector-scalar pair, $q = [q_v\ q_s]$, defined by "Quaternion Representation" on page 3-12. You can recover the axis-angle representation from the quaternion components:

$$\theta = 2\cdot\cos^{-1}(q_s)$$
$$\mathbf{n} = \boldsymbol{q}_v \Big/ \sqrt{1-{q_s}^2}$$

You can also construct the equivalent rotation matrix R from q.

$$R \ = \ (2q_s^2-1)I + 2q_sQ_v + 2\boldsymbol{q}_v^T \otimes \boldsymbol{q}_v$$
$$(Q_v)_{ik} = \sum\nolimits_j \varepsilon_{ijk}(\boldsymbol{q}_v)_j$$

The term $\boldsymbol{q}_v^T \otimes \boldsymbol{q}_v$ is the *outer product* of q_v with itself, the 3-by-3 matrix of q_v components multiplied by each other.

Transforming the Rotation Matrix Representation

The rotation matrix R is an orthogonal 3-by-3 matrix: $RR^T = R^TR = I$, defined in "Rotation Matrix Representation" on page 3-12. You can invert the rotation matrix representation to obtain the equivalent representations for the quaternion $q = [q_v\ q_s]$ and axis-angle ($\boldsymbol{n}$, θ

$$q_s = \tfrac{1}{2}\sqrt{Tr(R)+1}$$
$$\boldsymbol{q}_v = Tr(\boldsymbol{J} * R)\Big/\left(2\sqrt{Tr(R)+1}\right)$$
$$\theta = 2\cdot\cos^{-1}\left(\tfrac{1}{2}\sqrt{Tr(R)+1}\right)$$
$$\mathbf{n} = Tr(\boldsymbol{J} * R)\Big/\left(\sqrt{Tr(R)+1}\cdot\sqrt{3-Tr(R)}\right)$$

The trace *Tr* of a matrix is the sum of its diagonal elements.

The J matrices constitute a 3-vector of matrices defined by the antisymmetric permutation symbol, $(J^j)_{ik} = \varepsilon_{ijk}$. See "The Permutation Symbol ε and the Vector Cross Product" on page 3-10 for more details.

The RotationMatrix2VR block converts the rotation matrix to the axis-angle representation.

Transforming the Euler Angle Representation

The Euler angle representation of a rotation, defined by "Euler Angle Representation" on page 3-13, stands apart from the other three, insofar as you cannot derive it from the axis-angle representation. It depends on the choice of rotation axis sequence, which generates multiple definition conventions. The Euler angle representation, at certain limits, can also be singular. Use caution with Euler angle expressions.

If you choose a convention and three angles, then compute *R*, you can convert *R* to the other representations by the use of "Transforming the Rotation Matrix Representation" on page 3-15 above. But given the nine components of *R*, you must find the Euler angles by inverting the nine equations that result from this matrix equation. (Only three equations of the nine are independent.) In some cases, angles can be read from *R* by inspection.

For example, choose rotations with respect to a Body coordinate system (CS) triad, in a commonly used rotation axis sequence *Z*-*X*-*Z*, with Φ, θ, Ψ as the respective angles. The rotation matrix is $R_{\mathrm{WB}} = R_1(\Phi)*R_2(\theta)*R_3(\Psi)$,

$$R_{\mathrm{WB}}(\phi,\theta,\psi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\phi\cos\psi - \sin\phi\cos\theta\sin\psi & -\cos\phi\sin\psi - \sin\phi\cos\theta\cos\psi & 0 \\ \sin\phi\cos\psi + \cos\phi\cos\theta\sin\psi & -\sin\phi\sin\psi + \cos\phi\cos\theta\cos\psi & 0 \\ \sin\theta\sin\psi & \sin\theta\cos\psi & \cos\theta \end{pmatrix}$$

In this convention, you can read θ from the R_{33} component, then find Ψ from the R_{32} or R_{31} component. Obtain Φ from one of the other components, using $\cos^2\Phi + \sin^2\Phi = 1$, or by multiplying from the right by $R_3\Psi^{\mathrm{T}}$, then $R_2\theta^{\mathrm{T}}$. The second method yields a unique solution for the sine and cosine of Φ.

Converting the Angular Velocity

The rotation matrix R is defined in “Body Motion in SimMechanics” on page 3-4 and “Rotation Matrix Representation” on page 3-12.

The angular velocity vector ω is the rate at which a spinning CS rotates. R and the antisymmetric matrix Ω define ω:

$$\Omega = +(\mathrm{d}R/\mathrm{dt}) \cdot R^{\mathrm{T}} = -R \cdot \left(\mathrm{d}R^{\mathrm{T}}/\mathrm{dt}\right)$$

$$\Omega_{\mathrm{ik}} = +\sum\nolimits_{\mathrm{j}} \varepsilon_{\mathrm{ijk}} \omega_{\mathrm{j}}$$

$$\omega_{\mathrm{j}} = \left(\tfrac{1}{2}\right) \sum\nolimits_{\mathrm{ik}} \varepsilon_{\mathrm{ijk}} \Omega_{\mathrm{ik}}$$

You can also express the angular velocity in terms of Euler angles, by choosing a particular set of angles to represent R. See “Euler Angle Representation” on page 3-13 and “Transforming the Euler Angle Representation” on page 3-16.

The quaternion derivative is also related to the angular velocity:

$$\mathrm{d}\boldsymbol{q}_{\mathrm{v}}/\mathrm{dt} = \left(\tfrac{1}{2}\right)\left(q_{\mathrm{s}} \boldsymbol{\omega}_{\mathrm{Body}} - \boldsymbol{q}_{\mathrm{v}} \times \boldsymbol{\omega}_{\mathrm{Body}}\right)$$

$$\mathrm{d}q_{\mathrm{s}}/\mathrm{dt} = -\left(\tfrac{1}{2}\right)(\boldsymbol{q}_{\mathrm{v}} \cdot \boldsymbol{\omega}_{\mathrm{Body}})$$

Orienting a Body and Its Coordinate Systems

This section shows you a set of examples of orienting a test body and its attached coordinate systems (CSs). It makes detailed use of the rotation representations and conversions explained in “How SimMechanics Represents Body Orientation” on page 3-11, and shows how to use MATLAB workspace variables in your SimMechanics block dialogs.

- “Setting Up the Test Body” on page 3-18
- “Rotating the Body and Its CG CS Relative to World” on page 3-20
- “Rotating the Body Relative to Its Center of Gravity” on page 3-22
- “Creating and Rotating Body Coordinate Systems” on page 3-23

This sequence of examples assumes you are familiar with the basics of setting up and visualizing bodies and machines. See the tutorials of Chapter 2, “Building and Visualizing Simple Machines” and the Body block reference. Look for complete instructions on modeling and visualizing machines in Chapter 4, “Modeling Mechanical Systems” and Chapter 6, “Visualizing and Animating Machines”.

Setting Up the Test Body

The later examples require a configured body to work with. Here you set up a simple body with one Body coordinate system (CS), located at the center of gravity (CG).

Initializing the Body

First, create, connect, and visualize the initial body:

1. Open the SimMechanics block library and a new Simulink model window.
2. Drag and drop a Machine Environment, a Ground, and a Body block, as well as any Joint block you want, into your model window.
3. Open the Body dialog. In the Body coordinate systems area, delete the Body CS named CS2. Deselect the **Show port** check box for CS1.

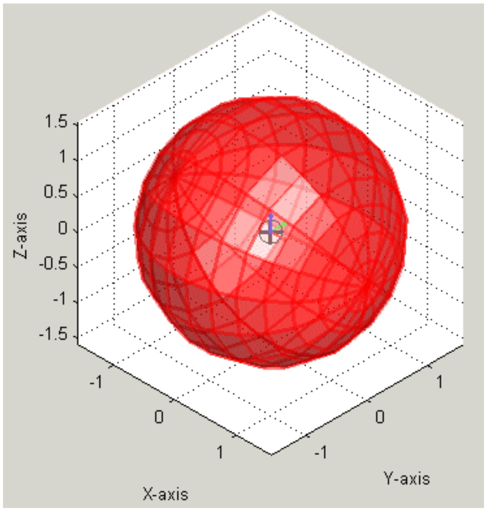
Leave the other defaults. The CG CS is located at **Origin position vector** `[0 0 0]`. Select **Show port** for the CG CS. Click **Apply**.

4 Open the Ground and select **Show Machine Environment port**. Click **OK**. Connect Ground to Machine Environment at the new Machine Environment port.

5 Connect the Body block at its CG CS port, through the Joint block, to Ground at its grounded CS port.

6 From the **Simulation** menu, open **Configuration Parameters** to the **SimMechanics** node. In the **Visualization** area, select the **Update machine visualization on update diagram** check box. Click **OK**.

7 From the **Edit** menu, select **Update Diagram**. When the window opens, open the **SimMechanics** menu, deselect **Convex Hulls**, and select **Ellipsoids**. The window now renders the body as a sphere.



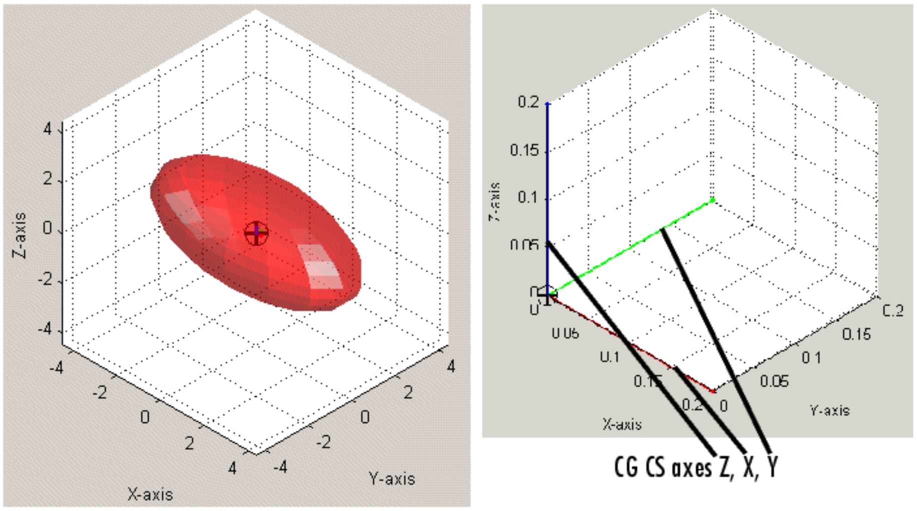
Configuring the Body

Now configure the body's mass and geometric properties:

1 In the Body dialog's **Mass properties** area, enter `11.5 kg` for the mass and `[18 0 0; 0 56 0; 0 0 56] kg*m`^{2} for the inertia. Click **Apply**.

2 Update your diagram. The body in the window changes to an ellipsoid, with *(x,y,z)* axes of length 2.02, 0.885, and 0.885 meters (m), respectively.

 Turn off the ellipsoid surface to see the CG CS triad alone. The *x*-, *y*-, and *z*-axes are red, green, and blue, respectively.



Test Body Ellipsoid: Initial Orientation, with Detail of CG CS Axes

Rotating the Body and Its CG CS Relative to World

In this example, you rotate the body, along with its CG CS axes, with respect to the World CS. The CG CS axes continue to have the same orientation with respect to the body shape. The rotation is a positive turn of 75 deg (degrees) about the axis (1,1,1) in World. In this example, you rotate with a quaternion.

Computing the Rotation as a Quaternion

The unit vector $\boldsymbol{n}$ in the (1,1,1) direction is $(1,1,1)/\sqrt{3}$. The rotation angle is θ = 75 deg = 1.3090 rad. At the MATLAB command line, define `th = pi*75/180` and compute the quaternion components:

```
q = [sin(th/2)/sqrt(3) sin(th/2)/sqrt(3) sin(th/2)/sqrt(3) ...
cos(th/2)]
```

```
q =
     0.3515     0.3515     0.3515     0.7934
```

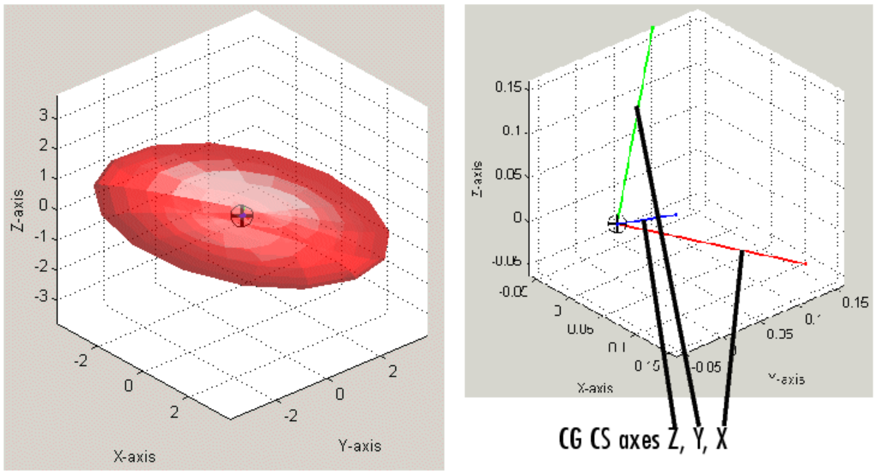
Rotating the Body and Its CG CS Axes with the Quaternion

To rotate the body and the CG CS coordinate axes together by this rotation,

1 In the Body dialog, click the **Orientation** tab in the Body coordinate systems area. Under the **Specified using convention** pull-down menu, select `Quaternion`.

2 Under **Orientation vector**, enter `q`. Leave the other defaults. The **Relative to coordinate system** field indicates that the rotation represented by `q` is oriented with respect to the World axes. Click **Apply**.

3 Update the diagram. The body and its CG CS rotate together relative to World:



Test Body Ellipsoid: Body and Its CG CS Axes Rotated Together

4 Finally rotate the body and its CG CS back to the original orientation by entering `[0 0 0 1]` under **Orientation vector** and clicking **Apply**. Update your diagram to refresh the visualization.

Rotating the Body Relative to Its Center of Gravity

You can also rotate the body *without* rotating its CG CS axes. To accomplish this requires leaving the Body CSs unchanged while rotating the body's inertia tensor relative to the CG CS. Here you use the same rotation as in the preceding example, but represent it as a rotation matrix.

Computing the Rotation as a Rotation Matrix

The unit vector $\boldsymbol{n}$ in the (1,1,1) direction is $(1,1,1)/\sqrt{3}$. The rotation angle is θ = 75 deg = 1.3090 rad. Compute the rotation matrix:

```
nDotJ = [0 -1/sqrt(3) 1/sqrt(3); 1/sqrt(3) 0 -1/sqrt(3); ...
-1/sqrt(3) 1/sqrt(3) 0]
nDotJ =
         0   -0.5774    0.5774
    0.5774         0   -0.5774
   -0.5774    0.5774         0

R = eye(3) + nDotJ*sin(th) + nDotJ^2*(1-cos(th))
R =
    0.5059   -0.3106    0.8047
    0.8047    0.5059   -0.3106
   -0.3106    0.8047    0.5059
```

Rotating the Body's Inertia Tensor

The components of the inertia tensor that you enter into the Body dialog are always defined relative to the CG CS axes. If you hold the CG CS axes fixed and rotate the body by a rotation matrix R, the inertia tensor transforms according to $\boldsymbol{I}_{\text{new}} = R^*\boldsymbol{I}_{\text{old}}{}^*R^{\text{T}}$. Compute this with MATLAB:

```
I = [18 0 0; 0 56 0; 0 0 56]
I =
    18     0     0
     0    56     0
     0     0    56

Irot = R*I*R'
Irot =
   46.2753  -15.4698    5.9711
```

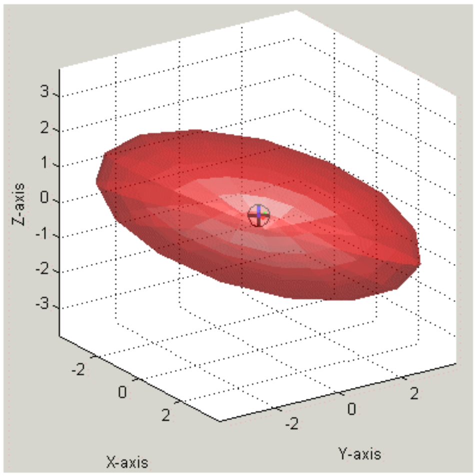
```
-15.4698   31.3911    9.4987
  5.9711    9.4987   52.3336
```

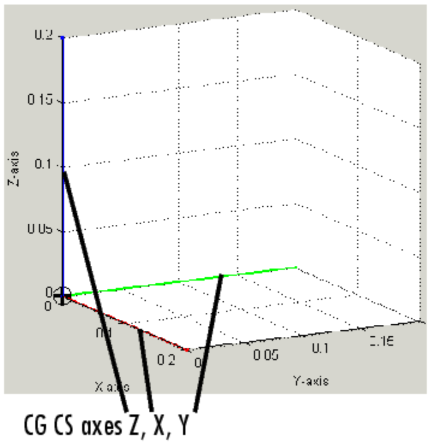
Rotating the Body with the Rotation Matrix

Symmetrize `Irot` by entering `Irot = (Irot + Irot')/2`. Then rotate the body alone relative to World and the CG CS.

1. In the Body dialog's **Mass properties** area, replace the existing inertia tensor with `Irot`. Click **Apply**.

2. Update the diagram. The body rotates again, as in the preceding example. But unlike that example, the CG CS axes do not change.





Test Body Ellipsoid: Body Rotated Relative to Its CG CS Axes

3. Finally rotate the body back to the original orientation by entering `I` in the **Inertia** field and clicking **Apply**. Update the diagram to refresh the visualization.

Creating and Rotating Body Coordinate Systems

In the preceding examples, you work with only one Body CS, the CG CS. In this example, you set up additional Body CSs and learn how to rotate them. It is common in mechanical applications to require extra Body CSs to locate sensors and actuators on bodies. Their axis orientations do not, in general, align with the orientation of the CG CS axes.

You visualize the body here using convex hulls, instead of ellipsoids, to articulate the Body CSs more clearly. Obtaining a full convex hull, with a surface enclosing a volume, requires at least three non-coplanar Body CSs, in addition to the CG CS.

This example also shows you how to obtain Euler angles and rotate with them.

Creating and Viewing the New Body CSs

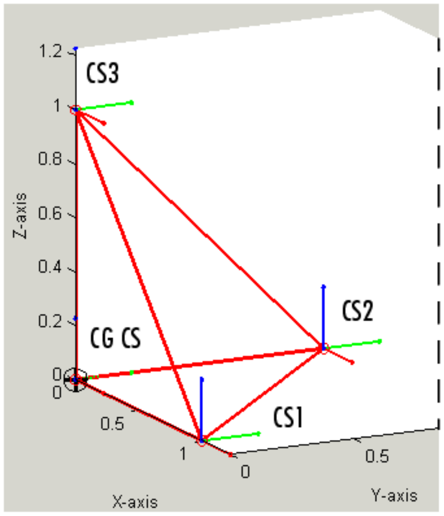
To change the visualization rendering to convex hulls,

1. Open the special **SimMechanics** menu in the visualization window. Select the **Machine Display** submenu.
2. Deselect **Ellipsoids** and select **Convex Hulls**.

To create and visualize the new Body CSs,

1. Go to the Body dialog's Body coordinate systems area. Add three new coordinate systems to the CS list. Name them CS1, CS2, and CS3.
2. For these three CSs, change the **Origin position vector** fields to `[1 0 0]`, `[0 1 0]`, and `[0 0 1]`, respectively. Click **Apply**.
3. Update the diagram. The SimMechanics window now renders the body as a tetrahedron composed of four adjoining triangular surfaces.
4. Make the new Body CSs easier to see by turning off the convex hull body surfaces and the axes grid. This step leaves only the Body CS triads and

the wire frame outline of the convex hull. All the Body CS triads are oriented the same way, parallel to the CG CS axes and the World axes.



Test Body Convex Hull with CG CS and Three Body Coordinate Systems

Computing the Rotation as a Set of Euler Angles

The preceding examples used two rotation representations, the quaternion and the rotation matrix, based on the axis-angle representation. In this example, you use the same rotation as before, but represented as a set of Euler angles.

The rotation axis sequence convention is Z-X-Z, with Φ, θ, Ψ as the first, second, and third angles, respectively, the same as presented in "Converting Rotation Representations" on page 3-14. To obtain the angles, you equate the rotation matrix form for that convention:

$$R_Z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

to the numerical form of R you computed in "Rotating the Body Relative to Its Center of Gravity" on page 3-22:

```
R =
    0.5059   -0.3106    0.8047
    0.8047    0.5059   -0.3106
   -0.3106    0.8047    0.5059
```

and invert the resulting equations. The solution for Θ and Ψ is

```
theta = acos(R(3,3))
theta =
    1.0404

psi = asin(R(3,1)/sin(theta))
psi =
   -0.3684
```

or about 60 and -21 deg, respectively.

Here is a method that yields a unique solution by using the structure of the Euler convention, $R = R_1(\Phi)*R_2(\theta)*R_3(\Psi)$. Multiply R on the right by $R_3(\Psi)^T*R_2(\theta)^T$, isolating $R_1(\Phi)$:

```
R3 = [cos(psi) -sin(psi) 0; sin(psi) cos(psi) 0; 0 0 1]
R3 =
    0.9329    0.3601         0
   -0.3601    0.9329         0
         0         0    1.0000

R2 = [1 0 0; 0 cos(theta) -sin(theta); 0 sin(theta) cos(theta)]
R2 =
    1.0000         0         0
         0    0.5059   -0.8626
         0    0.8626    0.5059

R1 = R*R3'*R2'
R1 =
    0.3601   -0.9329   -0.0000
    0.9329    0.3601   -0.0000
    0.0000    0.0000    1.0000

phi = acos(R1(1,1))
phi =
    1.2024
```

or about 69 deg.

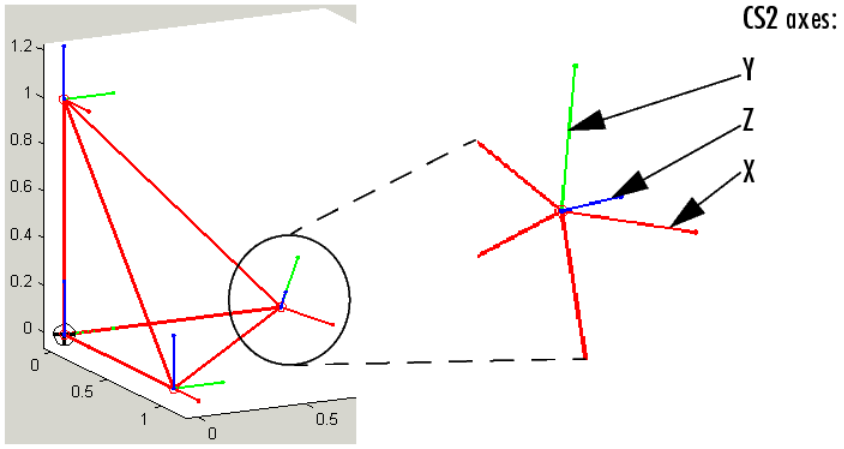
Rotating a Body CS Axis Triad

Here you rotate one of the Body CS axis triads, using the *Z-X-Z* rotation axis sequence convention and, for Euler angles, the Φ, θ, and Ψ you just found.

1. In the Body dialog's **Orientation** pane, locate the `CS2` entry.

   Under **Orientation vector**, enter `[phi theta psi]`. In the **Units** pull-down menu, select `rad`. In the **Specified using convention** pull-down menu, select `Euler Z-X-Z`. Click **Apply**.

2. Update your diagram. The CS2 axis triad, whose origin continues to be located at (0,1,0), now looks like this:



This rotated orientation of the CS2 axis triad is the same as that of the rotated CG CS in "Rotating the Body and Its CG CS Relative to World" on page 3-20. The two rotations are the same and produce the same result.

Modeling Mechanical Systems

SimMechanics gives you a complete set of block libraries for modeling machine parts and connecting them into a Simulink block diagram.

Refer to Chapter 3, “Representing Motion” to review body kinematics. If you need more information on rigid body mechanics, consult the physics and engineering literature, beginning with the Appendix B, “Bibliography”. Classic engineering mechanics texts include Goodman and Warner [3], [4] and Meriam [10]. The books of Goldstein [2] and José and Saletan [6] are more theoretically oriented.

Modeling Machines

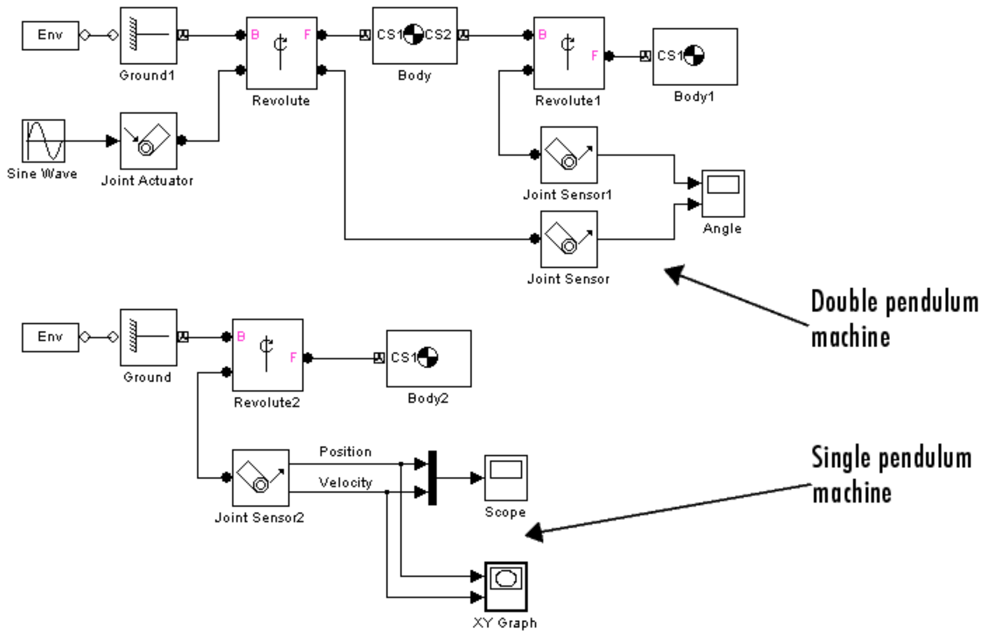
In SimMechanics the term *machine* has two meanings.

It refers to a physical system that includes at least one rigid body. SimMechanics provides a library of Simulink blocks that allow you to create Simulink models of machines.

It also refers to a topologically distinct and separate block diagram representing one physical machine. A model can have one or more machines.

About SimMechanics Models

A SimMechanics *model* consists of a *block diagram* composed of one or more *machines*, each of which is a set of connected blocks representing a single physical machine. For example, the following model contains two machines.



Comparison to Other Simulink Models

A SimMechanics model differs significantly from other Simulink models in how it represents a machine. An ordinary Simulink model represents the *mathematics* of a machine's motion, i.e., the algebraic and differential equations that predict the machine's future state from its present state. The mathematical model enables Simulink to simulate the machine. By contrast, a SimMechanics model represents the *physical structure* of a machine, the geometric and kinematic relationships of its component bodies. SimMechanics converts this structural representation to an internal, equivalent mathematical model. This saves you the time and effort of developing the mathematical model yourself.

Creating a SimMechanics Model

You create a SimMechanics model in much the same way you create any other Simulink model. First, you open a Simulink model window. Then you drag instances of SimMechanics and other Simulink blocks from the Simulink block libraries into the window and draw lines to interconnect the blocks (see "Connecting SimMechanics Blocks" on page 4-5).

The SimMechanics block library provides the following blocks specifically for modeling machines:

- Machine Environment blocks set the mechanical environment for a machine. Exactly one Ground block in each machine must be connected to a Machine Environment block. See Chapter 5, "Running Mechanical Models".
- Body blocks represent a machine's components and the machine's immobile surroundings (ground). See "Modeling Bodies and Grounds" on page 4-10.
- Joint blocks represent the degrees of freedom of one body relative to another body or to a point on ground. See "Modeling Joints" on page 4-20.
- Constraint and Driver blocks restrict motions of or impose motions on bodies relative to one another. See "Modeling Constraints and Drivers" on page 4-38.
- Actuator blocks specify forces, motions, variable masses and inertias, or initial conditions applied to bodies, joints, and drivers. See "Modeling Actuators" on page 4-45.

- Sensor blocks measure the forces on and motions of bodies, joints, and drivers. See "Modeling Sensors" on page 4-63.
- Force element blocks model interbody forces. See "Modeling Sensors" on page 4-63.

You can use blocks from other Simulink libraries in SimMechanics models. For example, you can connect the output of SimMechanics Sensor blocks to Scope blocks from the Simulink Sinks library to display the forces and motions of your model's bodies and joints. Similarly, you can connect blocks from the Simulink Sources library to SimMechanics Driver blocks to specify relative motions of your machine's bodies.

Connecting SimMechanics Blocks

In general, you connect SimMechanics blocks in the same way you connect other Simulink blocks: by drawing lines between them. Significant differences exist, however, between connecting standard Simulink blocks and connecting SimMechanics blocks. The following sections discuss these differences.

Connection Lines

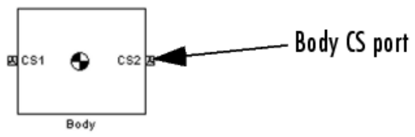
The lines that you draw between standard Simulink blocks, called signal lines, represent inputs to and outputs from the mathematical functions represented by the blocks. By contrast, the lines that you draw between SimMechanics blocks, called *connection lines*, represent physical connections and spatial relationships among the bodies represented by the blocks.

You can draw connection lines only between specialized connector ports available only on SimMechanics blocks (see next section) and you cannot branch existing connection lines. Connection lines appear as solid black when connected and as dashed red lines when either end is unconnected.

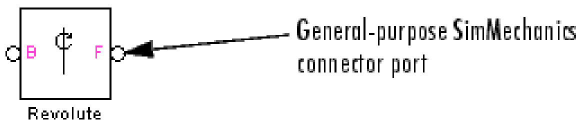
Connector Ports

Standard Simulink blocks have input and output ports. By contrast, most SimMechanics blocks contain only specialized *connector ports* that permit you to draw connection lines among SimMechanics blocks. SimMechanics connector ports are of two types: Body CS ports and general-purpose ports.

Body CS ports appear on Body and Ground blocks and define connection points on a body or ground. Each is associated with a local coordinate system whose origin specifies the location of the associated connection point on the body.

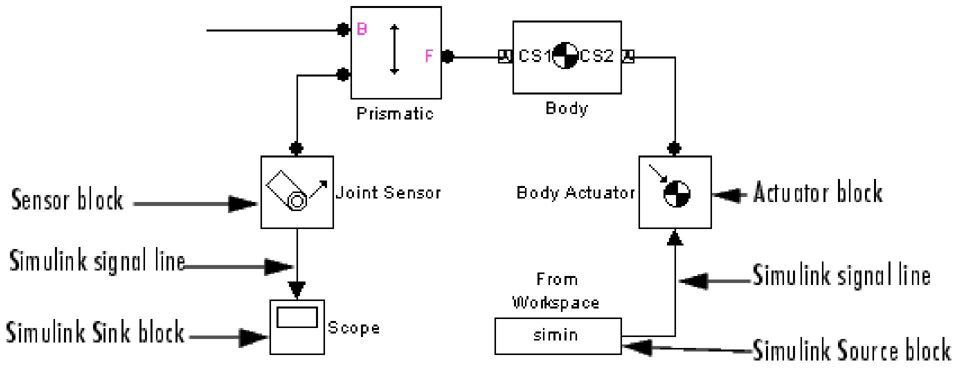


General-purpose connector ports appear on Joint, Constraint, Driver, Sensor, and Actuator blocks. They permit you to connect Joints to Bodies and connect Sensors and Actuators to Joints, Constraints, and Drivers. General-purpose connector ports appear as circles on the block icon. The circle is unfilled if the port is unconnected and filled if the port is connected.



Interfacing SimMechanics Blocks to Simulink Blocks

SimMechanics Actuator blocks (see “Modeling Actuators” on page 4-45) contain standard Simulink input ports. Thus, you can connect standard Simulink blocks to a SimMechanics model via Actuator blocks. Similarly, SimMechanics Sensor blocks contain output ports (see “Modeling Sensors” on page 4-63). Thus, you can connect a SimMechanics model to Simulink blocks via Sensor blocks.



Setting SimMechanics Block Properties at the Command Line

You cannot use the Simulink `set_param` and `get_param` commands to set or get SimMechanics block parameters. Instead, you must set block parameters via the block dialog boxes. You can open the dialogs by double-clicking the block, or by right-clicking the block and selecting **Open block**.

Creating SimMechanics Subsystems

Large, complex block diagram models are often hard to analyze. Enclosing functionally related groups of blocks in subsystems alleviates this difficulty and facilitates reuse of block groups in different models.

You can create subsystems containing SimMechanics blocks that you can connect to other SimMechanics blocks. You do this in two ways:

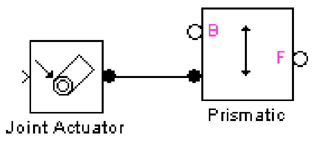
- Automatically
- Manually

The Simulink documentation explains more about creating subsystems.

Creating a Subsystem Automatically

To create a SimMechanics subsystem automatically,

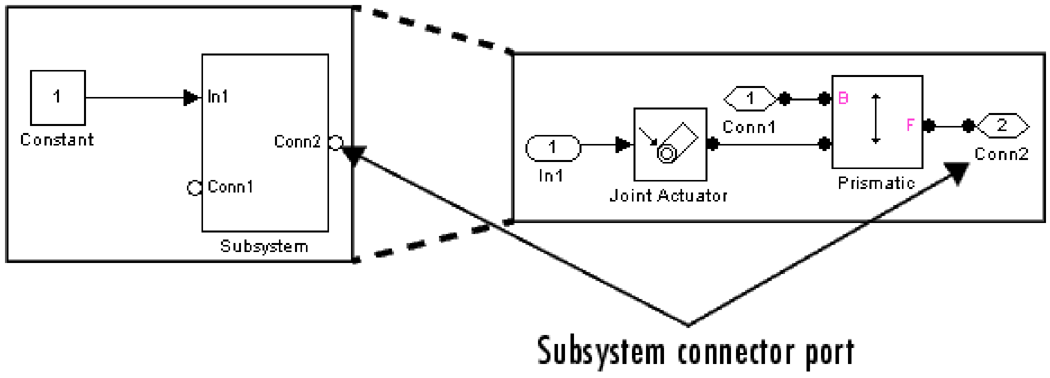
1 Create the subsystem block diagram in your model window, leaving unconnected ports for external connections.



2 Group-select the subsystem block diagram.

3 Select the **Create subsystem** command from the **Edit** menu of the Simulink model window.

The command replaces the block diagram with a Subsystem block containing the selected block diagram. The command also creates and connects SimMechanics Connection Port blocks for the ports that you left unconnected in the block diagram. The Connection Port blocks in turn create connector port icons on the subsystem icon, enabling you to connect external SimMechanics blocks to the new subsystem.



Creating a Subsystem Manually

Sometimes you need to make a subsystem configured differently from an automatically created one. To create a SimMechanics subsystem manually,

1. Drag a Subsystem block into your model window.

2. Open the Subsystem block.

3. Create the subsystem block diagram in the subsystem window.

4. Drag a Connection Port block from the SimMechanics Utilities library into the subsystem window for each port that you want to be available externally.

5. Connect the external connector ports to the Connection Port blocks.

Creating Custom SimMechanics Blocks with Masks

You can create your own SimMechanics blocks from subsystems, for example, a spring-loaded Joint block or a sphere Body block. To do this, create a block diagram that implements the functionality of your custom block, enclose the diagram as a subsystem, and add a mask (i.e., user interface) to the subsystem. To facilitate sharing your custom blocks across models or with other users, create a Simulink block library and add these masked subsystem blocks to the library. The Simulink documentation explains how to create custom blocks with masks.

Modeling Bodies and Grounds

The basic components of any mechanism are its constituent rigid bodies. In SimMechanics, the term *body* refers to any point or spatially extended object that has mass. SimMechanics bodies, unlike physical bodies, do not have degrees of freedom. The SimMechanics Bodies library contains two blocks for representing bodies in a Simulink model:

- Ground

  Models a point on an ideal body of infinite mass and extent that serves as a fixed reference point for machines (see “Modeling Grounds” on page 4-10).

- Body

  Models rigid bodies of finite mass and extent, including their attached body coordinate systems (see “Modeling Rigid Bodies” on page 4-12 and “Working with Body Coordinate Systems” on page 4-15).

Chapter 3, “Representing Motion” explains, with detailed examples, more about configuring bodies and their coordinate systems in space.

Machine Environment Required for Each Machine

One Ground block in each machine of your model plays a second role, connection to that machine’s Machine Environment block, which sets its mechanical environment. See Chapter 5, “Running Mechanical Models”.

Modeling Grounds

In SimMechanics, *ground* refers to a body of infinite mass and size that acts both as a reference frame at rest for a machine as a whole and as a fixed base for attaching machine components, e.g., the factory floor on which a robot stands. SimMechanics Ground blocks enable you to represent points on ground in your machine. This in turn enables you to specify the degrees of freedom that your system has relative to its surroundings. You do this by connecting Joint blocks representing the degrees of freedom between the Body blocks representing parts of your machine and the Ground blocks representing ground points.

Each Ground block has a single connector port to which you can connect a Joint block that can in turn be connected to a single Body block. Each Ground block therefore allows you to represent the degrees of freedom between a single part of your machine and its surroundings. If you want to specify the motion of other parts of your machine relative to the surroundings, you must create additional Ground blocks.

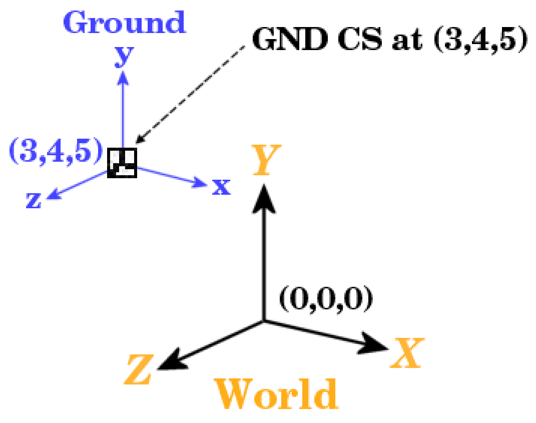
Note Each machine in a SimMechanics model must contain at least one Ground block connected to a Body block via a Joint block. Each submachine connected by a Shared Environment block must have at least one Ground.

Exactly one Ground block in each machine in your model must be connected to a Machine Environment block.

The World Coordinate System

SimMechanics uses an internal master coordinate system and reference frame called World. All grounds are at rest in World. The connector port of each Ground block defines a grounded coordinate system called GND. The GND coordinate system's axes are parallel to World. By default the origin of the grounded coordinate system coincides with the origin of the World coordinate system.

The **Location** field of a Ground block's dialog box allows you to move the origin of GND to some other point in the World coordinate system, as in the example "Building a Simple Pendulum" on page 2-11.



The GND coordinate system allows you to specify the positions and motions of parts of your machine relative to fixed points in the machine's surroundings.

Modeling Rigid Bodies

The SimMechanics Body block enables you to model rigid bodies of finite mass and extent. A body is rigid if its internal parts cannot move relative to one another.

About Body Blocks

A Body block allows you to specify the following attributes of a rigid body.

Mass Properties. These include the body's mass, which determines its response to translational forces, and its inertia tensor, which determines its response to rotational torques.

Body Coordinate Systems. By default a Body block defines three local coordinate systems, one associated with a body's center of gravity, labeled CG, and two others, labeled CS1 and CS2, respectively, associated with two other points on the body that you can specify. You can create additional body coordinate systems or delete them as necessary.

A Body block's dialog box allows you to specify a Body CS's origin (see "Setting a Body CS's Position" on page 4-15) and orientation (see "Setting a Body CS's Orientation" on page 4-17). The origin and orientation of a body's CG CS specify the body's starting location and orientation. The origins of the other body coordinate systems specify the initial locations of other points on the body.

SimMechanics allows flexibility in specifying the origins and orientations of a body's coordinate systems. You can specify the origin and orientation of a body CS relative to

- The World CS
- Any other CS on the same body
- The *Adjoining CS*, the CS on the neighboring body or ground directly connected by a Joint, Constraint, or Driver to the selected Body CS you are configuring

This simplifies creation and maintenance of models. The only limitation is that you must specify the origin and location of at least one of a machine's body coordinate systems relative to the World CS.

Home Configuration. Once you enter all the needed positions and orientations into the Bodies of your model, your machine is in its *home configuration*. The body velocities are zero, and any disassembled joints remain disassembled.

Connector Ports. Any Body CS can display a Body CS Port. A Body CS Port allows you to attach Joints, Actuators, and Sensors to a Body. By default, a Body's CS1 and CS2 coordinate systems each display a Body CS port. You can display a port for any other Body coordinate system as well, including a Body's CG CS.

Creating a Body Block

To create a Body block,

1 Drag a Body block icon from the SimMechanics Bodies Library and drop it into your model window.

2 Open the Body block's dialog box.

3 Enter the mass of the body you are modeling in the **Mass** field.

4 Select the units of mass from the adjacent units list.

5 Enter a 3-by-3 matrix representing the body's inertia tensor relative to its center of gravity coordinate system (CG CS) origin and axes in the **Inertia** field (see "Determining Inertia Tensors for Common Shapes" on page 4-14).

6 Enter the initial positions of the body's CG and coordinate systems in the **Position** pane.

7 Enter the initial orientation of the body's CG and coordinate systems in the **Orientation** pane.

8 Click **OK** or **Apply**.

Determining Inertia Tensors for Common Shapes

The following table enables you to determine the inertia tensors for some common shapes. For each shape of mass `m`, the table lists the shape's principal moments of inertia, I_1, I_2, and I_3, along the *x*-, *y*-, and *z*-axes of the shape's CG coordinate system.

Shape	I_1	I_2	I_3
Thin rod of length L aligned along z	$mL^2/12$	$mL^2/12$	0
Sphere of radius R	$2mR^2/5$	$2mR^2/5$	$2mR^2/5$
Cylinder of radius R and height h aligned along z	$(m/4)(R^2 + h^2/3)$	$(m/4)(R^2 + h^2/3)$	$mR^2/2$

Shape	I_1	I_2	I_3
Rectangular parallelopiped of sides a, b, and c aligned along x, y, z, respectively	$(m/12)(b^2 + c^2)$	$(m/12)(a^2 + c^2)$	$(m/12)(a^2 + b^2)$
Cone of base radius R and height h along z	$(m/4)(3R^2/5 + h^2)$	$(m/4)(3R^2/5 + h^2)$	$3mR^2/10$
Ellipsoid of semiaxes a, b, and c aligned along x, y, z, respectively	$(m/5)(b^2 + c^2)$	$(m/5)(a^2 + c^2)$	$(m/5)(a^2 + b^2)$

The corresponding inertia tensor for the shape is the following 3-by-3 matrix:

$$\begin{pmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{pmatrix}$$

Working with Body Coordinate Systems

Every body in SimMechanics has body coordinate systems (CSs) attached to it. The location of a body CS is the origin of that CS. The CS's rectangular *x*-*y*-*z* coordinate axes are rotated at some orientation. You set up body CS origins and orientations before running your model. But once the bodies start to move, the origins and orientations of a body's CSs remain fixed in the body. The elements of a body's inertia tensor also remain fixed in the body. Consult Chapter 3, "Representing Motion" for more about orienting bodies and body CSs.

The sections "Managing Body Coordinate Systems" on page 4-18 and "Creating Body CS Ports" on page 4-19 explain how to create custom body coordinate systems and Body CS ports or delete existing ports.

Setting a Body CS's Position

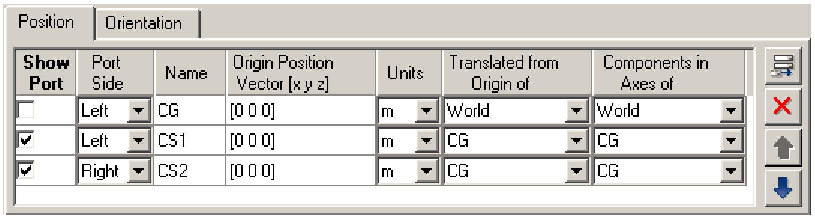
The **Position** pane of a Body block's dialog box allows you to specify the position of any of a body's local coordinate systems.

The **Translated from Origin of** and **Components in Axes of** lists in the pane together specify which other of your machine's coordinate systems you use as reference points and orientations to set up the coordinate systems of the body you are configuring.

To specify the position of a Body CS,

1 Open the Body block's dialog box.

   The dialog box's **Position** pane lists the body's local coordinate systems in a table.

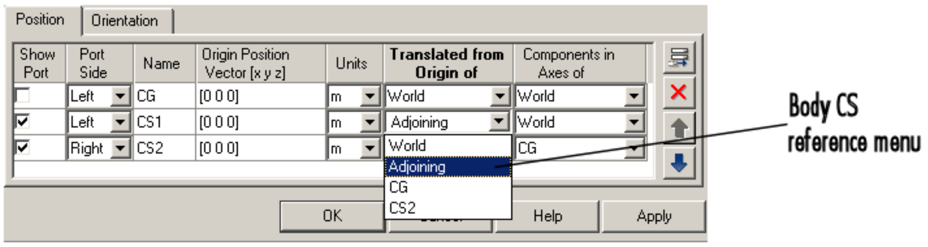
Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	CG	CG
✓	Right	CS2	[0 0 0]	m	CG	CG

   Each row specifies the position of the coordinate system specified in the **Name** column.

2 Select the units in which you want to specify the origin of the Body CS from the CS's **Units** list.

3 Specify the reference coordinate systems for the Body CS, i.e., the coordinate systems relative to which you want to measure the Body CS origin and the orientation of the Body CS's coordinate axes. The choices are World, the adjoining CS, and other Body CSs on the same Body.

   You must directly or indirectly define all Body CSs by reference to a Ground or to World. Indirect reference means that you specify a Body CS relative to another CS and so on, in a chain of references that ultimately ends in a Ground or World.



Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	Adjoining	World
✓	Right	CS2	[0 0 0]	m		CG

You do this by selecting the origin and orientation of the specification CS from the Body CS's **Translated from Origin of** and **Components in Axes of** lists, respectively. For example, suppose that you want to specify the position of CS2 relative to another coordinate system, whose origin is at the origin of CS1 but whose axes run parallel to those of the CG CS. Then you would select CS1 from the **Translated from Origin of** list of CS2 and CG from the **Components in Axes of** list of CS2.

4 Enter a vector specifying the location of the Body CS in the **Origin Position Vector [x y z]** field of the CS.

The components of the vector must be in the units that you selected and relative to the coordinate system that you selected. For example, suppose that you had selected `m` as the unit for specifying CS2's origin and CS1 and WORLD as the CSs specifying the origin and orientation for CS2. Now suppose that you want to specify the location of CS2 as one meter to the right of CS1 along the World *x*-axis. Then you would enter `[1 0 0]` as CS2's position vector.

5 Click **Apply** to accept the position setting or **OK** to accept the setting and dismiss the dialog box.

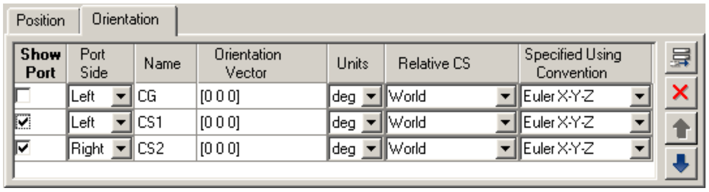
Setting a Body CS's Orientation

The **Orientation** pane of a Body block's dialog box allows you to specify the orientation of any of a body's local coordinate systems.

To specify the orientation of a Body CS,

1 Open the Body block's dialog box.

2 Select the dialog box's **Orientation** pane.



Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☐	Left	CG	[0 0 0]	deg	World	Euler X-Y-Z
☑	Left	CS1	[0 0 0]	deg	World	Euler X-Y-Z
☑	Right	CS2	[0 0 0]	deg	World	Euler X-Y-Z

3 Select the units (degrees or radians) in which you want to specify the orientation of the CS from the CS's **Units** list.

4 Select the coordinate system relative to which you want to specify the orientation of the Body CS from the Body CS's **Relative CS** list. The choices are World, the adjoining CS, and other Body CSs on the same Body.

5 Select the convention you want to use to specify the orientation of the Body CS from the CS's **Specified Using Convention** list.

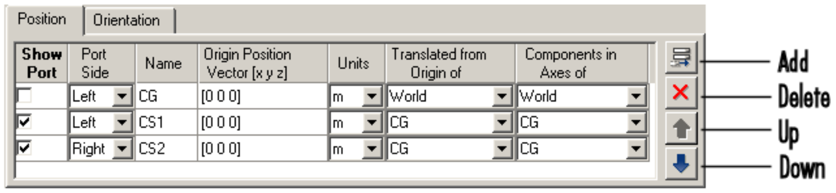
6 Enter a vector that specifies the orientation of the Body CS relative to the CS you choose for that purpose, according to the selected specification convention.

7 Click **Apply** to accept the orientation setting or **OK** to accept the setting and dismiss the dialog box.

Managing Body Coordinate Systems

You will often need to modify the default body coordinate systems of a Body block. You might want to connect a Body to more than two Joints, in which case you need not only more Body CSs, but their Body CS ports as well. Connecting Actuators and Sensors to Bodies requires a Body CS and Body CS port for each connection.

The Body coordinate systems panel of a Body block's dialog box contains a row of buttons that allow you to add, delete, and reorder a Body's local coordinate systems.



Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	CG	CG
✓	Right	CS2	[0 0 0]	m	CG	CG

To use these buttons, select a Body CS in the CS table and select

- **Delete** to remove the selected CS from the table
- **Up** to move the CS's entry one row up in the CS table
- **Down** to move the CS's entry one row down in the CS table

Select **Add** to add a new CS.

Creating Body CS Ports

To add or delete a port from a Body block's icon, open the block's dialog box and select or clear the CS's **Show Port** check box in the dialog box's Body CS table. Click **OK** or **Apply** to confirm the change.

Modeling Joints

In SimMechanics, a joint represents the degrees of freedom (DoF) that one body (the follower) has relative to another body (the base). The base body can be a finite rigid body or a ground. Unlike a physical joint, a SimMechanics joint has no mass, although some joints have spatial extension (see "Modeling with Massless Connectors" on page 4-29).

A SimMechanics joint does not necessarily imply a physical connection between two bodies. For example, a SimMechanics Six-DoF joint allows the follower, e.g., an airplane, unconstrained movement relative to the base, e.g., ground, and does not require that the follower ever come into contact with the base.

SimMechanics joints only add degrees of freedom to a machine, because the Body blocks carry no degrees of freedom. Contrast this with physical joints, which both add DoFs (with axes of motion) and remove DoFs (by connecting bodies). See "Counting Degrees of Freedom" on page 4-77.

SimMechanics provides an extensive Joints Library with blocks for modeling various types of joints. This section explains how to use these blocks:

- "About Joints" on page 4-20
- "Creating a Joint" on page 4-27
- "Modeling with Massless Connectors" on page 4-29
- "Modeling with Disassembled Joints" on page 4-33
- "Cutting Closed Loops" on page 4-36

About Joints

Modeling with Joint blocks requires an understanding of the following key concepts:

- Joint primitives
- Joint types
- Joint axes
- Joint directionality

- Assembly restrictions

Joint Primitives

Each Joint block conceptually represents one or more joint primitives that together specify the degrees of freedom that a follower body has relative to the base body. The following table summarizes the joint primitives found singly or multiply in Joint blocks.

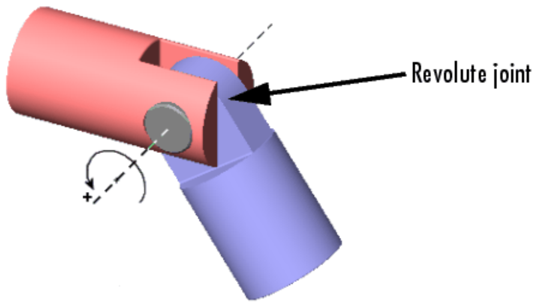
Primitive Type	Symbol	Degrees of Freedom
Prismatic	`P`	One degree of translational freedom along a prismatic axis
Revolute	`R`	One degree of rotational freedom about a revolute axis
Spherical	`S`	Three degrees of rotational freedom about a pivot point
Weld	`W`	Zero degrees of freedom

Joint Types

The blocks in the SimMechanics Joint Library fall into the following categories:

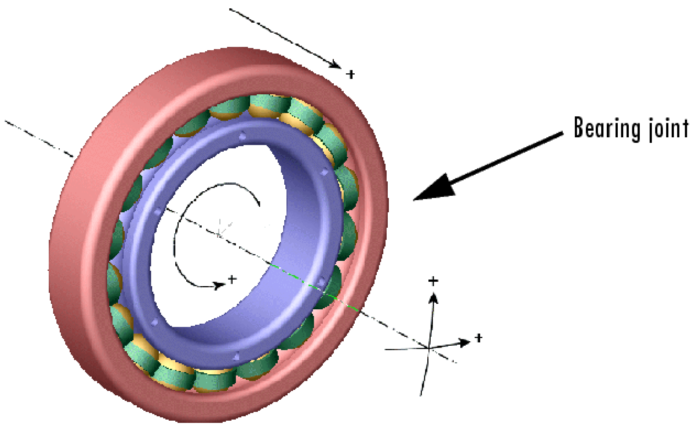
- Primitive joints

  Each of these blocks contains a single joint primitive. For example, the Revolute block contains a revolute joint primitive.



- Composite joints

  These blocks contain combinations of joint primitives, enabling you to specify multiple rotational and translational degrees of freedom of one body relative to another. Some model idealized real joints, for example, the Gimbal and Bearing joints.

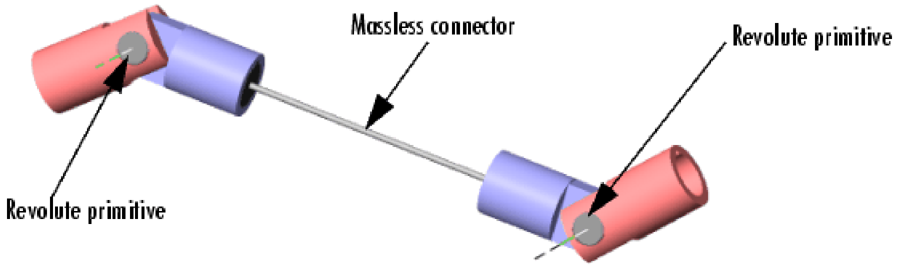


Others specify abstract combinations of degrees of freedom. For example, the Six-DoF block specifies unlimited motion of the follower relative to the base.

The Custom Joint allows you to create joints with any desired combination of rotational and translational degrees of freedom, in any order. The prefabricated composite Joints of the Joints library have the type and order of their primitives fixed. See "Axis Order" on page 4-24 under "Joint Axes" on page 4-24.

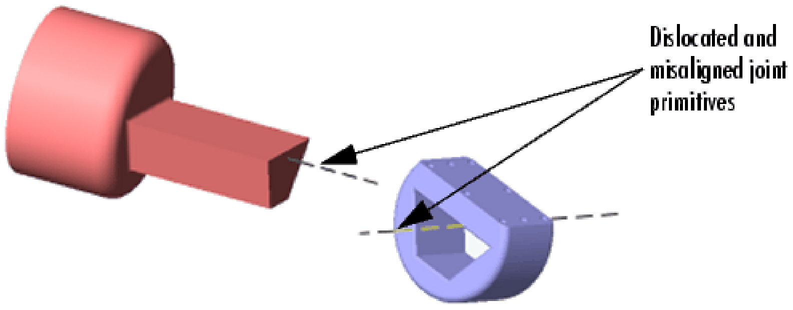
- Massless connectors

  These blocks represent extended joints with spatially separated joint primitive axes, for example, a Revolute-Revolute Massless Connector.



- Disassembled joints

  These blocks represent joints that SimMechanics assembles at simulation time, for example, a Disassembled Prismatic.



See "Assembly Restrictions" on page 4-26 and "Modeling with Disassembled Joints" on page 4-33.

Joint Axes

Joint blocks define one or more axes of translation or rotation along which or around which a follower block can move in relation to the base block. The axes of a Joint block are the axes defined by its component primitives:

- A prismatic primitive defines an axis of translation.
- A revolute primitive defines an axis of revolution.
- A spherical primitive defines a pivot point for axis-angle rotation.

For example, a Planar Joint block combines two prismatic axes and hence defines two axes of translation.

Axis Direction. By default the axes of prismatic and revolute primitives point in the same direction as the *z*-axis of the World coordinate system. A Joint block's dialog box allows you to point its prismatic and revolute axes in any other direction (see "Directing Joint Axes" on page 4-28).

Axis Order. SimMechanics executes the motion of composite joints one joint primitive at a time. A joint that defines more than one axis of motion also defines the order in which the follower body moves along each axis or about a pivot. The order in which the axes and/or pivot appear in the Joint block's dialog box is the order in which the follower body moves.

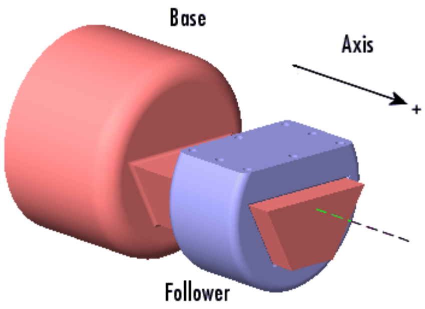
Different primitive execution orders are physically equivalent, unless the joint includes one spherical or three revolute primitives. Pure translations and pure two-dimensional rotations are independent of primitive ordering.

Joint Directionality

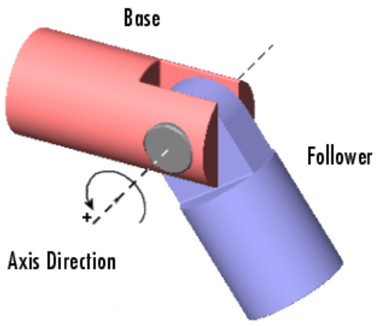
Directionality is a property of a joint that determines the dependence of the joint on the sign of forces or torques applied to it. A joint's directionality also determines the sign of signals output by sensors attached to the joint. SimMechanics assigns a directionality to every joint in your model. You must be able to determine the directionality of a joint in order to actuate it correctly and to interpret the output of sensors attached to it.

When assigning directionality to a joint, SimMechanics regards the joint's follower as moving relative to the joint's base. SimMechanics then assigns a directionality to the joint, taking into account the type of joint and the direction of the joint's axis, as follows.

Directionality of a Prismatic Joint. If the joint is prismatic, a positive force applied to the joint moves the follower in the positive direction along the axis of translation. A sensor attached to the joint outputs a positive signal if the follower moves in a positive direction along the joint's axis of translation relative to the base.



Directionality of a Revolute Joint. If the joint is revolute, a positive torque applied to the joint rotates the follower by a positive angle around the joint's axis of rotation, as determined by the right-hand rule. A sensor attached to the revolute joint outputs a positive signal if the follower rotates by a positive angle around the joint's axis of revolution, as determined by the right-hand rule.



Directionality of a Spherical Joint. Spherical joint directionality means the positive sense of rotation of the three rotational DoFs. Pick a rotation axis, rotating using the right-hand rule from the base Body CS axes. Then rotate the follower Body about that axis in the right-handed sense.

Directionality of Composite Joints. SimMechanics assigns a directionality separately to each joint primitive, based on the primitive's type and the direction of its axis of translation or rotation. In each case, SimMechanics regards the follower body of the composite joint as moving relative to the base body along or around the joint primitive's axis.

The order of primitives in the composite Joint's dialog determines the spatial construction of the joint. The first listed primitive is attached to the base, the second to the first, and so on, down to the follower, which is attached to the last primitive. Moving the first listed primitive moves the subsequent primitives in the list, as well as the follower, relative to the base. Moving any primitive moves the primitives below it in the list (but not those above it), as well as the follower. Moving the last listed primitive moves only the follower.

Changing the Directionality of a Joint. You can change the directionality of a joint by rewiring the Joint block to reverse the roles of the base and follower bodies or by reversing the sign (direction) of the joint axis.

Assembly Restrictions

Many joints impose one or more restrictions, called *assembly restrictions*, on the positions of the bodies that they join. The conjoined bodies must satisfy these restrictions at the beginning of simulation and thereafter within assembly tolerances that you can specify (see "Setting Assembly Tolerances" on page 5-5). For example, the attachment points of revolute and spherical joints must coincide within assembly tolerances; the attachment points of a Prismatic joint must be collinear with the prismatic axis within assembly tolerances; the attachment points of a Planar joint must be coplanar, etc. Composite joints, e.g., the Six-DoF joint, impose assembly restrictions equal to the most restrictive of its joint primitives. See the block reference for each Joint for information on the assembly restrictions, if any, that it imposes. Positioning bodies so that they satisfy a joint's assembly restrictions is called *assembling* the joint.

All joints except joints in the SimMechanics Disassembled Joints sublibrary require manual assembly. Manual assembly entails your setting the initial positions of conjoined bodies to valid locations (see “Assembling Joints” on page 4-29). SimMechanics assembles disassembled joints during the model initialization phase of simulation. It assumes that you have already assembled all other joints before the start of simulation. Hence joints that require manual assembly are called *assembled* joints. During model initialization and at each time step, SimMechanics also checks to ensure that your model’s bodies satisfy all assembly restrictions. If any of your model bodies fails to satisfy assembly restrictions, Simulink halts the simulation and displays an error message.

Creating a Joint

A joint must connect exactly two bodies. To create a joint between two bodies:

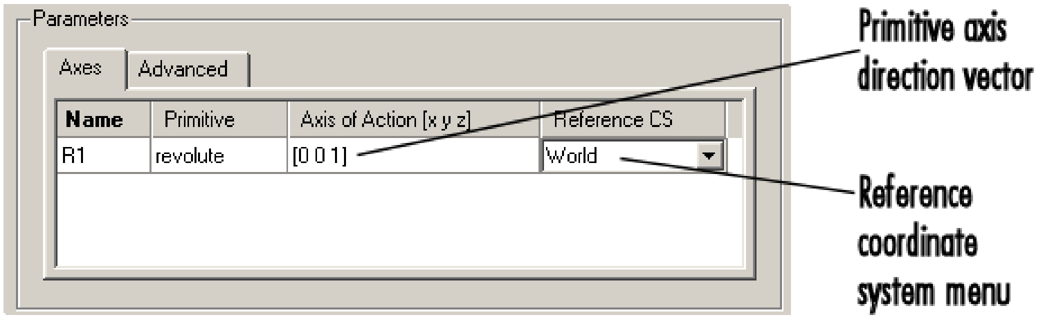
1. Select the Joint from the SimMechanics Joints library that best represents the degrees of freedom of the follower body relative to the base body.
2. Connect the base connector port of the Joint block (labeled B) to the point on the base block that serves as the point of reference for specifying the degrees of freedom of the follower block.
3. Connect the follower connector port of the Joint block (labeled F) to the point on the follower block that serves as the point of reference for specifying the degrees of freedom of the base block.
4. Specify the directions of the joint’s axes (see “Directing Joint Axes” on page 4-28).
5. If you plan to attach Sensors or Actuators to the Joint, create an additional port for each Sensor and Actuator (see “Creating Actuator and Sensor Ports on a Joint” on page 4-28).
6. If the joint is an assembled joint, assemble the bodies joined by the joint (see “Assembling Joints” on page 4-29).

Directing Joint Axes

By default the prismatic and revolute axes of a joint point in the same direction as the z-axis of the World coordinate system. To change the direction of the axis of a joint primitive:

1 Open the joint's dialog box and select a reference coordinate system for specifying the axis direction from the coordinate system list associated with the axis primitive.

   The options are the World coordinate system or the local coordinate systems of the base or follower attachment point. Choose the coordinate system that is most convenient.



Parameters

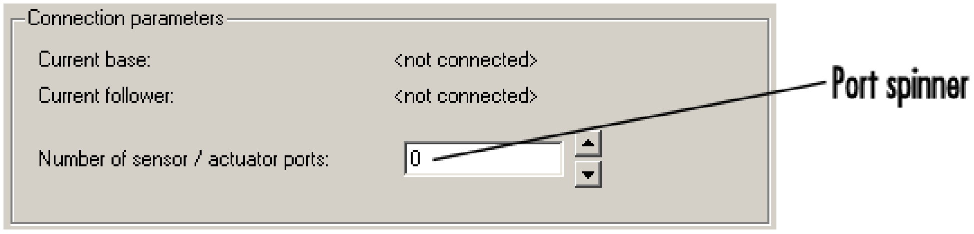
Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Axes
Advanced
Primitive axis direction vector
Reference coordinate system menu

2 Enter in the primitive's axis direction field a vector that points in the desired direction of the axis in the selected coordinate system.

Creating Actuator and Sensor Ports on a Joint

To create additional connector ports on a Joint for Actuators and Sensors, open the Joint's dialog box and set the **Number of sensor/actuator ports** to the number of Actuators and Sensors you plan to attach to the Joint.



Connection parameters

Current base: <not connected>

Current follower: <not connected>

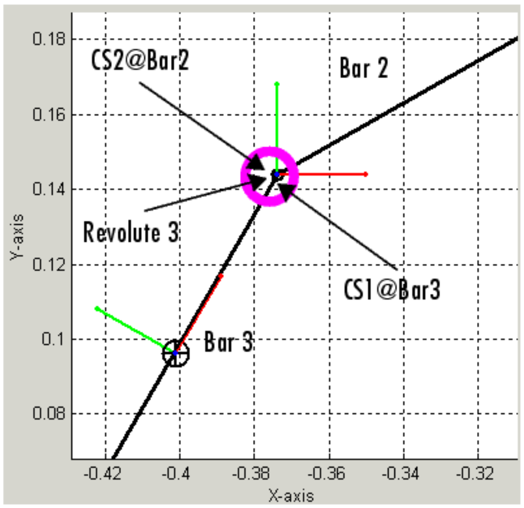
Number of sensor / actuator ports: 0

Apply the setting by clicking **OK** or **Apply**.

Assembling Joints

You must manually assemble all assembled joints in your model. Assembling a joint requires setting the initial positions of its base and follower attachment points such that they satisfy the assembly restrictions imposed by the joint (see "Assembly Restrictions" on page 4-26). Consider, for example, the model discussed in "Four Bar Mechanism" on page 2-36.

This model comprises three bars connected by revolute joints to each other and to two ground points. The model collocates the CS origins of the Body CS ports connected to each Joint, thereby satisfying the assembly restrictions imposed by the revolute joints.

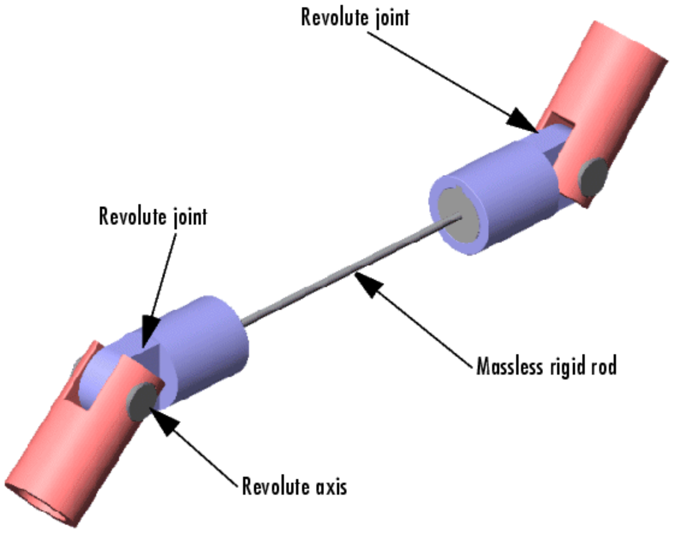


Assembled Revolute Joint in the Four Bar Mechanism

Modeling with Massless Connectors

Massless connectors simplify the modeling of machines that use a relatively light body to connect two relatively massive bodies. For example, you could use a Body block to model such a connector. But the resulting equations of motion might be ill-conditioned, because that connecting body's mass is small, and the simulation can be slow or error prone. A massless connector also avoids global inconsistencies that can arise if you use a Constraint block to model the connector.

A massless connector consists of a pair of joints located a fixed distance apart. Think of a massless connector as a massless rod with a joint primitive affixed at each end.



The initial orientation and length of the massless connector are defined by a vector drawn from the base attachment point to the follower attachment point. During simulation, the orientation of the massless connector can change but not its length. In other words, the massless connector preserves the initial separation of the base and follower bodies during machine motion.

Note You cannot actuate or sense a massless connector.

The SimMechanics Joints/Massless Connectors sublibrary contains three Massless Connectors:

- One with two revolute primitives (Revolute-Revolute)

- One with a revolute primitive and a spherical primitive (Revolute-Spherical)
- One with two spherical primitives (Spherical-Spherical)

You can set the direction of the axes of the revolute primitives.

Creating a Massless Connector

To create a massless connector between two bodies:

1. Drag an instance of a Massless Connector block from the Massless Connectors sublibrary into your model and connect it to the base and follower blocks.

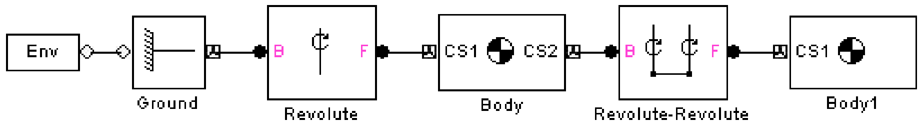
   If necessary, point the axes of the connector's revolute joints in the direction required by the dynamics of the machine you are modeling.

2. Assemble the connector by setting the initial positions of the base and follower body attachment points to the initial positions required by your machine's structure.

During simulation, the massless connector maintains the initial separation between the bodies though not necessarily the initial relative orientation.

Massless Connector Example: Triple Pendulum

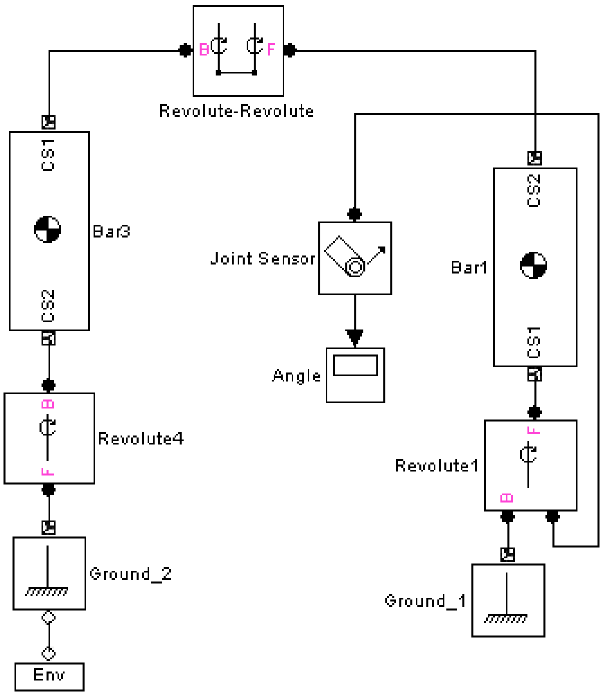
Consider a triple pendulum comprising massive upper and lower bodies and a middle body of negligible mass. The following model uses a Revolute-Revolute massless connector to model such a pendulum.



In this model, the joint axes of the Revolute-Revolute connector have their default orientation along the World z-axis. As a result, the lower arm (Body1) rotates parallel to the World's x-y plane.

Massless Connector Example: Four Bar Mechanism

The following model replaces one of the bars (Bar2) in the `mech_four_bar` model from the Demos library with a Revolute-Revolute massless connector.

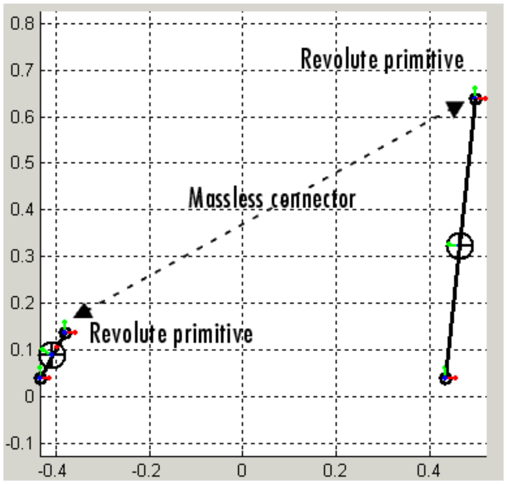


This model changes the Body CS origins of Bar3 to the following values.

Name	Origin position vector	Translated from origin of
`CG`	`[-0.027 0.048 0]`	`CS1`
`CS1`	`[0.054 0.096 0]`	`CS2`
`CS2`	`[0 0 0]`	`ADJOINING` (Ground_2)

This creates a separation between Bar3 and Bar1 equal to the length of Bar2 in the original model.

Try simulating both the original and the modified model. Notice that the massless connector version moves differently, because you eliminated the mass of Bar2 from the model. Notice also that the massless bar does not appear in the animation of the massless connector version of the model.



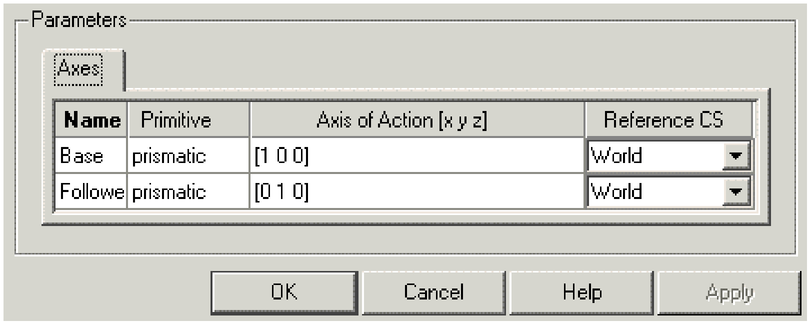
Modeling with Disassembled Joints

The SimMechanics Joints/Disassembled Joints sublibrary contains a set of joints that SimMechanics automatically assembles at the start of simulation; that is, SimMechanics positions the joints such that they satisfy the assembly restrictions imposed by the type of joint, e.g., prismatic or revolute. Using these joints eliminates the need for you to assemble the joints yourself.

Disassembled joints differ from assembled joints in significant ways. An assembled joint primitive has only one axis of translation or revolution or one spherical pivot point. A disassembled prismatic or revolute primitive has two axes of translation or rotation, one for the base and one for the follower body. A disassembled spherical primitive similarly has two pivot points.

Note Disassembled joints can appear only in closed loops. Each closed loop can contain only one disassembled joint.

The dialog box for a disassembled joint allows you to specify the direction of each axis.



Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

During model assembly, SimMechanics determines a common axis of revolution or translation that satisfies model assembly restrictions, and aligns the base and follower axes along the common axis.

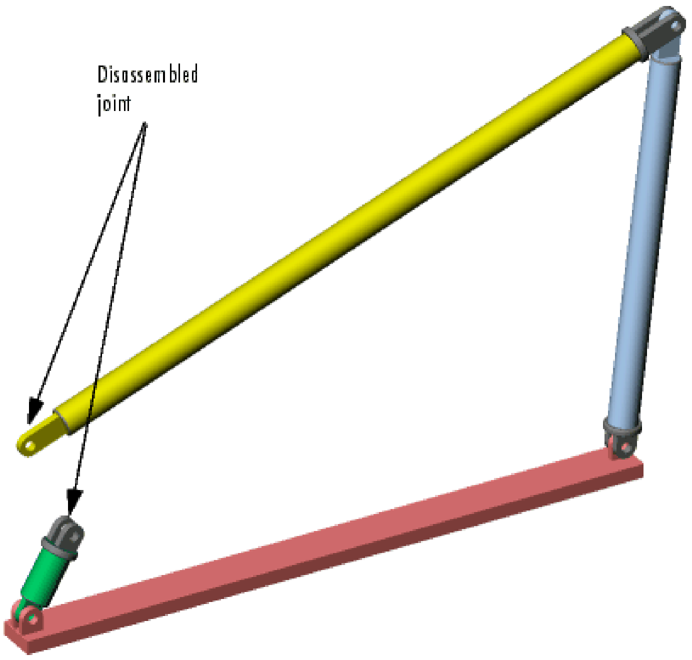
Controlling Automatic Assembly and the Assembled Configuration

If your machine contains Joint Initial Condition Actuator (JICA) blocks, the machine is moved from its home to its *initial configuration* by applying the initial condition information to the machine's joints first. Then any disassembled joints are assembled, leading to the *assembled configuration*.

During model assembly, SimMechanics might move bodies connected by assembled joints from their initial positions in order to assemble the disassembled joints. The SimMechanics solution to the assembly problem cannot be predicted beforehand, except in simple cases. To prevent SimMechanics from moving bodies during model assembly, use JICA blocks to specify the initial positions of bodies whose positions you want to remain fixed during the assembly process. This forces SimMechanics to find assembly solutions that satisfy the initial conditions specified by the JICA blocks.

Disassembled Joint Example: Four Bar Mechanism

This example creates and runs a model of a disassembled four bar machine.



Refer to the tutorial, "Four Bar Mechanism" on page 2-36, and the `mech_four_bar` demo:

1 Disconnect the Joint Sensor1 block from the Revolute3 block.

2 Replace Revolute3 with a Disassembled Revolute block from the Joints/Disassembled Joints sublibrary.

3 Open the Disassembled Revolute dialog box and, under **Axis of Action** for both `Base` and `Follower` axes, enter `[0 0 1]`. Close the dialog.

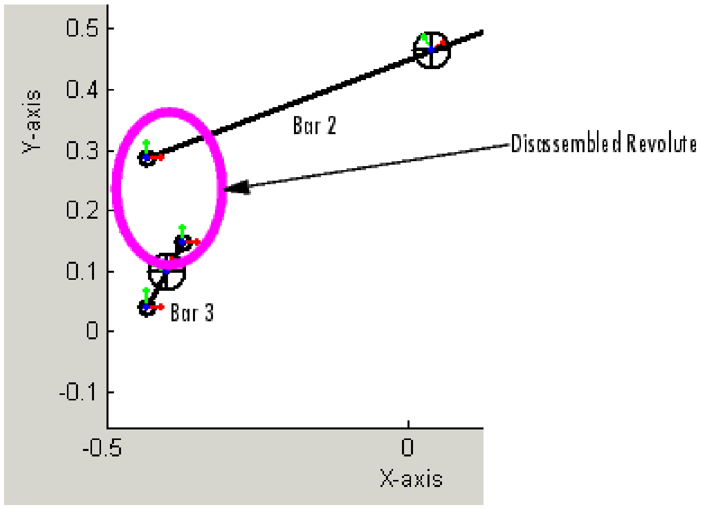
4 Open the Bar2 dialog box and dislocate the joint by displacing Bar2's CS2 origin from Bar 3's CS1 origin.

Do this by entering a nonzero vector under **Origin Position Vector [x y z]** for CS2, then changing the **Translated from Origin of** pull-down entry to `ADJOINING`. CS1 on Bar3 is the Adjoining CS of CS2 of Bar2. Close the dialog.

5 To avoid circular CS referencing, you must check the Bar3 dialog entry for CS1 on Bar3. Be sure that CS1 on Bar3 does *not* reference CS2 on Bar2. Reference it instead to CS2 on Bar3, which adjoins Ground_2.

6 Rerun the model.

Note that the motion is different from the manually assembled case.



Cutting Closed Loops

Simulink cannot solve models whose block diagrams contain closed loops. To simulate a model containing closed loops, SimMechanics internally converts a closed-loop model to an open-topology model, by cutting each of the model's closed loops once, at a joint, constraint, or driver block.

You can specify a joint to cut if the loop does not contain a disassembled joint, constraint, or driver. To do this, open the joint's dialog box and select the **Mark as the preferred cut joint** check box on the **Advanced** pane in that joint's dialog **Parameters** area.

SimMechanics follows these loop-cutting rules.

- If a loop contains a constraint, driver, or disassembled joint, SimMechanics cuts the loop at one of these blocks, regardless of your preferred cut joint. Selecting a preferred cut joint has no effect in this case.
- If the loop does not contain a constraint, driver, or disassembled joint, SimMechanics cuts the loop at the preferred cut joint if you have specified one.
- Otherwise, SimMechanics cuts the loop at the joint with the most degrees of freedom.

Note SimMechanics cuts a loop at a disassembled joint, constraint, or driver if one or more of these is present, regardless of your preferred cut choice.

Displaying the Cut Joints

To display automatically cut joints in your model, select the **Mark automatically cut joints** check box in the **Diagnostics** area of the **SimMechanics** node of your model's **Configuration Parameters** dialog. See "Configuring SimMechanics Simulation Diagnostics" on page 5-12.

For More About Disassembled and Cut Joints

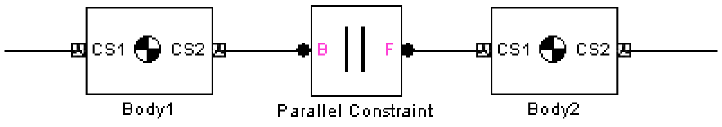
Refer to "Modeling with Disassembled Joints" on page 4-33 for more on disassembled joints. Consult "Verifying Machine Topology" on page 4-74 to see how SimMechanics analyzes closed loops in the model diagram.

For More About Constraints and Drivers

SimMechanics internally represents a cut Joint, Constraint, or Driver as an invisible constraint. See "Modeling Constraints and Drivers" on page 4-38 for more about these specialized blocks.

Modeling Constraints and Drivers

The SimMechanics Constraints & Drivers Library provides a set of blocks to model constraints on the relative motions of two bodies. You model the constraint by connecting the appropriate Constraint or Driver block between the two bodies. As with joints, the blocks each have a base and follower connector port, with the body connected to the follower port viewed as moving relative to the body connected to the base port. For example, the following model constrains Body2 to move along a track that is parallel to the track of Body1.



The blocks enable you to model time-independent constraints or time-dependent drivers.

- Constraint and unactuated Driver blocks model *scleronomic* (time-independent) constraints.
- Actuated Driver blocks (see "Actuating a Driver" on page 4-57) model *rheonomic* (time-dependent) constraints.

Scleronomic constraints lack explicit time dependence; that is, their time dependence appears only implicitly through the coordinates $\boldsymbol{x}$. Rheonomic constraints have explicit time dependence as well, in addition to implicit time dependence through the $\boldsymbol{x}$.

Holonomic constraint functions depend only on body positions, not velocities:

$$f(\boldsymbol{x}_B, \boldsymbol{x}_F; t) = 0$$

Constraints of the form

$$g(\boldsymbol{x}_B, \dot{\boldsymbol{x}}_B, \boldsymbol{x}_F, \dot{\boldsymbol{x}}_F; t) = 0$$

can sometimes be integrated into a form dependent only on positions; but if not, they are *nonholonomic*. For example,

- The one-dimensional rolling of a wheel of radius R along a line (the x-axis) imposes a holonomic constraint, $x = R\theta$.
- The two-dimensional rolling of a sphere of radius R on a plane (the xy-plane) imposes a nonholonomic constraint, $ds = R \cdot d\theta$, with $ds^2 = dx^2 + dy^2$. This constraint is nonholonomic because there is not enough information to solve the constraint independently of the dynamics.

What Constraints and Drivers Do

Constrained and driven bodies are still free to respond to externally imposed forces/torques, but only in a way consistent with the constraints.

Constraints and drivers can only remove degrees of freedom from a machine. Constraints and unactuated Drivers prevent the machine from moving in certain ways. Unactuated Drivers hold the constrained degrees of freedom between the connected pair of bodies in their initial state. Actuated Drivers externally impose a relative motion between pairs of bodies, starting with the bodies' initial state. See "Counting Degrees of Freedom" on page 4-77.

This section discusses modeling constraints and drivers in a general way.

- "Directionality of Constraints and Drivers" on page 4-39
- "Solving Constraints" on page 4-40
- "Restrictions on Using Constraint and Driver Blocks" on page 4-40

The section ends with two examples, "Constraint Example: Gear Constraint" on page 4-41 and "Driver Example: Angle Driver" on page 4-43.

See the reference pages for information on the specific constraint that a Constraint or Driver block imposes.

Directionality of Constraints and Drivers

Like joints, constraints and drivers have directionality. The sequence of base to follower body determines the directionality of the constraint or driver. The directionality determines how the sign of Driver Actuator signals affects the motion of the follower relative to the base and the sign of signals output by constraint and driver sensors.

Solving Constraints

When simulating a model, SimMechanics uses a constraint solver to find the motion, given the model's Constraint and Driver blocks. You can specify both the constraint solver type and the constraint tolerances that SimMechanics uses to find the constraint solution. See "Implementing Constraints" on page 5-6 for more information.

Mitigating Constraint Singularities

Some constraints, whether time-independent (Constraints) or time-dependent (Drivers), can become singular when the constrained bodies take on certain relative configurations; for example, if the two body axes line up when the Bodies are connected by an Angle Driver. The simulation slows down as a constraint becomes singular.

If you find a constrained model running slowly, consider selecting the **Use robust singularity handling** option in the **Constraints** pane of your machine's Machine Environment block dialog. See "Handling Motion Singularities" on page 5-9.

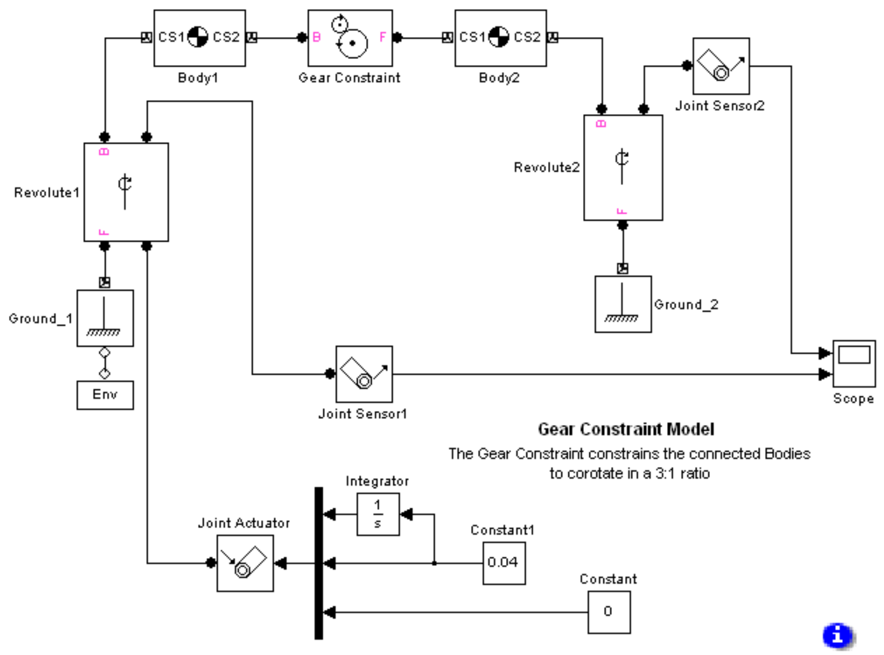
Restrictions on Using Constraint and Driver Blocks

The following restrictions apply to the use of Constraint and Driver blocks in a model:

- Constraint and Driver blocks can appear only in closed loops.
- A closed loop cannot contain more than one Constraint or Driver block.
- A Constraint or Driver must connect exactly two Bodies.

Constraint Example: Gear Constraint

The `mech_gears` model illustrates the Gear Constraint. Open the Body and Gear Constraint blocks.



Body1 and Body2 have their CG positions 2 meters apart. CS1 and CS2 on Body1 are collocated with the Body1 CG, and similarly, CS1 and CS2 on Body2 are collocated with the Body2 CG.

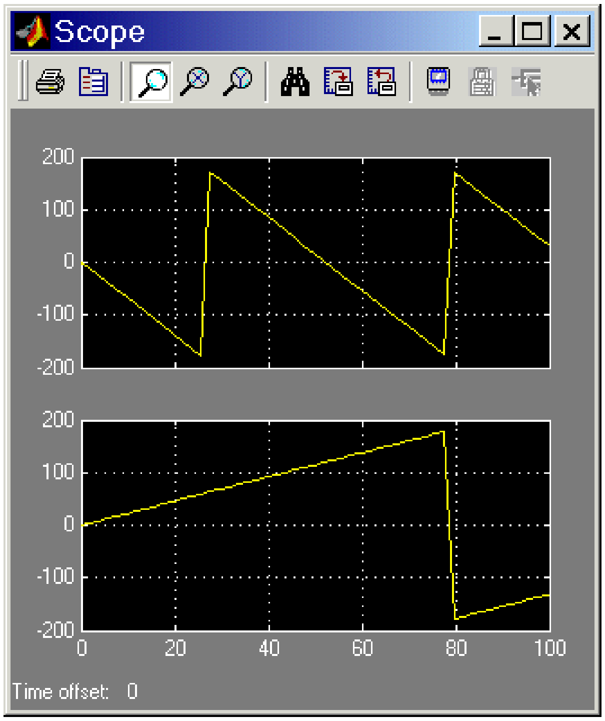
The Gear Constraint between them has two pitch circles. One is centered on the CS2 at the base Body, which is Body1, and has radius 1.5 meters. The other is centered on CS1 at the follower Body, which is Body2, and has radius 0.5 meters. The distance between CS2 on Body1 and CS1 on Body2 is 2 meters. The sum of the pitch circle radii equals this distance, as it must.

Visualizing the Gear Motion

The model is set up to open the visualization feature automatically upon simulation start, with MATLAB Graphics and convex hulls, as explained in

"Introducing the SimMechanics Visualization Window" on page 6-11. Start the simulation and watch the CG CS axis triads spin around. The CG triad at Body2 rotates three times faster than the CG triad at Body1, because the pitch circle centered on Body2 is three times smaller.

You can see the same behavior in the Scope. The upper plot shows the motion of Revolute2, and the lower plot the motion of Revolute1. Note that angular motion is mapped to the interval (-180º, +180º] degrees.



The Gear Constraint is inside a closed loop formed by

Ground_1–Revolute1–Body1–Gear Constraint–Body–Revolute2–Ground_2

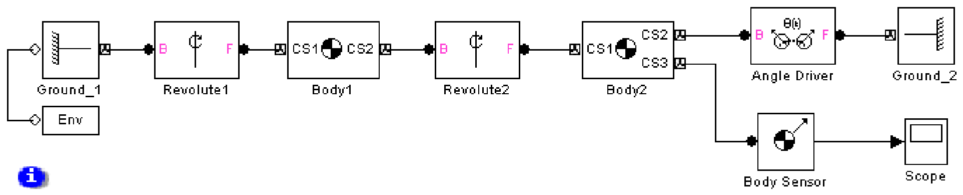
Although Ground_1 and Ground_2 are distinct blocks, they represent different points on the same immobile ground at rest in World. So the blocks form a loop.

Driver Example: Angle Driver

The following two models illustrate the Angle Driver, both without and with a Driver Actuator.

The Angle Driver Without a Driver Actuator

The first is `mech_angle_unact`. Open the Body2 block.



The bodies form a double pendulum of two rods. The Body Sensor is connected to Body2 at CS3 = CS2 and measures all three components of Body2's angular velocity vector with respect to the ground.

The Angle Driver is connected between Body2 and Ground_2. Because the Angle Driver is not actuated in this model, it acts during the simulation as a time-independent constraint to hold the angle between Body2 and Ground_2 constant at its initial value.

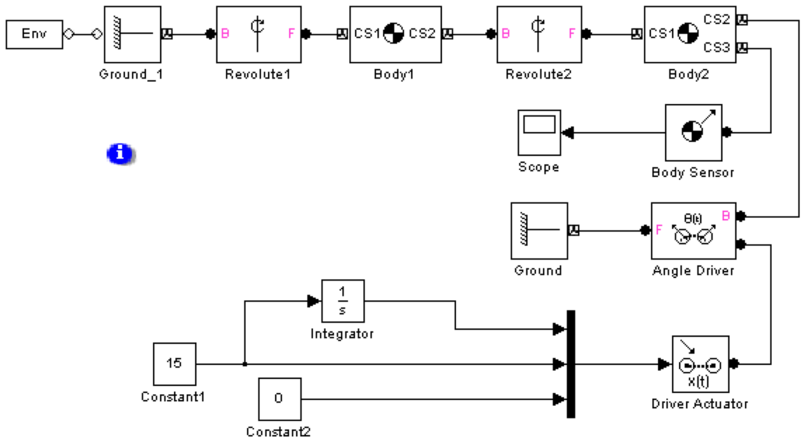
Visualizing the Angle Driver Motion

The model is set up to open the visualization feature automatically upon simulation start, with MATLAB Graphics and convex hulls, as explained in "Introducing the SimMechanics Visualization Window" on page 6-11.

Start the simulation. The upper body swings like a pendulum, but the lower body maintains its horizontal orientation with respect to the horizontal ground. The Scope measures Body2's angular velocity with respect to ground, and this remains at zero.

The Angle Driver With a Driver Actuator

The second model is `mech_angle_act`. Open the Driver Actuator block.



The Driver Actuator drives the Angle Driver block. Here, the Actuator accepts a constant angular velocity signal from the Simulink blocks. The Actuator also requires the angle itself and the angular acceleration, together with the angular velocity, in a vector signal format. The Angle Driver's angle signal is added to the angle's initial value.

The Body Sensor again measures three components of Body2's angular velocity with respect to the ground. Constant1 drives the angle at 15°/second. While the simulation is running, this angle changes at the constant rate. At the same time, the assembly and the constant length of the two pendulum rods must be maintained by Simulink, while both rods are subject to gravity. As the two axes line up, the mutual constraint between the bodies enforced the Driver becomes singular. The simulation slows down.

As in the Gear Constraint model, the two Ground blocks in these models represent points on the same immobile ground at rest in World, so the Angle Driver is part of a closed loop.

Modeling Actuators

The SimMechanics Actuators & Sensors Library provides a set of Actuator blocks that enable you to apply time-dependent forces and motions to bodies, joints, and drivers. You can also vary a body's mass and inertia tensor.

Caution SimMechanics allows you to connect an Actuator to a Ground. But it displays an error if you attempt to simulate or update a model containing such a connection. This is because ground is immobile and cannot be actuated.

You can use Actuator blocks to perform the following tasks:

- Apply a time-varying force or torque to a body or joint. See "Actuating a Body" on page 4-46 and "Actuating a Joint" on page 4-52.
- Specify the position, velocity, and acceleration of a joint or driver as a function of time. See "Actuating a Joint" on page 4-52 and "Actuating a Driver" on page 4-57.
- Specify the initial position and velocity of a joint primitive. See "Specifying Initial Positions and Velocities" on page 4-57.
- Specify the mass and/or inertia tensor of a body as a function of time. See "Varying a Body's Mass and Inertia Tensor" on page 4-49.

In general, actuators can apply any combination of forces and motions to a machine provided that

- The applied forces and motions are consistent with each other and with the machine's geometry, constraints, and assembly restrictions.
- It is possible to find a unique solution for the motion of each actuated degree of freedom (DoF).

Stabilizing Numerical Derivatives in Actuator Signals

To actuate bodies, joints, and drivers, you often need to differentiate an incoming Simulink actuation signal. A typical example is motion actuation of a joint, which requires position, velocity, and acceleration of each joint primitive as a function of time. You specify this information as a set of Simulink signals.

Simulink provides a Derivative block for numerical differentiation of a signal. However, this block's output is often not stable or accurate enough for use in SimMechanics. Here are recommended alternatives to the Derivative block.

- To differentiate a signal, use a transfer function block (Transfer Fcn). This block actually performs a combination of differentiation and integration with a Laplace transform, acting to smooth the output, which is not exactly the derivative.
- Start by specifying the highest-derivative signal (such as an acceleration), then integrate this signal to obtain lower-derivative signals (such as velocity and position) using the Integrator block.

The first method is illustrated by the `mech_stewart_control` model. For an example of the second method, see the `mech_body_driver` model.

Actuating a Body

You can use the Body Actuator to apply forces and/or torques, but not motions, to bodies. (You can apply motions to a body indirectly, using Joint Actuators. See "Applying Motions to Bodies" on page 4-48.)

To actuate a body,

1. If there is not already an unused connector port available for the Actuator create a Body CS port on the Body for the Actuator. See the Body block reference if you need to learn how.

2. Drag a Body Actuator block from the Sensors & Actuators library into your model and connect its output port to a Body CS port on the Body.

3. Open the Actuator's dialog box.

4. Choose to apply a force or torque to the body:

   - Select the **Applied force** check box if you want to apply a force to the body, and select the units of force from the adjacent list.

- Select the **Applied torque** check box if you want to apply a torque to the body, and select the units of torque from the adjacent list.

5 Select the coordinate system used to specify the applied torque from the **With respect to CS** list.

   The list allows you to choose either the World CS or the Body CS of the port to which you attached the Actuator.

6 Create vector signals that specify the value of the applied torque and force at each time step.

   You can use any Simulink source block (for example, an Input port block or a Sine Wave block) or combination of Simulink blocks to create the Body Actuator signal. You can also use the output of a Sensor block connected to the Body as input to the Actuator, thereby creating a feedback loop. Such loops are useful for modeling springs and dampers (see “Checking Model Validity” on page 4-74).

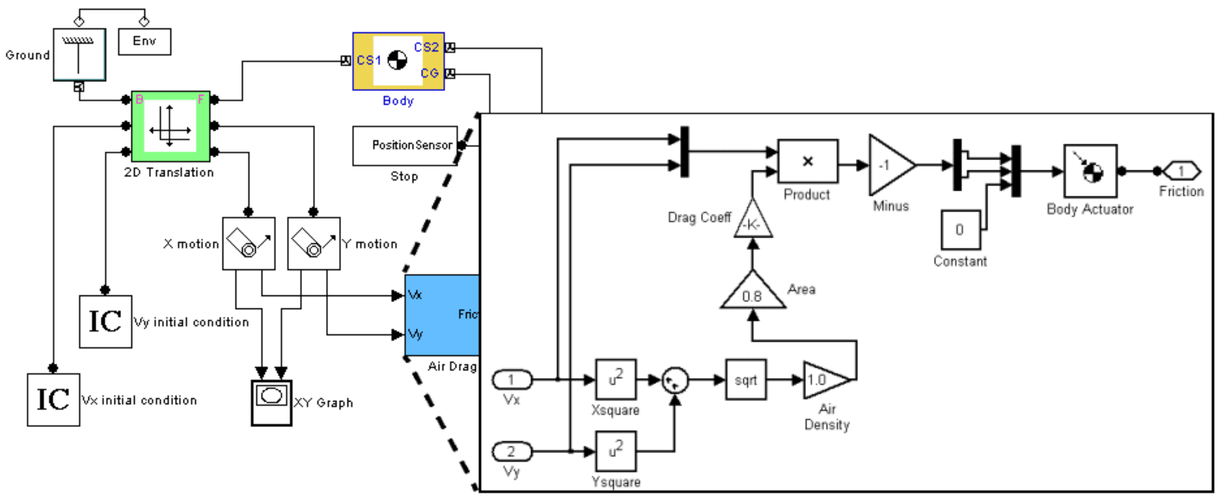
7 Connect the force and/or torque signal to the input port of the Actuator.

   If you are applying both a force and a torque to the body, connect the force and torque signals to the inputs of a two-input Mux block. Then connect the output of the Mux block to the input of the Actuator.

Body Actuator Example: Pure Kinetic Friction

The `mech_ballistic_kin_fric` model in the Demos library provides an example of how to implement pure kinetic friction. This type of friction is a continuous force that depends on a body’s motion relative to a medium (such as air), as well as on physical characteristics of the body. Kinetic friction, unlike “stiction,” involves no “sticking” or locking of motion, and the friction is not discontinuous. While you could use the Joint Stiction Actuator, this is not necessary. This model applies air friction or drag to a projectile with a Body Actuator.

Open the Air Drag subsystem. If you double-click the block, a mask dialog box opens asking for the drag coefficient `Cd`. If you right-click the block and select **Look under mask**, the subsystem itself appears:



The Air Drag subsystem computes the air friction according to a standard air friction model. (See the Aerospace Blockset documentation for more information.) The drag always opposes the projectile's motion and is proportional to the product of the air density, the projectile's cross-sectional area, and the square of its speed.

Run the model with the default drag coefficient (zero). The XY Graph window opens to plot the parabolic path of the projectile. Now open the Air Drag dialog again and experiment with different drag coefficients C_d. Start with small values such as $C_d = 0.05$. For a rigid sphere, C_d is two. The effect of the drag is dramatic in that case.

Applying Motions to Bodies

The Body Actuator block cannot actuate a Body with motion signals. But you can construct such body motion actuators with a combination of other blocks. See "Joint Actuator Example: Body Driver" on page 4-54.

Varying a Body's Mass and Inertia Tensor

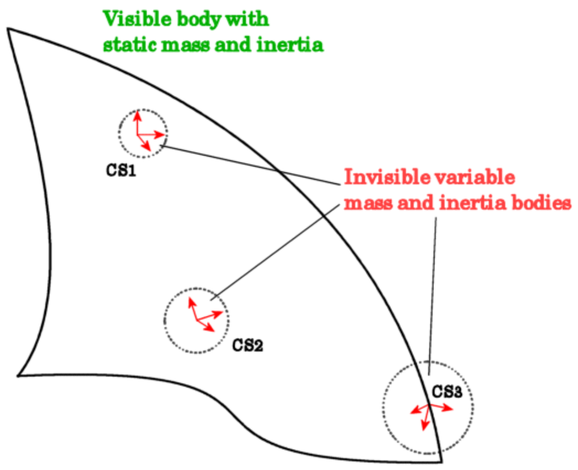
The Variable Mass & Inertia Actuator block gives you a way to vary a body's mass and/or inertia tensor as external functions of time. You specify these functions with incoming Simulink signals.

Caution The Variable Mass & Inertia Actuator block does not apply any thrust forces or torques to the Body so actuated. Mass loss or gain in a particular direction results in thrust forces and torques on the body. You must apply these forces/torques to the Body separately with Body Actuator blocks.

The variable mass/inertia actuator affects a body's motion only when you apply forces/torques on the body. When a body's motion is determined only by initial conditions, changing the mass or inertia tensor of a body does not affect its motion, because the variable mass/inertia actuator does not apply forces/torques to the body.

The Variable Mass & Inertia Actuator block changes the actuated Body's mass and rotational inertia by attaching an invisible body to the actuated body at a particular Body coordinate system (CS). This invisible body has a mass and an inertia tensor that vary in time as specified by the Actuator's external Simulink signal. SimMechanics treats the actuated body and the invisible body as a single composite body. The composite body has a new mass, new center of gravity (CG), and new inertia tensor compounded from its two constituent bodies.

You can add multiple Variable Mass & Inertia Actuator blocks to one Body. In that case, SimMechanics treats the actuated body and all attached invisible bodies as a single composite body. This composite body's mass, CG, and inertia tensor are compounded from its constituent bodies.


Visible body with static mass and inertia
CS1
Invisible variable mass and inertia bodies
CS2
CS3

Attaching Variable Mass and Inertia Bodies to a Visible Body

To vary the mass and/or inertia tensor of a Body with this Actuator:

1. From the Sensors & Actuators library, drag a Variable Mass & Inertia Actuator block into your model.
2. Attach the Actuator's connector port to the Body CS on the Body where you want the invisible variable mass to be. If a suitable Body CS port does not exist on the Body, open its dialog and create one.
3. Create an external Simulink signal to model the time-varying mass and/or inertia tensor for this invisible body. Connect it to the Variable Mass & Inertia Actuator block's Simulink input port.

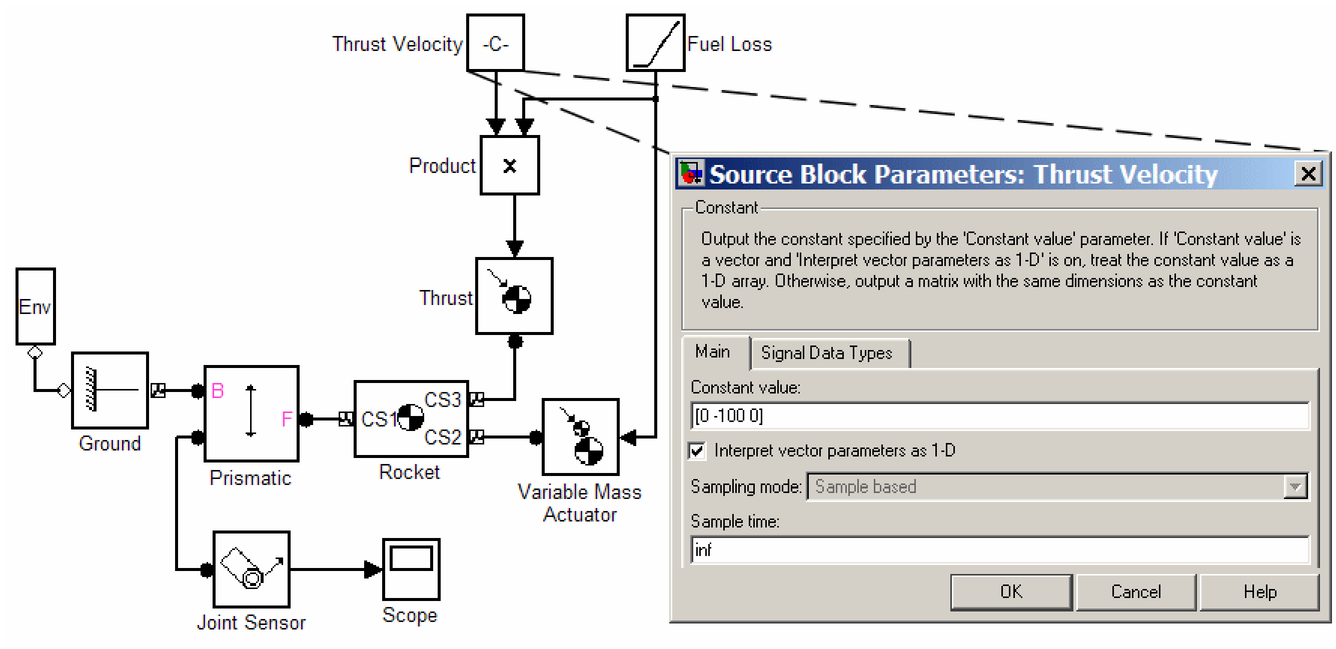
   This Simulink signal can have one, nine, or ten components, depending on whether you are varying the mass only, the inertia tensor only, or both.

Example: Simple Rocket

The following model simulates a simple rocket. It treats the rocket as a point mass moving upward (+*y* direction) with an exhaust pointing downward (-*y* direction). The rocket loses mass at a constant rate.

The Rocket block is the point mass. The Thrust Velocity block represents the downward exhaust and, multiplied by the mass loss represented by the Fuel Loss block, actuates the Rocket body with a thrust force pointing upward. The Thrust block (a body actuator) applies this force at the *local* Body CS, which, for a point rocket, is identical to the Rocket's CG CS.

The same mass loss from the Fuel Loss block that produces the thrust force also must vary the rocket's mass directly. The Variable Mass Actuator block accomplishes this by feeding the same mass loss signal to the Rocket block.



Constant

Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.

Constant value:
[0 -100 0]

☑ Interpret vector parameters as 1-D

Sampling mode: Sample based

Sample time:
inf

Actuating a Joint

You individually actuate each of the prismatic and revolute primitives of an assembled joint with a Joint Actuator. You can apply

- Forces or translational motions (but not both) to prismatic primitives
- Torques or rotational motions (but not both) to revolute primitives

---

Caution You cannot actuate spherical or weld primitives, disassembled joints, or massless connectors.

SimMechanics allows you to connect multiple Actuators to the same joint primitive. But it halts and displays an error message if you attempt to update or simulate a model containing such a connection.

Exception: You can apply a Joint Initial Condition Actuator and force or torque actuation (including stiction) to the same primitive. You cannot apply a Joint Initial Condition Actuator and motion actuation to the same primitive. See "Specifying Initial Positions and Velocities" on page 4-57.

To actuate a prismatic or revolute joint primitive of an assembled joint:

1. Create an Actuator port on the Joint block for the primitive (see "Creating Actuator and Sensor Ports on a Joint" on page 4-28).
2. Drag a Joint Actuator or Joint Stiction Actuator from the Sensors & Actuators library into your model and connect its output port to the Actuator port on the Joint.

   The remaining steps in this procedure apply to the creation of a standard Joint Actuator. For information on creating a stiction actuator, which applies classical Coulombic friction to a prismatic or revolute joint, see the Joint Stiction Actuator block reference page.
3. Open the Joint Actuator's dialog box.
4. Select the primitive you want to actuate from the **Connected to primitive** list on the dialog box.

5. Select the type of actuation you want to apply from the **Actuate with** pull-down menu, either `Generalized Forces` or `Motion`.

6. If you are actuating a prismatic primitive:

   - If you selected `Generalized Forces` as the actuation type, select the units of force from the **Applied force units** list.
   - If you selected `Motion` as the actuation type, select the units for each motion to be actuated (position, velocity, acceleration).

7. If you are actuating a revolute primitive:

   - If you selected `Generalized Forces` as the actuation type, select the units of torque from the **Applied torque units** list.
   - If you selected `Motion` as the actuation type, select the units for each motion to be actuated (angle, angular velocity, angular acceleration).

8. Click **OK** to apply your choices and dismiss the dialog box.

   Each joint primitive that you motion-actuate is lost as a true degree of freedom in your machine. That is because the DoF can no longer respond freely to externally applied forces or torques. See "Counting Degrees of Freedom" on page 4-77.

9. Create a signal that specifies the applied force, torque, or motions at each time step.

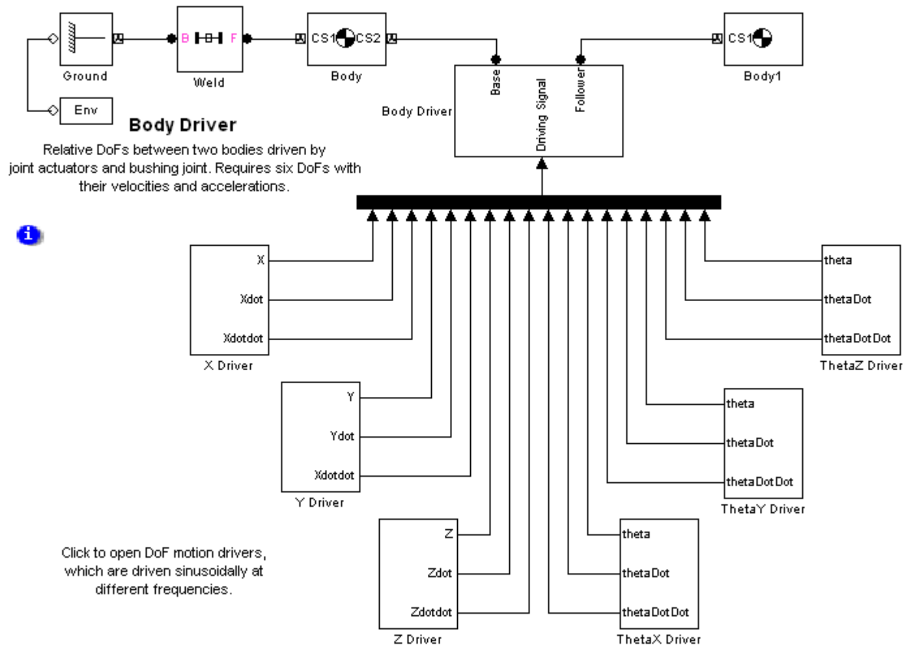
   You can use any Simulink source block or any combination of blocks to create the actuator signal. You can also connect the output of a Sensor block attached to the Joint to the Actuator input, thereby creating a feedback loop. You can use such loops to model springs and dampers attached to the joint.

   A force or torque signal must be a scalar signal. A motion signal must be a 1-D array signal comprising three components: position, velocity, and acceleration. The directionality of the joint determines the response of the follower to the sign of the actuator signal (see "Joint Directionality" on page 4-24).

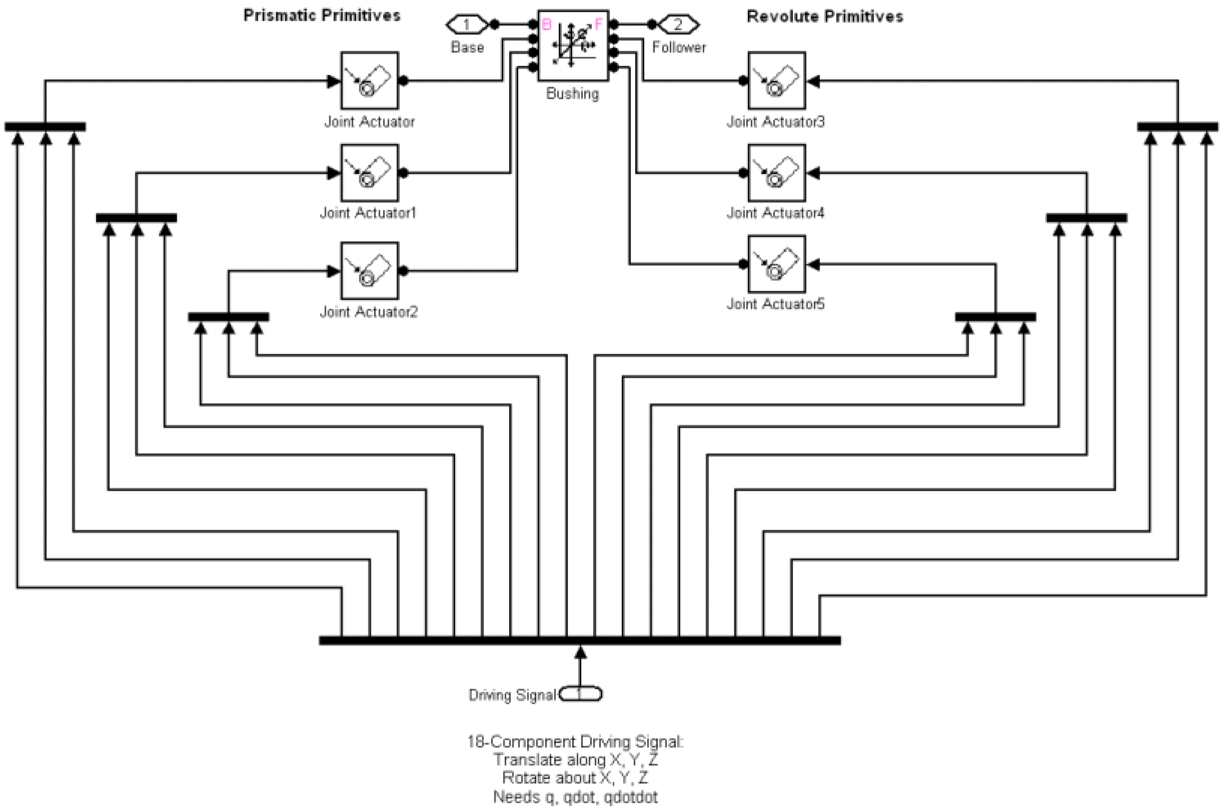
10. Connect the Actuator signal to the Actuator port on the Joint.

Joint Actuator Example: Body Driver

The `mech_body_driver` model illustrates the use of Joint Actuators to create a custom driver.



The Body Driver subsystem accepts an 18-component signal that feeds the coordinates, velocities, and accelerations for all six relative DoFs between Body and Body1. The subsystem uses a Bushing block that contains three translational and three rotational primitives to represent the relative DoFs:

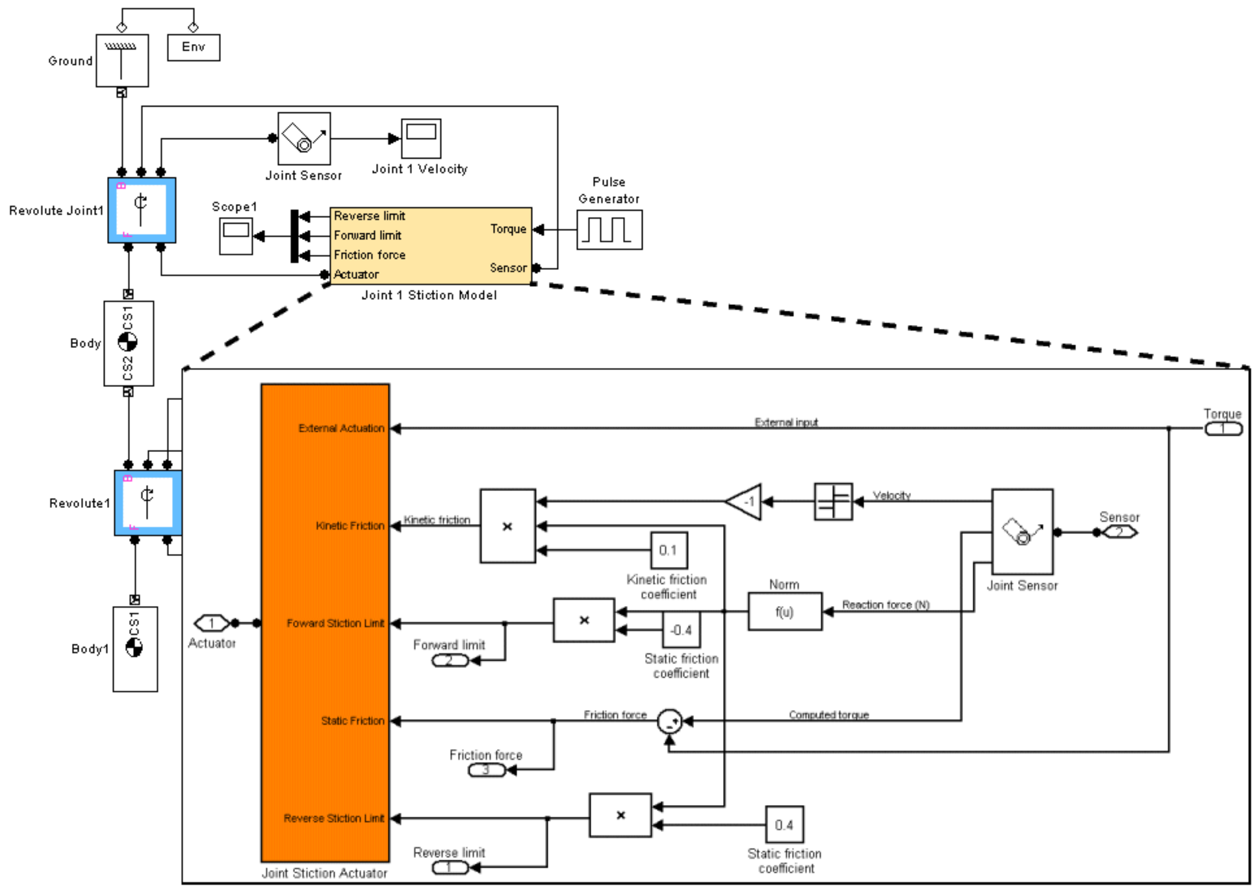


You can modify the body driver to move only one of the bodies, thereby creating a motion actuator. To move Body1 relative to World, for example, remove the blocks Body and Weld and connect the subsystem Body Driver directly to Ground.

Joint Stiction Actuator Example: Mixed Static and Kinetic Friction

The `mech_dpen_sticky` model in the Demos library illustrates a driven double pendulum, with "sticky" friction or stiction applied to both revolute joints with the Joint Stiction Actuator block.

Open the unmasked Joint1 or Joint2 Stiction Model blocks (marked in yellow) to view the subsystems:



Each Stiction subsystem contains a Joint Stiction Actuator block (marked in orange) that requires static and kinetic friction coefficients via their respective blocks. For either revolute, an angular velocity threshold, specified through the block dialog, determines if a joint locks. Once locked, the joint cannot move until a combination of forces reaches a threshold specified by the `Forward Stiction Limit` or `Reverse Stiction Limit`.

Run the model with different kinetic and static friction coefficients and different velocity thresholds. View the results in the Scope blocks and through a visualization window. You can find more details on how stiction works in SimMechanics by consulting the Joint Stiction Actuator block reference page.

Actuating a Driver

Actuating a Driver with a Driver Actuator allows you to specify the time dependence of the rheonomic constraint applied by the Driver.

To actuate a Driver:

1. Create an additional connector port on the Driver for the Actuator.

   Create the additional port in the same way you create an additional Sensor/Actuator port on a Joint (see "Creating Actuator and Sensor Ports on a Joint" on page 4-28).

2. Drag an instance of a Driver Actuator from the Sensors & Actuators library into your model.

3. Connect the Actuator's output port to the Actuator port on the Driver.

4. Create a signal that specifies the time dependence of the Driver constraint.

5. Connect the actuation signal to the input port of the Driver Actuator.

Specifying Initial Positions and Velocities

The Joint Initial Condition Actuator (JICA) block allows you to specify the initial positions and velocities of unactuated joints and hence the bodies attached to them. You can use JICA blocks to

- Specify nonzero initial joint velocities

The default initial velocity of a joint primitive is zero. You must use a JICA block to specify a joint's initial velocity if the initial velocity is not zero.

- Override the initial position settings of a body pair

  The CG CS origin settings in the dialog boxes of Body blocks specify the bodies' initial positions. Using JICA blocks, you can override these initial body positions by resetting their relative positions in the Joints connecting them.

When your model simulation starts, SimMechanics first puts your machines into their home configurations with the Body dialog data. It then moves your machines to their initial configurations by applying JICA data.

Caution You cannot simultaneously actuate a joint primitive with a Joint Initial Condition Actuator and motion actuation from a Joint Actuator block.

Using JICA Blocks

Specifying initial conditions on a joint primitive is a special kind of actuation, one that occurs only once at the beginning of simulation. That is why the JICA block resides in the Sensors & Actuators library.

Note A JICA block, unlike other Actuators, does not have an input port. The JICA's dialog box specifies the Actuator input completely.

With a JICA block, you can specify the initial positions and velocities of any combination of prismatic and revolute primitives within a given Joint. (You cannot specify ICs for spherical and weld primitives.)

To specify the initial velocity and/or position of a joint primitive:

1. Drag a JICA block from the Sensors & Actuators library and drop it into your model window.

2. Create an additional connector port on the Joint block containing the primitive whose initial condition you want to specify.

3 Connect the connector port on the JICA block to the new connector port on the Joint block.

Caution Do not connect the JICA block to the Joint ports marked "B" or "F" (base or follower). These ports are intended for connecting to Bodies.

4 Open the JICA block's dialog box. From the primitive list for the Joint, choose the primitives you want to actuate by selecting their check boxes.

5 Enter the initial positions of the actuated primitives, relative to the Body CSs attached to the Joint, in the **Position** field.

From the pull-down menu on the right, select **Units** for the initial positions.

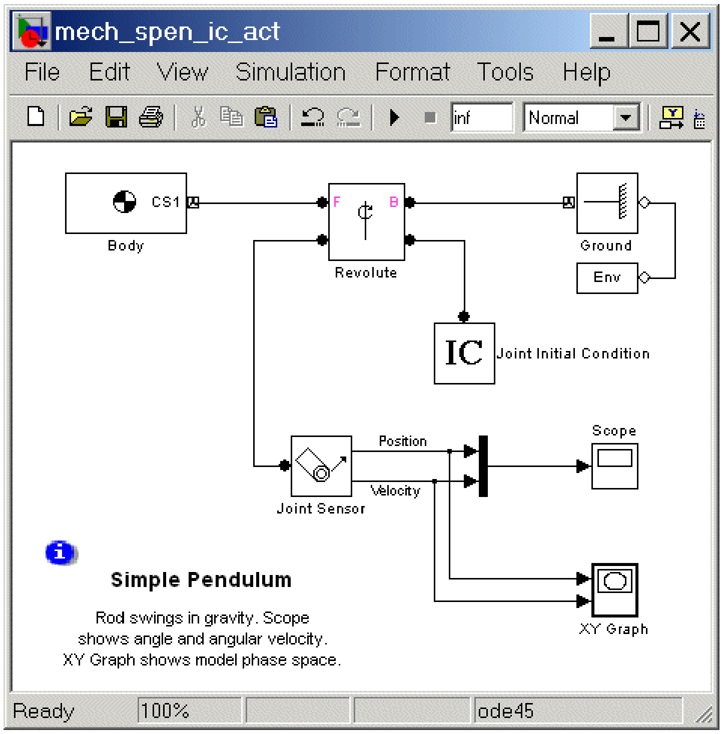
6 Enter the initial velocities of the actuated primitives, relative to the Body CSs attached to the Joint, in the **Velocity** field.

From the pull-down menu on the right, select **Units** for the initial velocities.

7 Click **Apply** or **OK**.

JICA Example: A Simple Pendulum

Open `mech_spen` from the Demos library, then open the Sensors & Actuators library. Follow the steps from the preceding section, “Using JICA Blocks” on page 4-58, to connect one Joint Initial Condition Actuator block to the Revolute block and configure it. This Joint contains only one primitive, `R1`, which is the primitive listed in the JICA dialog box.



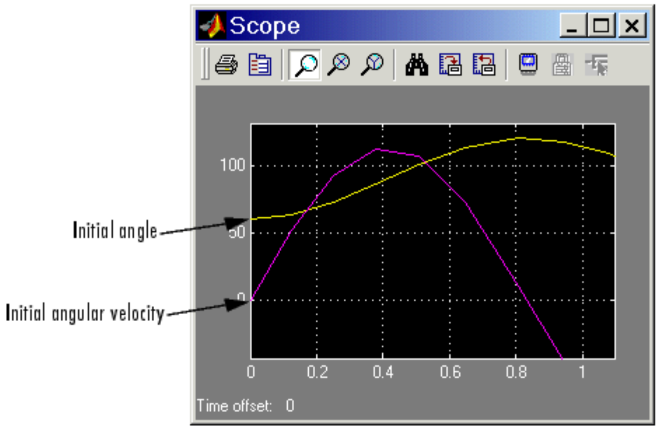
Simple Pendulum

Rod swings in gravity. Scope shows angle and angular velocity. XY Graph shows model phase space.

Set the initial conditions in two ways and compare the resulting simulations in the scope:

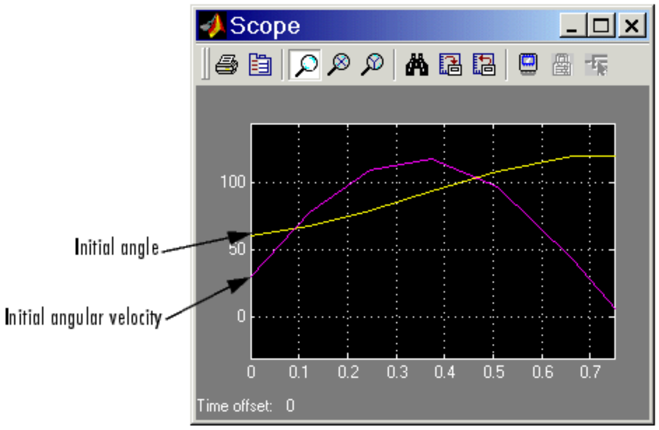
1 First set the initial **Position** (angle) to `60 deg`, which is 60° down from the left horizontal (30° clockwise from vertically down), and set the initial **Velocity** to `0 deg/s`.

2 Run the simulation for one second. Note in the scope that the initial angle (yellow curve) is displaced upward to 60°, while the initial velocity (purple curve) still starts at zero.



3 Now reset the initial **Velocity** to `30 deg/s`, leaving the initial **Position** (angle) at `60 deg`.

4 Rerun the simulation for one second. Note in Scope that the initial angle is still displaced upward to 60°, but the initial velocity is also displaced upward to 30°/sec.



The joint directionality is assigned in `mech_spen` so that the positive rotation axis is the +*z*-axis. Looking from the front, positive rotation swings down and right, counterclockwise.

Modeling Sensors

The SimMechanics Sensors & Actuators library provides a set of Sensor blocks that enable you to measure

- Body motions (see “Sensing Body Motions” on page 4-63)
- Joint motions and forces or torques on joints (see “Sensing Joint Motions and Forces” on page 4-65)
- Constraint reaction forces and torques (see “Sensing Constraint Reaction Forces” on page 4-65)

Note You can feed Sensor output back into Actuator blocks to model springs, dampers, and other mechanical devices that depend on force feedback. See “Actuating a Body” on page 4-46, “Actuating a Joint” on page 4-52, “Modeling Force Elements” on page 4-69, and “Checking Model Validity” on page 4-74.

Home Configuration and Position-Orientation Measurements

The Body and Joint Sensor blocks can measure the position and/or orientation of bodies and degrees of freedom. They make these measurements relative to the home configuration of the machine, the machine state *before* the application of initial condition actuators and assembly of disassembled joints. Thus motion sensors include the effect of the latter, which act before the simulation starts.

For further discussion, see “Modeling with Disassembled Joints” on page 4-33 and “Specifying Initial Positions and Velocities” on page 4-57, and “Kinematics and the Machine’s State of Motion” on page 3-2.

Sensing Body Motions

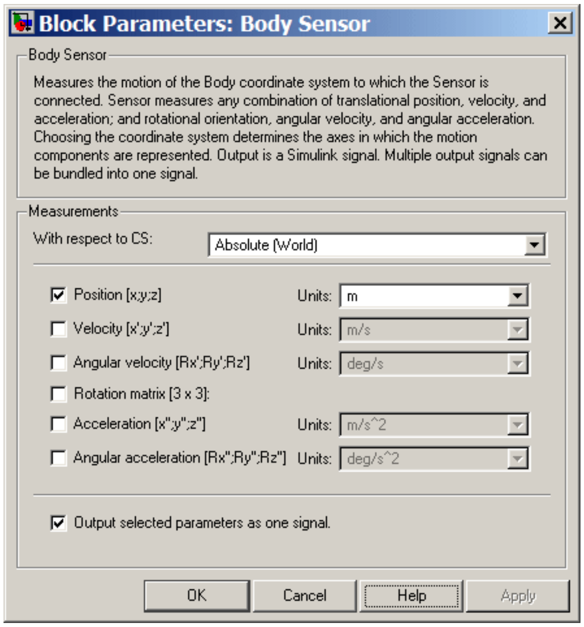
To sense the position, velocity, or acceleration of a body represented by a Body block with a Body Sensor:

1 If the Body block does not have a spare local coordinate system with a Body CS port, create one (see "Managing Body Coordinate Systems" on page 4-18).

2 Drag a Body Sensor block from the Sensors & Actuators library into your model.

3 Connect its connector port to a spare Body CS port on the Body.

4 Open the Sensor's dialog box.


Block Parameters: Body Sensor

Body Sensor

Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

Measurements

With respect to CS: Absolute (World)

[✓] Position [x;y;z]	Units: m
[ ] Velocity [x';y';z']	Units: m/s
[ ] Angular velocity [Rx';Ry';Rz']	Units: deg/s
[ ] Rotation matrix [3 x 3]:	
[ ] Acceleration [x";y";z"]	Units: m/s^2
[ ] Angular acceleration [Rx";Ry";Rz"]	Units: deg/s^2

[✓] Output selected parameters as one signal.

5 Select the coordinate system relative to which the sensor measures its output from the **With respect to CS** list.

6 Select the check boxes next to the motions that you want to sense (see the Body Sensor block reference page).

7 If you have chosen to sense more than one type of motion and want the Sensor to multiplex the motions into a single output signal, select the **Output selected parameters as one signal** check box.

8 Click **OK** or **Apply**.

9 Connect the output of the Body Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

Sensing Joint Motions and Forces

The Joint Sensor block enables you to measure the motions of degrees of freedom. It can also measure the relative forces and torques between the bodies connected to the joint. These include the *computed force* or *torque* (the force or torque needed to reproduce the joint's motion) and the *reaction force* and *torque* on a joint primitive. (You cannot measure the computed force or torque on a spherical or weld primitive.) You must connect a separate Joint Sensor block to a Joint block for each joint primitive that you want to sense.

To sense the motions, forces, and torques of a joint primitive contained by a Joint block:

1 If the Joint block does not have a spare Sensor port, create one (see "Creating Actuator and Sensor Ports on a Joint" on page 4-28).

2 Drag a Joint Sensor block from the Sensors & Actuators library into your model.

3 Connect its connector port to the spare Sensor port on the joint.

4 Use the Sensor's dialog box to configure the Sensor to measure the motions, forces, and torques that you want to measure (see the Joint Sensor block reference page).

5 Connect the output of the Joint Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

Sensing Constraint Reaction Forces

The Constraint & Driver Sensor block enables you to measure the reaction forces and torques induced on the constraints modeled by SimMechanics Constraint and Driver blocks.

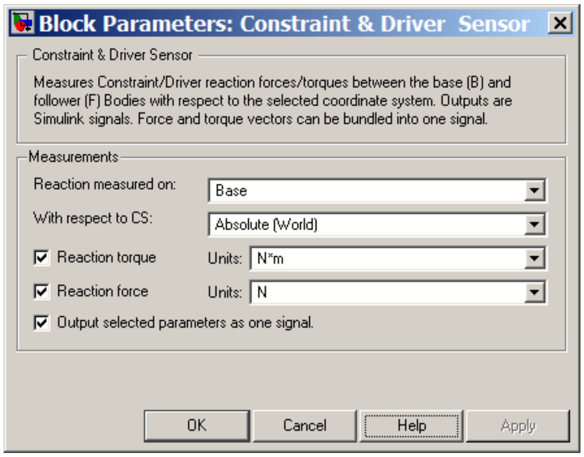
To sense the reaction force and/or torque induced by a constraint or driver,

1 If the Constraint or Driver does not have a spare Sensor port, create one.

2 Drag a Constraint & Driver Sensor block from the Sensors & Actuators library into your model.

3 Connect its connector port to a Sensor port on the Constraint or Driver block.

4 Open the Sensor block's dialog box.



Block Parameters: Constraint & Driver Sensor

Constraint & Driver Sensor

Measures Constraint/Driver reaction forces/torques between the base (B) and follower (F) Bodies with respect to the selected coordinate system. Outputs are Simulink signals. Force and torque vectors can be bundled into one signal.

Measurements

Reaction measured on: Base

With respect to CS: Absolute (World)

☑ Reaction torque Units: N*m

☑ Reaction force Units: N

☑ Output selected parameters as one signal.

5 Select the body (follower or base) on which to measure the reaction force from the **Reactions measured on** list.

6 Select the coordinate system relative to which the Sensor measures its output from the **With respect to coordinate system** list.

7 Select the **Reaction torque** check box if you want the Sensor to output the reaction torque on the base (or follower) body.

8 Select the **Reaction force** check box if you want the Sensor to output the reaction force on the base (or follower) body.

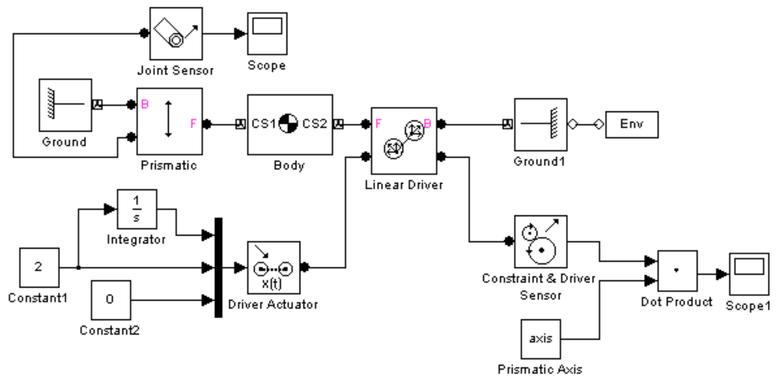
9 If you have chosen to output both reaction force and torque and want the Sensor to multiplex them into a single output signal, select the **Output selected parameters as one signal** check box.

10 Click **OK** or **Apply**. Connect the output of the Constraint & Driver Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

Not all the reaction force/torque components are significant. Only those components projected into the subspace of constrained or driven degrees of freedom (DoFs) are physical. Components orthogonal to the constrained or driven degrees of freedom are not physical.

Example: Linear Driver

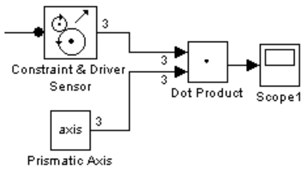
In this example, you drive a body along the *x*-axis, but only allow it a prismatic DoF tilted at an angle in the *x*-*y* plane. Construct the following model.



Configure the Constraint & Driver Sensor to measure only the reaction force, not the torque. Configure the Linear Driver to drive the Body along the World *x*-axis, but set up the Prismatic with a primitive axis along (1, 2, 0). The body can then move only along this axis, but is driven along the horizontal *x*-axis. Measure all motions and forces in World. Leave all other settings at default.

Open the Scopes and run the model. The measured reaction force lies along the *x*-axis, with a value of -19.62 N (newtons). Because the constrained DoF is not parallel to the *x*-axis, you need to project the reaction force along the unit vector $(1, 2, 0)/\sqrt{5}$ defining the direction of the prismatic primitive to obtain the physical part.

Add to the model the Simulink blocks that form a dot product between the reaction force signal (three components) and the prismatic unit vector (also three components). (You can define a workspace vector for this axis and use it in both the joint and the dot product.) Reconnect Scope1 to measure this physical component of the reaction force.



The physical component of the reaction force is -(19.62 N)·(1/$\sqrt{5}$) = -8.77 N. The component of the reaction force orthogonal to (1, 2, 0) is not physical.

Modeling Force Elements

Internal forces are forces the machine applies to itself as a result of its own motion. Unlike actuation forces, you do not apply these forces from outside the machine with Simulink signals. The body motions instead generate the forces and torques directly.

The Force Elements library provides ready-made blocks to represent certain kinds of internal forces and torques acting between bodies.

- “Inserting a Linear Force Between Bodies” on page 4-69
- “Inserting a Linear Force or Torque Through a Joint” on page 4-71

You can also create your own customized sensor-actuator feedback loops to model springs, dampers, and more complex internal forces.

- “Customizing Force Elements with Sensor-Actuator Feedback” on page 4-72

Inserting a Linear Force Between Bodies

A generalized linear force between two bodies is a linear function of the two bodies’ relative displacement vector ***r*** and relative velocity ***v***, with constant coefficients. The Body Spring & Damper block models a force acting between two bodies along the axis ***r*** connecting them:

$$F = -k(r - r_0) - bv_{||}$$

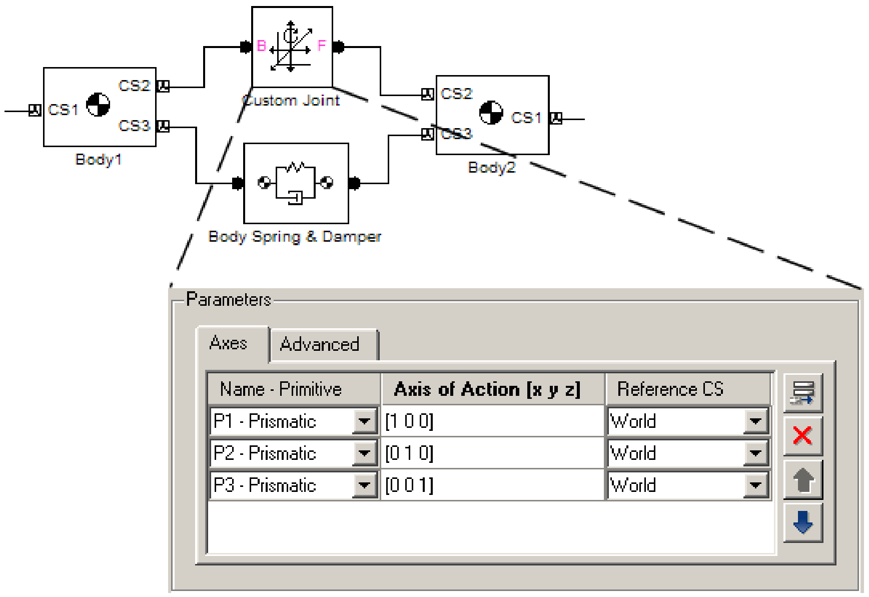
The block is connected on either side to Bodies at a Body coordinate system (CS). The displacement ***r*** is a vector from one Body CS on one Body to the other Body CS on the other Body. Newton’s third law requires that the forces that the bodies exert on one another be equal and opposite.

The common physical system this force model represents is a spring-damper combination, where the damper is a dashpot acting only along the spring axis. The damping is solely a function of the component $v_{||}$ of the velocity vector projected along the displacement ***r***. (Thus the damping in this block cannot represent the damping due to a viscous medium, because there is no damping force perpendicular to the spring axis. See “Inserting a Linear Force or Torque Through a Joint” on page 4-71.)

You enter the constant parameters r_0, k, and b in the Body Spring & Damper dialog. r_0 is the spring's natural length, the length it has when no forces are acting on it. The spring constant k and damping constant b should be nonnegative.

To complete a linear force model between bodies, you need to model the translational degrees of freedom (DoFs) between them, as the Force Element block itself does not represent these DoFs. You can use any Joint block containing at least one prismatic primitive to represent translational motion. The two Bodies, the Joint, and the Body Spring & Damper must form a closed loop.

The following block diagram represents two Bodies with a damped spring between them. The Custom Joint represents the bodies' relative translational DoFs with three prismatic primitives. In this case, CS2 and CS3 on Body1 are the same, and CS2 and CS3 on Body2 are the same. Thus, the Joint is connected to the same Body CSs that define the ends of the spring-damper axis.



Name - Primitive	Axis of Action [x y z]	Reference CS
P1 - Prismatic	[1 0 0]	World
P2 - Prismatic	[0 1 0]	World
P3 - Prismatic	[0 0 1]	World

Inserting a Linear Force or Torque Through a Joint

Another way of inserting a linear force element between two bodies is to connect it to a joint that already connects the bodies. You have to apply the force element, like an actuator, to each primitive in the joint individually. This approach has several advantages over the Body Spring & Damper:

- You can create a different force law, with a different spring length, spring constant, and damping constant, for each of the joint's primitives.
- The spring and damper forces acting on each primitive act independently in their respective directions, instead of depending on just the interbody distance with a single spring length, spring constant, and damping constant.

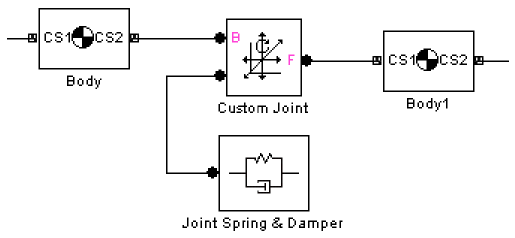
  This allows you to create spring and damping forces that act independently in two or three dimensions, unlike the Body Spring & Damper force, which acts only along a single axis. Damping forces acting on multiple primitives act as a two- and three-dimensional viscous medium, not as a dashpot.
- The joint representing the DoFs between the bodies is already present.

You use the Joint Spring & Damper block to implement such spring-damper forces/torques together with a Joint. With it, you can apply a linear spring and damper force to each prismatic primitive and a linear torsion and damper torque to each revolute primitive in a Joint block. (You cannot apply these torques to a spherical primitive.)

Pick a Joint already connected between two Bodies. You connect the Joint Spring & Damper block to a Joint block at a sensor/actuator port on the Joint. (The section "Actuating a Joint" on page 4-52 explains how to create such a port.) The Joint Spring & Damper dialog then lists each primitive in the Joint.

For each prismatic primitive you want to actuate with a spring-damper force, you specify a natural spring length (offset), spring constant, and damping constant. For each revolute primitive you want to actuate with a torsion-damper torque, you specify a natural torsion angle (offset, or angle in which the primitive points absent any torques), torsion constant, and damping constant. You make these specifications in the Joint Spring & Damper dialog.

Here are two bodies connected by a Custom Joint in turn connected to a Joint Spring & Damper block.



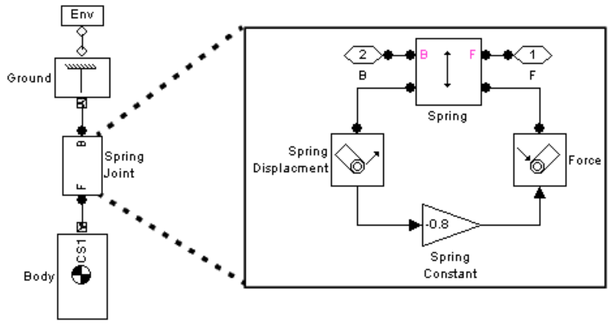
Unlike the example in the preceding section, "Inserting a Linear Force Between Bodies" on page 4-69, the Custom Joint can have up to three prismatics and three revolutes, each with a separate linear force or torque acting through it. Each force or torque acts equally and oppositely on each body, following Newton's third law.

Customizing Force Elements with Sensor-Actuator Feedback

You can create your own force elements acting through Joints or on Bodies by using Sensor-Actuator feedback loops. With this technique, you can not only model linear forces, but any force that depends on body or joint positions and velocities.

This simple example illustrates the method with a linear spring force law. Hooke's law states that the force exerted by an extended spring is proportional to its displacement from its unextended position: $F = -kx$.

The following SimMechanics model represents a spring that obeys Hooke's law.



The model uses the Gain block labeled Spring Constant to multiply the displacement of the prismatic joint labeled Spring along the World's *y*-axis by the spring constant `-0.8`. The output of the Gain block is the force exerted by the spring. The model feeds the force back into the prismatic joint via the Actuator labeled Force. The model encapsulates the spring block diagram in a subsystem to clarify the model and to allow a spring to be inserted elsewhere.

Checking Model Validity

Simulink can simulate a SimMechanics model only if it is valid. A model is valid if it satisfies the following rules:

- Each machine in the model contains at least one Ground, and exactly one Ground in each machine is connected to a Machine Environment block. Each submachine connected to a full machine by a Shared Environment block must have at least one Ground.

  See “Modeling Machines” on page 4-3.
- Every machine in the model is topologically valid. See “Verifying Machine Topology” on page 4-74.
- The model contains at least one degree of freedom. See “Counting Degrees of Freedom” on page 4-77.

Verifying Machine Topology

To avoid simulation failures, you must ensure that the topology of your block diagram is valid. A block diagram is topologically valid if each machine that it contains is valid. A machine is valid if its *spanning tree* is valid. Thus to determine if your model is valid, first determine the spanning tree of each machine that it contains and then the validity of each resulting tree.

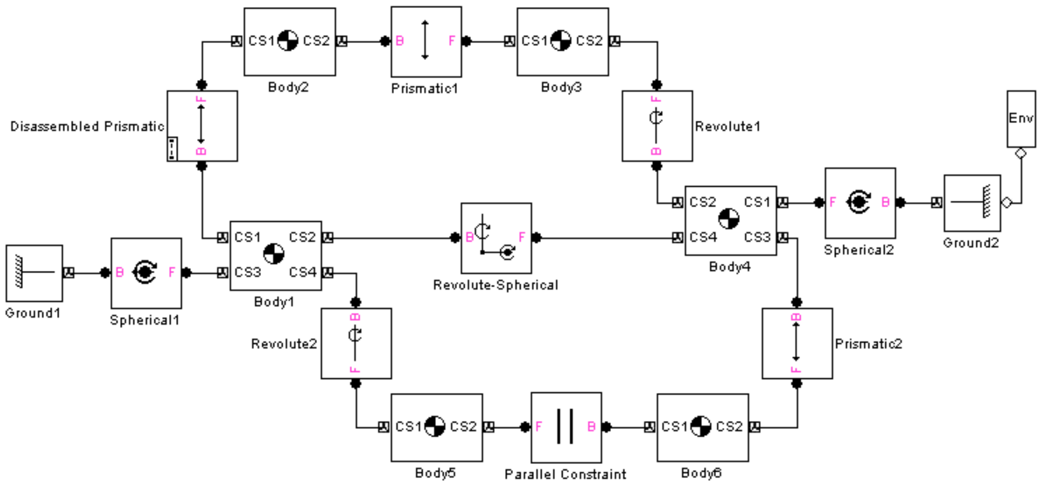
Machine Topology and Subsystems

When examining your model’s topology, be sure to inspect all its subsystems, including masked subsystems, down to the bottom of the model’s subsystem hierarchy.

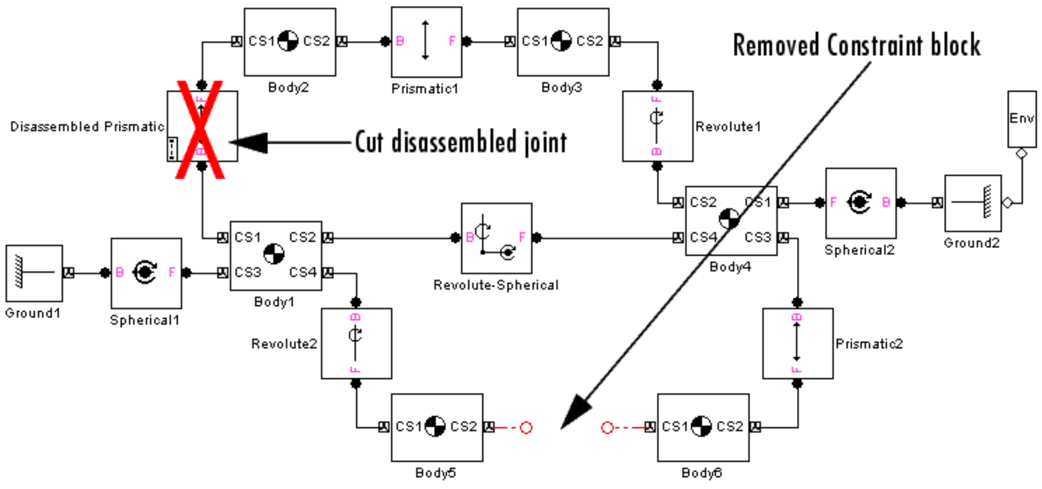
Determining a Machine’s Spanning Tree

To determine the spanning tree of a machine, remove all blocks from the machine except Body and Joint blocks and open every closed loop in the resulting reduced machine. To open a closed loop, follow the loop-cutting rules in “Cutting Closed Loops” on page 4-36.

For example, here is a machine with two closed loops.



Cutting the top loop at the Disassembled Prismatic and removing the Parallel Constraint block (thus simultaneously cutting the bottom loop) yields the machine's spanning tree, as shown here.



Determining the Validity of a Spanning Tree

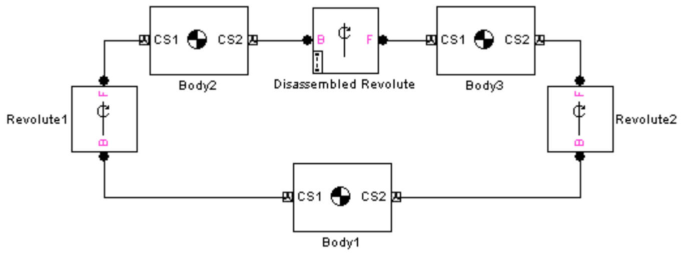
To be valid, a spanning tree must meet these requirements:

- The spanning tree must have at least one Ground block to serve as a reference to World.
- Every Joint block must be connected to exactly two Body blocks.
- Every non-Ground Body block must have a unique path to a Ground block. (This need not be true of the machine itself.) This ensures that, while each body moves via joints relative to other bodies, SimMechanics can resolve all bodies' motions into absolute motions with respect to World.
- Every non-Ground Body block at an end of a sequence of Bodies must have nonzero inertia (mass or inertial moment) associated with all joint primitives that can move. Each translational DoF must carry a nonzero mass, and each rotational DoF a nonzero inertial moment. This prevents infinite accelerations when forces and torques are applied.

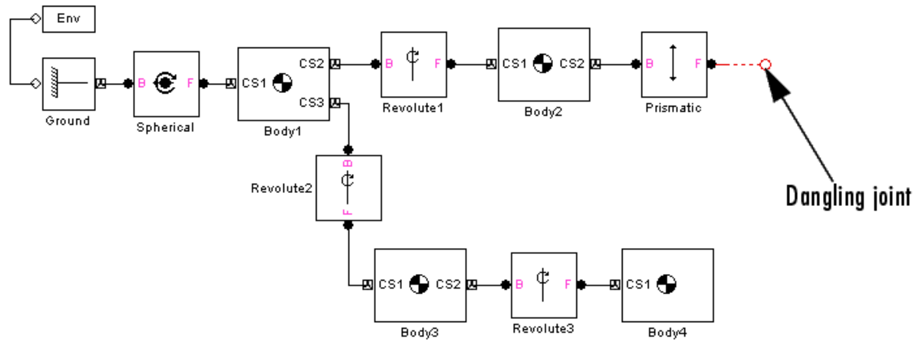
Examples of Invalid Machine Topologies

Here are some examples of invalid topologies:

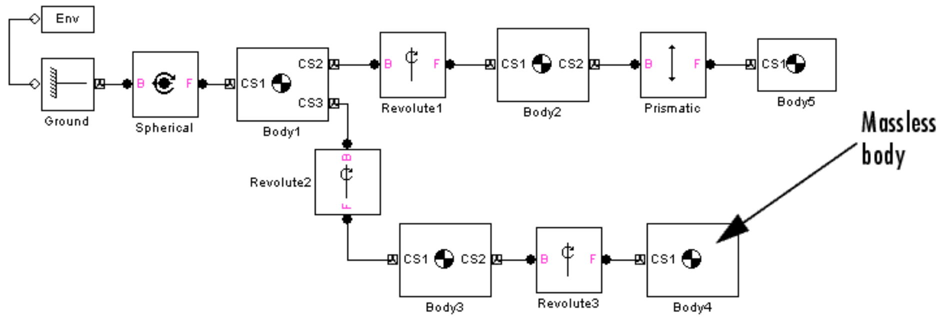
- This one-loop machine lacks a Ground block.



- This open machine has a dangling Joint block.



- Another open machine features a zero-mass body at one end of a chain of bodies.



The last two invalid examples are dynamically (but not topologically) equivalent, because a zero-mass body is dynamically no body at all.

Counting Degrees of Freedom

Identifying and counting the independent degrees of freedom (DoFs) of a machine are important for trimming and linearizing SimMechanics models (see “Trimming Mechanical Models” on page 8-18 and “Linearizing Mechanical Models” on page 8-32) and for correcting simulation errors (see “Troubleshooting Simulation Errors” on page 5-17).

Your SimMechanics model must have at least one DoF to be valid. A free physical body has six DoFs: three translational and three rotational. But in a machine, connections between bodies by joints, constraints, and drivers, and motion actuation by joint and body actuators reduce the machine's independent DoFs to a smaller number. You also reduce a body's DoFs if you confine the machine's motion to one or two spatial dimensions.

In SimMechanics, a Body block has no DoFs. Connecting Joints to a Body adds DoFs to the machine. The joint primitives represent the Body's DoFs relative to other connected Bodies or Grounds. Connecting Constraint and Driver blocks to Bodies or motion-actuating joint primitives in Joints removes DoFs from the machine. A locked Joint Stiction Actuator also removes a DoF.

Degrees of Freedom in Subsystems

When you examine your model to identify and count its DoFs, be sure to open and inspect all its subsystems, including masked subsystems, to the bottom of the model's subsystem hierarchy.

Finding Independent Degrees of Freedom

Here is the formula for determining the *number of independent DoFs* your model has:

# of independent DoFs = # of body DoFs + # of primitive DoFs -
# of motion restrictions

The following three steps define each term on the right side:

1. Calculate the *number of body DoFs* from the number of Body and Joint blocks in your model:

   # of body DoFs = 6 * (number of Bodies - number of Joints)

   If you have confined the machine to move in only two dimensions, replace the 6 by 3. If you have confined the machine to move in only one dimension, replace the 6 by 1.

2. Calculate the *number of primitive DoFs* by adding up the primitive DoFs from the Joint dialog boxes:

- Count one for each prismatic (P) or revolute (R) primitive.
- Count three for each spherical (S) primitive.
- Count zero for each weld (W) primitive.

Do not count a primitive DoF that is motion-actuated by a Joint Actuator.

3 Calculate the *number of motion restrictions* by adding up the motion restrictions of each Constraint and Driver block and from each locked Joint Stiction Actuator. Different blocks from the Constraints & Drivers library impose different numbers of motion restrictions. Stiction actuators apply to individual joint primitives.

Constraint Block	Restrictions	Driver Block	Restrictions
Gear	One	Angle	One
Parallel	Two	Distance	One
Point-Curve	Two	Linear	One
		Velocity	One

Be sure not to count redundant motion restrictions. These are restrictions that forbid the motion of joint primitives that could not move anyway even if the constraint were removed, because of how the joints are configured.

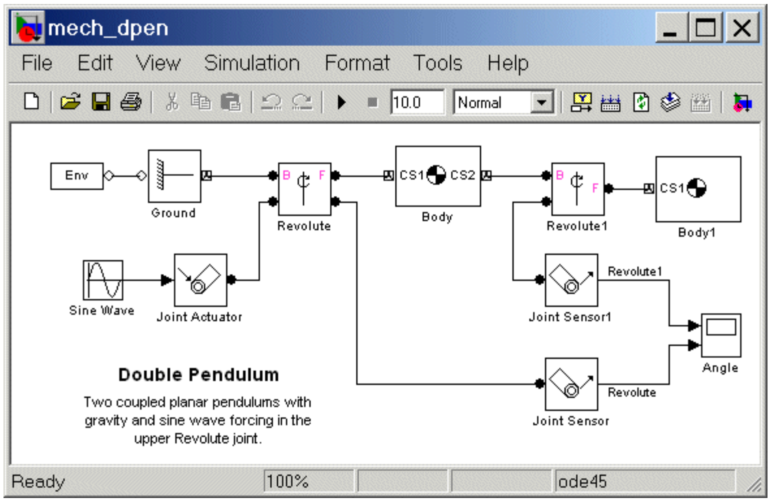
Example: A body is connected to a ground by a single prismatic. You place a constraint on the body that prevents it from moving perpendicularly to the prismatic axis. The body could not move in that direction even if you removed the constraint. So the constraint is redundant, and you would not count it as a motion restriction.

The Role of Joint Stiction Actuators

A Joint Stiction Actuator can remove or restore a DoF during a simulation. It is the only block that can change the number of independent DoFs after you start simulating. You must count an additional motion restriction during the period when a stiction-actuated primitive is locked. The primitive counts as another DoF if it is unlocked.

DoF Example: Double Pendulum

The mech_dpen model from the Demos library represents planar double pendulum motion actuated by a Joint Actuator.



Double Pendulum

Two coupled planar pendulums with gravity and sine wave forcing in the upper Revolute joint.

The double pendulum has two rigid bodies, such as two rods, confined to move in two dimensions. Ignoring the Joint Actuator temporarily, there are two bodies, two joints, and two revolute primitives, and thus 3 * (2 - 2) + 2 = 2 independent DoFs. There are many ways to represent these two DoFs, but the two revolute primitives are the simplest way.

Including the Joint Actuator in the DoF count removes the revolute primitive in the Revolute block as an independent DoF. So this model actually only has one independent DoF, the revolute primitive in the Revolute1 block.

DoF Example: Four Bar Mechanism

The example in “Four Bar Mechanism” on page 2-36 has four revolutes. You can establish that only 3 * (3 - 4) + 4 = 1 of these DoFs is actually independent and arrive at the same result obtained in the example.

Running Mechanical Models

SimMechanics gives you multiple ways to simulate and analyze machine motion in the Simulink environment. Running a mechanical simulation is similar to running a simulation of any other type of Simulink model. It entails setting various simulation options, starting the simulation, and dealing with simulation errors. See the Simulink documentation for a general discussion of these topics. This chapter focuses on aspects of simulation specific to SimMechanics models.

Running SimMechanics Models in Simulink

Simulink provides an extensive set of simulation options that apply to any type of model. SimMechanics provides additional options that apply specifically to simulating mechanical models. This chapter discusses those standard Simulink options for which mechanical models entail special consideration and the additional mechanical system-specific options of SimMechanics.

Distinguishing Models and Machines

Following the distinction introduced in "Modeling Machines" on page 4-3, SimMechanics requires you to make two categories of settings, one for each machine in a model and one for the entire model.

- "Machine Settings via the Machine Environment Block" on page 5-2
- "Model-Wide Settings via Simulink" on page 5-2

The first uses the Machine Environment block dialog, the second the Simulink **Configuration Parameters** dialog. To configure a mechanical model for simulation, you need to interact with both dialogs.

Machine Settings via the Machine Environment Block

Mechanical settings for a specific machine are located in that machine's connected Machine Environment block. This block controls the machine's mechanical environment, including simulation dynamics, machine dimensionality, gravity, tolerances, constraints, motion analysis modes, and visualization. See "Configuring a Machine's Mechanical Environment" on page 5-3 for details.

Model-Wide Settings via Simulink

Mechanical and general settings for an entire model are located in the Simulink **Configuration Parameters** dialog. Every node in this dialog is relevant to controlling your model's simulation, including visualization. At a minimum, you need to examine the settings in the **Solver** and **SimMechanics** nodes. See "Controlling the Simulation" on page 5-11 for details.

Configuring a Machine's Mechanical Environment

Each machine in your model has a connected Machine Environment block. This block controls the mechanical environment for that machine. Unless otherwise noted, this section refers exclusively to this block's dialog.

- "The Machine Environment Block" on page 5-3

The Machine Environment settings include the following:

- "Setting Gravity" on page 5-3
- "Choosing Your Machine's Dimensionality" on page 5-4
- "Setting Assembly Tolerances" on page 5-5
- "Implementing Constraints" on page 5-6
- "Analyzing the Motion" on page 5-7
- "Handling Motion Singularities" on page 5-9

The Machine Environment Block

Every machine in your model requires exactly one Machine Environment block to be connected to one of its Ground blocks. The settings that you enter in that Machine Environment block determine the mechanical environment for that machine only. Other machines are controlled by their respective Machine Environment blocks.

This section guides you through the major choices you implement through this dialog's four panes. See the Machine Environment reference page for a complete description.

Setting Gravity

A uniform gravity field is applied to the motion of every machine. The default is a constant vector of `[0 -9.81 0]` with units of meters/seconds^2 and x-, y-, and z-components, respectively.

You can change this value to a different constant vector by modifying the entry in the **Gravity vector** field of the **Parameters** pane of the Machine

Environment dialog. You can change the units by using the units pull-down menu.

Gravity as an External Simulink Signal

In addition to constant gravity, SimMechanics lets you apply a time-varying, albeit spatially uniform, gravity vector through a Simulink signal. You enable this option by selecting the **Input gravity as signal** check box in the **Parameters** pane.

Once you make this selection, the Machine Environment block acquires a Simulink inport to accept this Simulink signal. The signal must be a three-component vector. You can still change the units through the pull-down menu.

Choosing Your Machine's Dimensionality

In general, you simulate machine motion in all three spatial dimensions. If a machine can move in only two dimensions, however, ignoring the third dimension makes the simulation more efficient. By default, SimMechanics automatically determines whether your machine moves in all three or only two dimensions and optimizes the simulation accordingly.

You can override this default by requiring SimMechanics to simulate in either three or two dimensions. You choose the simulation dimension of a machine in the **Machine dimensionality** pull-down menu of the **Parameters** pane of the Machine Environment dialog. If you attempt to simulate a three-dimensional machine in two dimensions, the simulation stops with an error.

Determining the Dimensionality of Your Machine Manually

Your machine must meet certain criteria before SimMechanics can simulate it in two dimensions:

- The prismatic primitives must define a set of parallel planes.
- The revolute primitives must rotate about axes perpendicular to the prismatic planes.

The bodies of a two-dimensional machine do not all have to lie in a single plane, but they should only slide and rotate in parallel planes.

Blocks That Require Three-Dimensional Simulation

The SimMechanics library contains certain blocks that, if you use them in a machine, require you to simulate in three dimensions.

- Any Joint block with more than two prismatic primitives, more than one revolute primitive, or any spherical primitives
- Disassembled Joints
- Massless connectors

Code Generated from Two-Dimensional Models

Code generated from simulations restricted to two-dimensional motion is also restricted to two-dimensional motion. See "Restrictions on Two-Dimensional Simulation" on page 5-34.

Setting Assembly Tolerances

The linear and angular assembly tolerance specify the precision with which

- A model must specify the initial locations and angles of a machine's joints.
- SimMechanics must solve the initial positions and angles of a machine's unassembled joints.

The **Parameters** pane of Machine Environment allows you to change the default assembly tolerances in the **Linear assembly tolerance** and **Angular assembly tolerance** fields. You can also adjust the linear and angular units in the respective pull-down menus.

How SimMechanics Implements Assembly Tolerances

SimMechanics checks the locations and angles of a machine's assembled joints when it initializes the model and later during the simulation. If any of the joint locations or angles fails to meet the corresponding assembly tolerances, Simulink halts the simulation and displays an error message. If this happens, you should check your machine to ensure that it specifies the locations and angles of its assembled joints to the precision specified in the **Parameters** pane. If not, either change the locations and angles that fail to meet the assembly tolerances or increase the tolerances themselves.

Implementing Constraints

If your machine contains implicit or explicit constraints on a system's motion, SimMechanics uses one of the three possible constraint solvers to find a solution for the motion that meets the constraints:

- Stabilizing solver
- Tolerancing solver
- Machine precision solver

This section describes the constraint solvers. These constraint choices are found on the **Constraints** pane of the Machine Environment dialog. If you choose the tolerancing solver, you must also specify the constraint tolerances.

Stabilizing Constraint Solver

This solver adds a self-correcting term to the equations of motion that stabilizes the numerical solution, i.e., causes it to evolve toward, rather than drift away from, the actual solution. SimMechanics uses this solver by default. It is typically faster than the other solvers, but can settle into a solution that exceeds the machine's assembly tolerances. If assembly tolerance errors occur during the simulation, use one of the other solvers instead.

Tolerancing Constraint Solver

This solver finds the system's motion while imposing the constraints to the tolerance that you specify. Specifically, the solver stops refining the solution when the difference between two successive solutions satisfies the condition

$$|error| < \max(|rtol * x|, atol)$$

where *error* is the difference between successive solutions, *rtol* is the relative constraint tolerance, *x* is the motion to be solved, and *atol* is the absolute constraint tolerance. See "Setting Constraint Tolerances" on page 5-7 following.

This solver is recommended if you plan to run the simulation in Kinematics mode. It is more accurate than the stabilizing solver, but less accurate than the machine precision solver, with a computational efficiency in between the two.

Setting Constraint Tolerances

If you use the tolerancing solver, the constraint tolerances that SimMechanics uses are under your control. You can view and change the constraint tolerances in the **Relative tolerance** and **Absolute tolerance** fields of the **Constraints** pane.

Machine-Precision Constraint Solver

Solves the constraints to the numerical precision of the computer on which the simulation is running. Select this solver if you want to obtain the most accurate simulation permitted by the computer, regardless of simulation time.

Analyzing the Motion

You can use SimMechanics to compute

- The motion that results from applying forces to a mechanical system (forward dynamics)
- The forces required to produce a specified motion in a mechanical system (inverse dynamics)
- The steady-state motion of a mechanical system (trimming)
- The effect of slightly perturbing a mechanical system's motion (linearization)

To compute any of these results, you must build an appropriate model and choose an appropriate mode of motion analysis.

Choosing an Analysis Mode

The **Parameters** pane of the Machine Environment dialog allows you to choose the analysis mode you want to simulate in. You make this choice via the **Analysis mode** pull-down menu.

Forward Dynamics Mode

This mode computes the positions and velocities of a system's bodies at each time step, given the initial positions and velocities of its bodies and any forces applied to the system. Use this mode to simulate a model that represents the initial positions and velocities of the system's bodies and the forces on those bodies.

Run these examples in the Forward Dynamics mode:

- "Running a Demo Model" on page 1-5
- "Building a Simple Pendulum" on page 2-11 and "Four Bar Mechanism" on page 2-36

as well as the many examples of Chapter 4, "Modeling Mechanical Systems".

Inverse Dynamics Mode

This mode computes the forces required to produce a specified velocity for each body of an open-loop system. Use this mode to simulate an open-loop system whose model specifies the velocity of every degree of freedom of every body at every time step.

See "Inverse Dynamics Mode with a Double Pendulum" on page 8-8 for an example of using this mode to find the forces on an open-loop system.

Kinematics Mode

Computes the forces required to produce a specified velocity for each body of a closed-loop system. Use this mode to simulate a closed-loop system whose model specifies the velocity of every independent degree of freedom at every time step. The tolerancing constraint solver is recommended in this mode. (See "Implementing Constraints" on page 5-6.)

See "Inverse Dynamics Mode with a Double Pendulum" on page 8-8 for an example of using this mode to find the forces on a closed-loop system.

Trimming Mode

This is a variant of Forward Dynamics mode that allows you to run the Simulink `trim` command on your model. The `trim` command in turn allows you to find steady-state solutions for your model.

Trimming mode inserts a subsystem and an output port at the top level of your model. These blocks output signals corresponding to the constraints on the system represented by your model. Configure the `trim` command to find equilibrium points where the constraint signals are zero. This ensures that the equilibrium points found by the `trim` command satisfy the constraints on the modeled system.

See "Trimming Mechanical Models" on page 8-18 for examples of using this mode to find the equilibrium points of a mechanical system.

Special Settings If You Linearize a Machine

You can determine the effect of small perturbations on system motion by linearizing your machine. To linearize, set the analysis mode to Forward Dynamics and run the Simulink `linmod` command on your model.

You can fix the size of the perturbation or let SimMechanics find an optimal perturbation for you. Enter these settings in the **Linearization** pane.

See "Linearizing Mechanical Models" on page 8-32 for examples of using this mode to find the effect of small perturbations on mechanical motion.

Handling Motion Singularities

At certain simulation times, one or more degrees of freedom in a mechanical system might change relatively quickly compared to the others. If these sudden, quick motions are too fast compared to the slower motions, Simulink and SimMechanics have difficulty finding an accurate solution in a reasonable simulation time. Imposing constraints on the motion often exacerbates this problem. In extreme cases, the simulation can stop with an error.

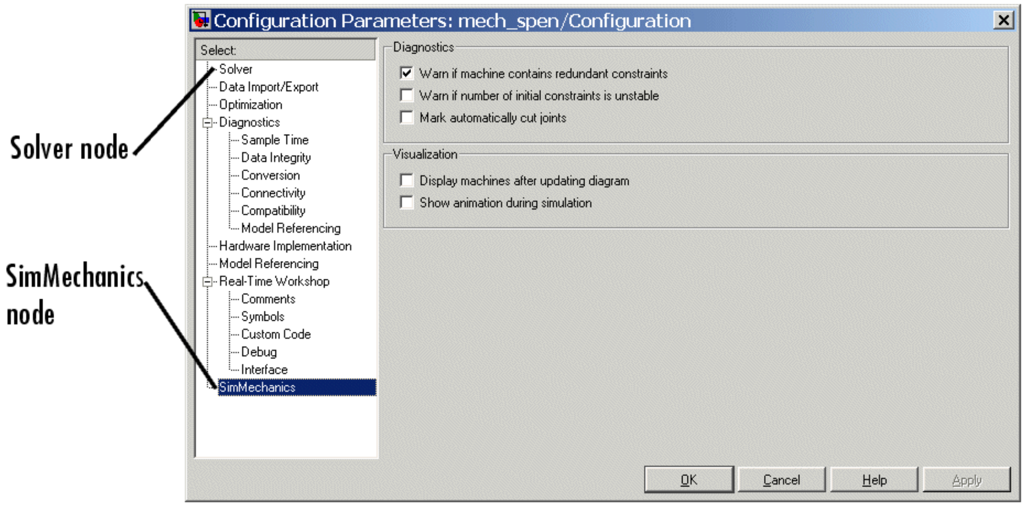
You can alleviate these motion singularities by selecting the **Use robust singularity handling** on the **Constraints** pane of the Machine Environment dialog. This option requires extra computation whether or not singularities exist. Select it only if you cannot find a Simulink solver that solves your model in a reasonable amount of time without it.

See "Adjusting Constraint Tolerances" on page 5-25 and "Changing the Simulink Solver and Tolerances" on page 5-26 for more discussion of motion singularities and their relationship to the Simulink solvers.

Controlling the Simulation

The simulation controls for an entire model are located in the **Configuration Parameters** dialog, accessed through the Simulink **Simulation** menu. See Simulink documentation for complete details about this dialog. This section, unless otherwise noted, refers exclusively to this dialog.

You must check and possibly adjust two nodes of this window, **Solver** and **SimMechanics**, before running a mechanical model.



Diagnostics

- [✓] Warn if machine contains redundant constraints
- [ ] Warn if number of initial constraints is unstable
- [ ] Mark automatically cut joints

Visualization

- [ ] Display machines after updating diagram
- [ ] Show animation during simulation

Simulink Configuration Parameters Dialog (SimMechanics Node Shown)

The controls specific to SimMechanics are located on the **SimMechanics** node, which has two active areas, **Diagnostics** and **Visualization**.

- “Configuring SimMechanics Simulation Diagnostics” on page 5-12
- “Visualizing Your Machines” on page 5-13

The choice and configuration of the solver are Simulink settings, located on the **Solver** node. This node has two active areas, **Simulation time** and **Solver options**.

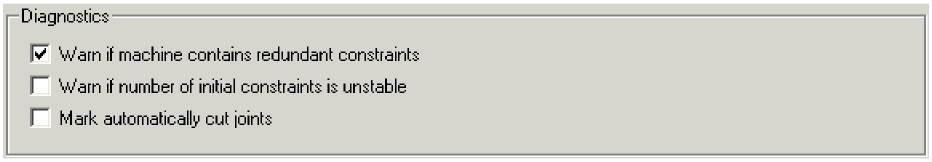
- “Choosing a Simulink Solver” on page 5-13

Once SimMechanics and Simulink are configured, you can run your model.

- "Starting the Simulation" on page 5-14

Configuring SimMechanics Simulation Diagnostics

SimMechanics can provide certain diagnostics to help you understand and, if necessary, troubleshoot simulation problems and errors. You can adjust these diagnostics in the **Diagnostics** area of the **SimMechanics** node of **Configuration Parameters**.


Diagnostics

[✓] Warn if machine contains redundant constraints

[ ] Warn if number of initial constraints is unstable

[ ] Mark automatically cut joints

See "How SimMechanics Works" on page 5-15 and "Troubleshooting Simulation Errors" on page 5-17 to learn about correcting SimMechanics simulation errors.

Warning on Redundant Constraints

Selecting the **Warn if machine contains redundant constraints** check box causes SimMechanics to warn you if there are more constraints than necessary in your model. This situation by itself does not cause simulation errors. But too many constraints might lead, during simulation and in certain configurations, to conflicting constraints and errors.

The check box is selected by default.

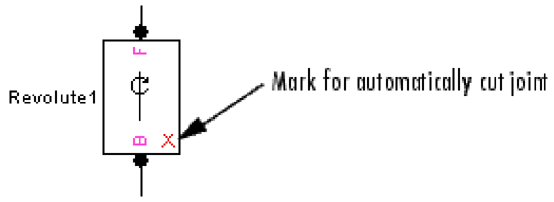
Warning on Unstable Constraints in Initial State

Selecting the **Warn if number of initial constraints is unstable** check box causes SimMechanics to warn you if small changes to your model's initial state leads to changes in the number of constraints. During simulation and in certain configurations, this instability could lead to too few or too many (conflicting) constraints on your system and prevent SimMechanics from finding a solution for the motion.

The check box is not selected by default.

Marking Automatically Cut Joints

Selecting the **Mark automatically cut joints** check box causes Simulink to mark the icons of Joint or Constraint/Driver blocks that it cuts during simulation of the model. See “Cutting Closed Loops” on page 4-36 for an additional discussion. The check box is not selected by default.



Visualizing Your Machines

Configuring visualization requires three steps. See “Starting SimMechanics Visualization” on page 6-2 for complete details about SimMechanics visualization.

- You enter the visualization settings for an entire model in the **Visualization** area of the **SimMechanics** node of the **Configuration Parameters** dialog. To open visualization, you must select at least one of these check boxes.

  Model-wide visualization is turned off by default.

- You can choose whether or not to visualize a specific machine in your model through the **Visualization** pane of its Machine Environment dialog. A single window displays all selected machines in a model.
- All other visualization controls are located on the SimMechanics visualization window itself.

Choosing a Simulink Solver

SimMechanics uses one of the ordinary differential equation (ODE) solvers of Simulink to solve a system’s equations of motion, typically in tandem with a constraint solver (see “Implementing Constraints” on page 5-6).

Simulink provides an extensive suite of ODE solvers that represent the most advanced numerical techniques available for solving differential equations in general and equations of motion in particular. The **Solver** node of the **Configuration Parameters** dialog allows you to select any of these solvers for use by Simulink in solving the model's dynamics. See the Simulink documentation for complete details about choosing a Simulink solver.

Setting Simulink Solver Tolerances

By default Simulink automatically determines the absolute tolerance used by ODE solvers. The resulting tolerance might not be small enough for a mechanical system, particularly a nonlinear or chaotic system. Try running a simulation with the relative tolerance set to `1e-3` (the default) and the absolute tolerance set to `1e-4`. Then increase the tolerances if the simulation takes too long or decrease them if the solution is not sufficiently accurate.

Solver Tolerances and Stiction

If your model contains one or more Joint Stiction Actuator blocks, you must also take into account the velocity thresholds of these blocks when setting the absolute tolerance of the ODE solver. If the absolute tolerance of the solver is greater than a joint's velocity threshold, the simulation might never detect the locking or unlocking of a joint. To prevent this from happening, set the absolute tolerance to be no more than 10% of the size of the smallest stiction velocity threshold in your model.

Starting the Simulation

Once Simulink and SimMechanics are configured to simulate a mechanical system, you can run your model.

As the simulation proceeds, you might encounter warnings, errors, and unexpected or unsatisfactory results. Consult these sections to learn how to identify errors and improve your simulation.

- "How SimMechanics Works" on page 5-15
- "Troubleshooting Simulation Errors" on page 5-17
- "Improving Performance" on page 5-23
- "Generating Code" on page 5-28
- "Limitations" on page 5-33

How SimMechanics Works

You might find this brief overview of how SimMechanics works helpful for constructing models and understanding errors. Fixing errors is discussed in "Troubleshooting Simulation Errors" on page 5-17.

There are four major phases of the machine simulation sequence. The first two occur before SimMechanics actually starts machine motion.

1. "Model Validation" on page 5-15
2. "Machine Initialization" on page 5-15
3. "Force Analysis and Motion Integration" on page 5-16
4. "Stiction Mode Iteration" on page 5-16

The premotion machine configurations (home, initial, and assembled) are discussed in "Kinematics and the Machine's State of Motion" on page 3-2 and in their respective "Glossary" on page Glossary-1 entries.

Model Validation

SimMechanics first checks your data entries from the dialogs and the local connections among neighboring blocks. It then validates the Body coordinate systems; the joint, constraint, and driver geometries; and the model topology. Body positions and orientations defined purely by Body dialog entries constitute the *home configuration*.

Machine Initialization

SimMechanics next checks the assembly tolerances of Joints that you manually assembled.

SimMechanics then cuts each closed loop once. An invisible equivalent constraint replaces each cut Joint, Constraint, or Driver block. SimMechanics checks all constraints and drivers for mutual consistency and eliminates redundant constraints. It also checks whether a small perturbation to the initial state changes the number of constraints. Such a singularity might lead, during machine motion, to violation of the constraints.

Any Joint Initial Condition Actuators now impose initial positions and velocities, changing body geometries from their dialog box configurations as necessary and transforming the machines from their home configurations to their *initial configurations*. SimMechanics then finds an assembly solution for disassembled joints and initializes them in position and velocity, defining the *assembled configuration*. Assembly tolerances are checked again.

A “sticky” joint primitive, actuated by a Joint Stiction Actuator, can be in one of three stiction modes: locked, waiting, or unlocked. SimMechanics finds a mutually consistent set of stiction modes for all sticky joints.

Force Analysis and Motion Integration

In Forward Dynamics or Trimming analysis mode, SimMechanics begins the solution of machine motion by applying and integrating external forces and torques, stepping in simulation time. It maintains assembly, constraint, and solver tolerances and checks constraint and driver consistency. It also detects whether, within one Joint block, distinct joint primitive axes align and destroy one or more independent DoFs. Such an event is a joint axis singularity.

In Inverse Dynamics and Kinematics modes, SimMechanics now applies motion constraints, drivers, and actuators to find the machine motion and derive forces and torques. It also checks tolerances and consistency and detects singular alignment of joint primitives.

Stiction Mode Iteration

If stiction is present, SimMechanics checks at each time step whether the sticky joints transition from one stiction mode to another, then checks for mutual consistency of locked and unlocked sticky joint primitives across the whole machine.

Troubleshooting Simulation Errors

SimMechanics simulations can stop before completion with one or more error messages. You might find the previous section, "How SimMechanics Works" on page 5-15, useful for tracing errors. Some common errors also appear in "Modeling Machines" on page 4-3 and "Checking Model Validity" on page 4-74. This section discusses generic error types.

Most errors and error-fixing strategies fall into broad categories. These groupings are reflected in the keywords occurring in the error messages that SimMechanics displays. These sections summarize these groupings.

- "Data Validation Errors" on page 5-17
- "Ground and Body Geometry Errors" on page 5-17
- "Joint Geometry Errors" on page 5-18
- "Block Connection and Topology Errors" on page 5-19
- "Motion Inconsistency and Singularity Errors" on page 5-19
- "Analysis Mode Errors" on page 5-22

Data Validation Errors

Every numerical entry you make in SimMechanics must be a real numerical expression or MATLAB equivalent. Spatial vectors are 3-vectors, such as `[3 4 5]`. Spatial tensors are 3-by-3 matrices, such as rotation matrices and the inertia tensor.

Note You can specify a two-dimensional curve in the Point-Curve Constraint block with 2-vectors.

Ground and Body Geometry Errors

Every machine must have a least one Ground block. Every Body block must have at least one Body CS, defined at the body's center of gravity (CG). You must directly or indirectly define the Body coordinate systems (CSs) of a machine relative to a Ground or to World. You cannot enter cyclic (circular)

Body CS definitions. The Body CS definitions must separately satisfy these criteria in the **Position** and **Orientation** tabs of the Body dialog.

For example, defining CS3 relative to CS2, defining CS2 relative to CS1, then defining CS1 relative to CS3, results in a definition that is both cyclic and missing any reference to a Ground or World. You could break the cycle by referencing CS1 to a Ground or to World.

To be rendered in visualization, a Body must be connected to at least one Joint that is connected to the rest of the machine. You cannot visualize with equivalent ellipsoids a body whose principal inertial moments do not satisfy the *triangle inequalities*. (See "Rendering Body Shapes in SimMechanics" on page 6-5.)

Joint Geometry Errors

The geometric configuration of joints, constraints, and drivers can conflict with assembly requirements and restrictions on certain blocks.

Assembly Tolerances Violated

Assembled joints must satisfy assembly tolerances on their connected Body CSs at all times. Disassembled joints assembled at machine initialization must also satisfy assembly tolerances during the simulation.

Zero Massless Connector Distance

The initial distance between two Body CS origins connected by a massless connector must be nonzero. The massless connector holds the distance between two Body CS origins constant during motion.

Composite Joints: Restrictions Among Primitives

Certain composite Joint blocks place restrictions on their primitive joint axes. For example, Bearing must have its prismatic axis `P1` aligned to its third revolute axis `R3`.

Block Connection and Topology Errors

General rules on how to connect SimMechanics blocks are discussed in Chapter 4, “Modeling Mechanical Systems”. In particular, consult these sections of that chapter:

- “Modeling Machines” on page 4-3
- “Checking Model Validity” on page 4-74

Some restrictions are properties of individual blocks, as explained in their reference pages. See the SimMechanics block reference.

Motion Inconsistency and Singularity Errors

Inconsistencies in motion arise from misapplication of constraints, drivers, and actuators, from conflicting stiction requirements, and incorrect simulation dimensionality.

Motion simulation errors often occur because of *singularities* or dividing by very small numbers. SimMechanics can integrate certain singularities, at a cost (see “Choosing a Simulink Solver” on page 5-13).

Zero Masses and Moments of Inertia

A body moving on a prismatic axis must have nonzero mass if you actuate it with forces. A body rotating about a revolute axis or pivoting about a spherical must have nonzero inertial moments about the axis or pivot if you actuate it with torques. If you want a massless rigid body, consider using a Massless Connector from the Joints/Massless Connectors sublibrary.

Note You can use point bodies (nonzero mass but zero moments) in SimMechanics, if the connected revolute axes and spherical pivots are dislocated from the body. Although the moments are zero about a point body’s CG, the displacement of the body from the axis or pivot shifts the moments from zero to nonzero values.

Alignment of Distinct Primitives

Within a single Joint block, two distinct prismatic axes or two distinct revolute axes should never align during the simulation. If either occurs, a translational or rotational DoF is lost, and SimMechanics cannot determine the subsequent motion. An example of primitive axis alignment singularity is “gimbal lock.” Two of the three revolute primitive axes in the Gimbal block become parallel, reducing the number of independent DoFs in the Joint from three to two.

No Degrees of Freedom

Your machine cannot move if it has no degrees of freedom. Each Constraint, Driver, and motion-actuating Actuator block you add to a machine reduces the number of independent DoFs. (See “Counting Degrees of Freedom” on page 4-77.) Cure such errors by removing one or more of these blocks from your machine, until you have at least one independent DoF.

Incorrect Machine Dimensionality

You cannot run a three-dimensional machine with a simulation restricted to two dimensions. See “Choosing Your Machine’s Dimensionality” on page 5-4.

Redundant Constraints

Some constraints can restrict what another constraint is already restricting. Fix these errors by identifying and removing the redundancies.

Violated Constraints

Some machine motions or simulations might not be able to maintain assembly tolerances at a particular simulation step while simultaneously satisfying the constraints. One or more joints become disassembled, followed by an error.

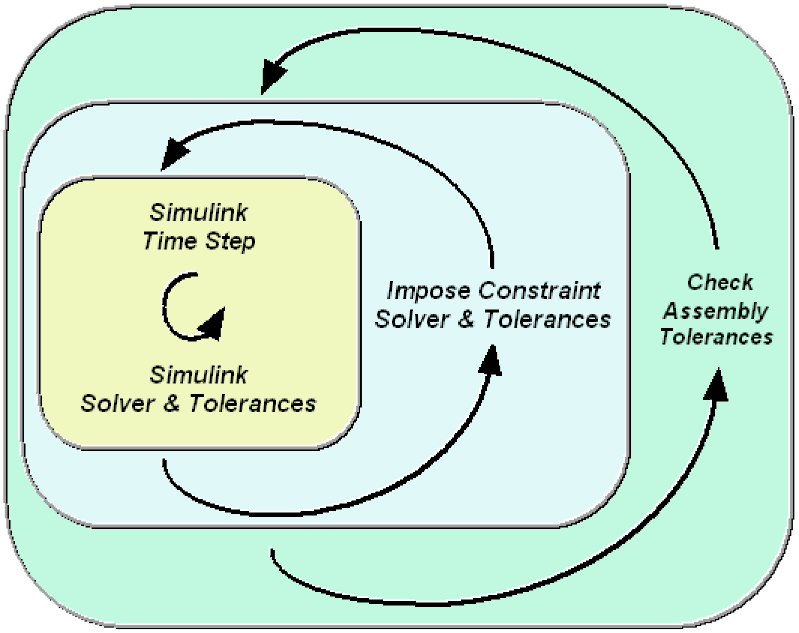
You can correct this situation in several ways. First, identify the joint, constraint, or driver causing the error and examine its physical configuration when the error occurs to isolate the conflict. Then try any combination of these steps:

- Decrease the Simulink solver tolerances.
- Switch to a more robust Simulink solver.
- Decrease the constraint solver tolerances.

- Switch to the machine precision constraint solver.
- Increase the assembly tolerances.

See "Choosing a Simulink Solver" on page 5-13 and "Implementing Constraints" on page 5-6.

SimMechanics tries to harmonize your choices of ODE solver and solver tolerances, constraint solver and tolerances, and assembly tolerances in this dynamic hierarchy:



Conflicting Actuators

You cannot put more than one actuator on a joint primitive.

Note You can simultaneously place an initial condition actuator and a force/torque actuator on a joint primitive.

The Joint Stiction Actuator block does accept an input signal for nonfrictional forces/torques, which the block adds to the stiction.

Sticky Joints in Conflict

If your machine has two or more stiction-actuated ("sticky") joints, a conflict among them can put SimMechanics into an infinite loop and prevent determination of the machine motion. Or one locked joint can prevent the other joints, sticky or not, from moving. The machine stops moving.

For example, one sticky joint becomes unlocked and requires the other to lock, which then requires the first to lock.

Remove these conflicts by removing one or more stiction actuators or by changing the Joint Stiction Actuator locking thresholds.

Analysis Mode Errors

Certain restrictions apply to the analysis modes presented in "Analyzing the Motion" on page 5-7. Consult individual analysis modes for more:

- "How SimMechanics Works" on page 5-15
- "Finding Forces from Motions" on page 8-7
- "Trimming Mechanical Models" on page 8-18
- "Linearizing Mechanical Models" on page 8-32

Improving Performance

SimMechanics is a general-purpose mechanical simulator. With it, you can model and simulate many types of machines with very different behaviors. In some cases, the settings you use for “well-behaved” machines are not optimal for more-difficult-to-simulate systems. Simulink and SimMechanics give you great freedom to change the mathematical and mechanical settings used in your simulations. Use this flexibility to avoid simulation errors and optimize performance, subject to the fundamental tradeoff between speed and accuracy.

- “Simplifying the Degrees of Freedom” on page 5-23
- “Adjusting Constraint Tolerances” on page 5-25
- “Smoothing Motion Singularities” on page 5-25
- “Changing the Simulink Solver and Tolerances” on page 5-26
- “Adjusting the Time Step in Real-Time Simulation” on page 5-27

Consult “Generating Code” on page 5-28 to learn about speeding up simulations by generating and compiling code from your models.

Simplifying the Degrees of Freedom

In general, the more degrees of freedom (DoFs) you add to your model, the slower the simulation.

Eliminating Unnecessary Degrees of Freedom

Under certain circumstances, a model can contain DoFs not practically necessary to predict system behavior. For example, a subsystem might contain very light masses whose motion is almost completely determined by the heavier masses in the system and that have almost no inverse influence on the larger system.

Consider freezing or eliminating such degrees of freedom from your model in order to speed up the simulation.

Freezing "Fast" and "Slow" Degrees of Freedom

A related distinction can be made between DoFs that change rapidly and those that change slowly. Such systems are "stiff" (literally, in the case of a stiff spring that oscillates at a very high frequency) and often hard to simulate accurately in a reasonable time.

One approach to improving the speed is to selectively freeze certain DoFs.

1 First, freeze or eliminate the "fast" DoFs and simulate only the "slow" DoFs.

2 Then freeze the "slow" DoFs in some representative configuration and simulate the motion of only the "fast" DoFs.

Such a split simulation between "fast" and "slow" DoFs can isolate important features of the system behavior, while ignoring unimportant features.

Caution Splitting DoFs between "fast" and "slow" sets and simulating the two sets separately neglects coupling between the two sets of DoFs. Only a full simulation can capture such coupling.

See "Solving Stiff Systems" on page 5-26 for a different approach to handling speed mismatches among DoFs.

Removing Stiction Actuators

Stiction requires computationally expensive algebraic loops. If possible, remove Joint Stiction Actuator blocks from your model to speed it up.

Simulating in Two Dimensions

If your machine moves in only two dimensions, not three, it qualifies for the SimMechanics two-dimensional simulation option. By reducing the linear and rotational directions from three to two and three to one, respectively, this option can noticeably improve simulation performance.

See "Choosing Your Machine's Dimensionality" on page 5-4.

Adjusting Constraint Tolerances

Maintaining constraints on a system's DoFs is a major and computationally expensive part of a simulation. If your simulation seems to run slowly or stops with constraint errors, especially when the mechanism passes through certain configurations, consider relaxing the constraint tolerances and/or solver. This step generally speeds up the simulation, although it also makes the simulation less accurate. Decreasing the tolerances increases the accuracy of the simulation but can increase the time required to simulate the model.

To view and change these settings in your machine, see "Implementing Constraints" on page 5-6.

Smoothing Motion Singularities

Singularities in a system's equations of motion can dramatically slow down a standard Simulink solver or even prevent it from finding a solution to a system's equations of motion. Because mechanical motion can become singular, SimMechanics provides the option of *robust singularity handling,* which works together with your selected solver to solve singular equations of motions efficiently. This feature allows Simulink in many cases to simulate models that otherwise cannot run or cannot be solved in a reasonable amount of time.

To enable robust singularity handling, see "Handling Motion Singularities" on page 5-9.

Avoiding Singular Initial Configurations

Avoid starting a machine in a singular configuration. Its subsequent motion violates assembly tolerances, as the simulation incorrectly removes one or more necessary constraints. A common singular configuration is one where the machine can move in two or three dimensions, but starts in exactly one or two dimensions, respectively.

Work around an initial singularity by slightly misaligning the joint axes, within assembly tolerances, before starting the simulation.

The **SimMechanics** node of the **Configuration Parameters** dialog allows you to enable simulation warnings for possible singular initial configurations. See "Configuring SimMechanics Simulation Diagnostics" on page 5-12.

Changing the Simulink Solver and Tolerances

The Dormand-Prince solver (`ode45`) that Simulink uses by default works well for many mechanical systems. But if your simulation seems to be slow and/or inaccurate you should consider changing the solver and/or adjusting the solver's relative and absolute tolerances. Chaotic and highly nonlinear systems especially require experimentation with different solvers and tolerances to obtain optimal results.

Consult the Simulink documentation for more about choosing Simulink solvers and tolerances.

Solving Stiff Systems

The default `ode45` Simulink solver typically requires too much time to solve systems that are stiff, that is, have bodies moving at widely differing speeds or have many discontinuities in their motions. An example of a stiff system is a pair of coupled oscillators in which one body is much lighter than the other and hence oscillates much more rapidly. Any of the following Simulink solvers might require significantly less time than `ode45` to solve a stiff system:

- `ode15s`: Variable-order solver based on a backward differentiation rule.
- `ode23t`: Trapezoidal rule solver. Use if your system is slightly stiff, to avoid numerical damping.
- `ode23tb`: Implicit Runge-Kutta method solver. More efficient than `ode15s` if the solution has many discontinuities.
- `ode23s`: Modified Rosenbrock method solver of order 2. This solver is also more efficient than `ode15s`, if the solution has many discontinuities.

Real-Time Simulation and Ignoring Motion Details with Fixed-Step Solvers

For most mechanical systems, variable time-step solvers are preferable. Fixed time-step solvers, depending on the size of the time step, often fail to resolve certain motion details.

Using a fixed-step solver can be advantageous in some cases, however:

- If you want to ignore unimportant motion details. Ignoring them can speed up your simulation, especially for a larger time step.
- If you are simulating in real time with generated code. Fixed-step solvers are typically, but not exclusively, the norm for real-time simulation.

For such cases, choose one of Simulink's fixed-step solvers and select the largest time step that produces reasonable simulation results.

Most of Simulink's fixed-step solvers are explicit. For stiff systems and larger time steps, an implicit solver such as the `ode14x` fixed-step solver can be superior to an explicit solver in speed and accuracy.

Adjusting the Time Step in Real-Time Simulation

A real-time simulation using code generated and compiled from your model must keep up with the actual mechanical motion. To this end, you must ensure that the solver time step is greater than the computation time needed by your compiled model.

To meet this condition, you might have to increase the time step or decrease the computation time. Increasing the time step often requires removing the model's "fast" DoFs. Decreasing the computation time requires simplifying your model. You can do this most easily by removing DoFs and/or constraints. See "Simplifying the Degrees of Freedom" on page 5-23.

Reference

[1] Moler, C. B., *Numerical Computing with MATLAB*, Philadelphia, Society for Industrial and Applied Mathematics, 2004, Chapter 7.

Generating Code

You can use SimMechanics with Real-Time Workshop to generate stand-alone C or C++ code from your mechanical models and enhance simulation speed and portability. Certain features of Simulink make use of generated or external code. This section explains these features.

- “Using Code-Related Products and Features” on page 5-29
- “How SimMechanics Code Generation Differs from Simulink” on page 5-30
- “Using Run-Time Parameters in Generated Code” on page 5-31

Some SimMechanics features are restricted when you translate a model into code. See “Limitations” on page 5-33.

Note Code generated from SimMechanics models is intended for rapid prototyping and hardware-in-the-loop applications. It is not intended for use as production code in embedded controller applications.

Related Simulink Code Generation Documentation

Consult the documentation for Real-Time Workshop, xPC Target, Simulink Accelerator, and “Writing S-Functions” for general information on installing and using these products.

Reasons for Generating Code

Code generation has many purposes and methods in Simulink. In SimMechanics, there are three essential rationales:

- Compiled code versions of Simulink models run faster than the original block diagram models. The time savings for regular Simulink models can be dramatic. For the SimMechanics portions of a model, the performance improvements are not as extreme, yet can still be significant.
- A more important consideration for SimMechanics models is that converting the SimMechanics part of a model to code frees your model from requiring SimMechanics to run. For example, converting a SimMechanics subsystem to an S-function block allows you to run a model with Simulink alone.

- An equally important consideration for SimMechanics is the stand-alone implementation of generated/compiled code. Once you convert part or all of your model to code, you can deploy the stand-alone executable program on virtually any platform, independently of MATLAB.

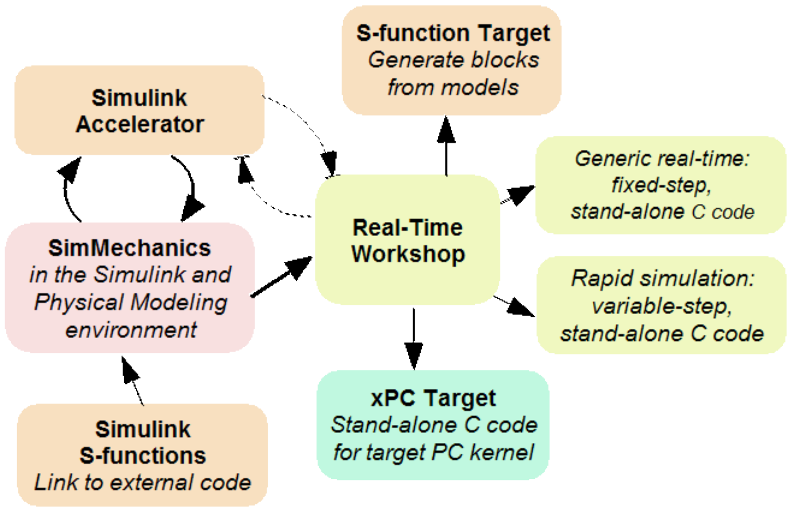
  Converting a model or subsystem to code also hides the original model or subsystem.

Using Code-Related Products and Features

Simulink, Real-Time Workshop, and xPC Target, using several code-related technologies, enable you to link existing code to your models and generate code versions of your models.

Code-Related Task	Component or Feature
Link existing code written in C or other supported languages to Simulink models	Simulink S-functions to generate customized blocks
Speed up Simulink simulations	Simulink Accelerator
Generate stand-alone fixed-step code from Simulink models	Real-Time Workshop
Generate stand-alone variable-step code from Simulink models	Real-Time Workshop Rapid Simulation Target (RSim)
Convert Simulink models to code and run them on a target PC	xPC Target
Generate blocks representing a Simulink models or subsystems	S-function Target
Generate code for designated models or subsystems	Model Reference

This diagram summarizes the product and feature flow of generating code from models and linking existing code to models.


S-function Target
Generate blocks from models
Simulink Accelerator
Generic real-time: fixed-step, stand-alone C code
Real-Time Workshop
SimMechanics in the Simulink and Physical Modeling environment
Rapid simulation: variable-step, stand-alone C code
xPC Target
Stand-alone C code for target PC kernel
Simulink S-functions
Link to external code

How SimMechanics Code Generation Differs from Simulink

In general, using the code generated from SimMechanics models is similar to using code generated from normal Simulink models. There are certain differences.

SimMechanics and Simulink Code Are Generated Separately

Real-Time Workshop generates code from the SimMechanics blocks separately from the Simulink blocks in your model. The generated SimMechanics code does not pass through `model.rtw` or the Target Language Compiler. All the code generated from a single model resides in the same directory, however.

SimMechanics Code Reuse Is Not Supported

Reusable subsystems in Simulink reuse code that is generated once from the subsystem. You cannot generate reusable code from subsystems containing SimMechanics blocks.

Using Run-Time Parameters in Generated Code

When SimMechanics generates code for a model, it creates a set of code source and header files. This set includes `*modelname*.c` and `*modelname*_data.c`, containing all the model's run-time parameters. (For C++, these are `.cpp` files.) In addition, SimMechanics generates two files that contain data structures and function prototypes for the SimMechanics blocks alone.

The `*modelname*.c` file contains all the run-time parameters used in the compiled simulation. `*modelname*_data.c` and the two special SimMechanics files are auxiliaries to aid in locating and changing the run-time data.

Changing Run-Time Parameters

As with code generated from any Simulink model without parameter inlining, you can change any run-time parameters by modifying their values in the block parameters data structure implemented in `*modelname*_data.c`. In this data structure, however, SimMechanics block parameters are not associated with their original blocks. Rather, SimMechanics block parameters are grouped together into a single vector associated with the first SimMechanics S-function for each machine in the model.

The data structures and functions found in the special SimMechanics files, `rt_mechanism_data.h` and `rt_mechanism_data.c`, allow you to modify SimMechanics block parameters in generated code. The special header file contains a data structure, `MachineParameters_*modelname_uniqueid*`, for each machine in the model, that includes a field for each block run-time parameter. To modify mechanical run-time parameters,

1. Use the function `rt_vector_to_machine_parameters_*modelname_uniqueid*` in the special code source file to create an instance of the machine parameters data structure from the vectorized parameters associated with the SimMechanics S-function.

2. Make the necessary modifications to the values in the data structure instance.

3. Use `rt_machine_parameters_to_vector_*modelname_uniqueid*` to reconstruct the vectorized parameters from the data structure instance.

4. Recompile your generated code.

Example: Changing a Block Parameter

This code listing is an example of a simple function that updates the mass of the first body in the demo mech_dpen. The argument p should be a pointer to the parameter vector associated with the SimMechanics S-function. The argument mass is the new mass for the first body. You should call this function before model initialization.

```
void update_mech_dpen_parameters(real_T *p, real_T mass)
{
    MachineParameters_mech_dpen_752c07b6 ds;
    /*
     * convert parameter vector into data structure
     */
    rt_vector_to_machine_parameters_mech_dpen_752c07b6(p, &ds);
    /*
     * change the mass of the first body in the double pendulum
     */
    ds.Body.Mass = mass;

    /*
     * convert the data structure back to the parameter vector
     */
    rt_machine_parameters_to_vector_mech_dpen_752c07b6(&ds, p);
}
```

Tunable Parameters Not Supported by SimMechanics Code Generation

A tunable parameter is a Simulink run-time parameter that you can change while the simulation is running. SimMechanics blocks do not support tunable parameters in either simulations or generated code.

SimMechanics Run-Time Parameter Inlining Ignores Global Exceptions

If you choose to enable parameter inlining for code generated from a SimMechanics model, SimMechanics inlines all its run-time parameters. If you choose to make some of the global SimMechanics block parameters exceptions to inlining, the exceptions are ignored. You can change global tunable parameters only by regenerating code from the model.

Limitations

Some Simulink features and tools either do not work with models containing SimMechanics blocks or work only with restrictions. Others work with SimMechanics models but only on the normal Simulink blocks in those models.

Changing Block Properties at the Command Line

Changing the block properties of SimMechanics blocks at the command line is not recommended.

Restricted Simulink Tools

Certain Simulink tools are restricted in use with SimMechanics.

- Simulink configurable subsystems work with SimMechanics blocks only if all of the block choices have consistent port signatures.
- For Iterator, Function-Call, Triggered, and While Iterator nonvirtual subsystems cannot contain SimMechanics blocks.
- SimMechanics supports external mode, but without visualization.
- SimMechanics supports Simulink model referencing, with these restrictions:
  - A SimMechanics model can be referenced only once by another model.
  - SimMechanics does not support reparameterization in a referencing block.

Unsupported Simulink Tool

The Simulink Profiler does not work with SimMechanics models.

Simulink Tools Not Compatible with SimMechanics Blocks

Some Simulink tools and features do not work with SimMechanics blocks:

- Execution order tags do not appear on SimMechanics blocks.
- SimMechanics blocks do not invoke user-defined callbacks.

- You cannot tune SimMechanics block parameters during simulation.
- You cannot set breakpoints on SimMechanics blocks.
- Reusable subsystems cannot contain SimMechanics blocks.
- You cannot use the Simulink Fixed-Point Settings interface with SimMechanics blocks.
- The Report Generator reports SimMechanics block properties incompletely.

Restrictions on Two-Dimensional Simulation

Certain blocks are not supported in two-dimensional simulation mode. These include disassembled joints, massless connectors, and joints that can move in three dimensions. See "Choosing Your Machine's Dimensionality" on page 5-4.

Restrictions with Generated Code

Code generated from models containing SimMechanics blocks has certain limitations.

Stiction-Related Algebraic Loops Disabled

Stiction implemented with Joint Stiction Actuator blocks requires algebraic loops iterated at a single time step to detect discrete events. In generated code versions of models with stiction (including models run with Simulink Accelerator), the mode iteration to determine joint locking and unlocking instead occurs over multiple time steps, possibly reducing simulation accuracy.

Closed-Loop Limitations

Closed-loop models in certain analysis mode configurations use nonlinear solvers with no upper limit on iterations. Code generated from such models is valid but, in general, not truly "real time." These configurations include:

- Forward Dynamics mode when **Constraint solver type** in the Machine Environment block is set to `Machine Precision` or `Tolerancing`
- Kinematics mode

Restrictions on Code Generated from Two-Dimensional Machines

If you generate code from a model containing one or more machines simulated in two dimensions, the generated code is also restricted to two-dimensional motion. Thus, if you change run-time parameters in the generated code, you must ensure that the new values do not violate the two-dimensional motion restriction.

The choice of machine dimensionality is either automatic or manual, but this restriction on generated code applies in either case. See “Choosing Your Machine’s Dimensionality” on page 5-4.

Fixed-Point Not Supported by SimMechanics Code Generation

You must run code generated from models containing SimMechanics blocks on floating-point processors.

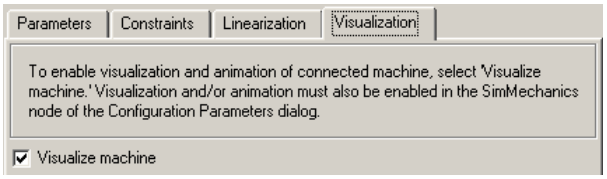
Visualizing and Animating Machines

You can visualize your machine's bodies in SimMechanics with a special window based on MATLAB Graphics. You can also construct your own virtual world and drive it as an external visualization client with SimMechanics signals by constructing your own interface.

Starting SimMechanics Visualization

Starting visualization in SimMechanics requires two choices, one for your entire model, the second for each machine in your model. These choices are part of configuring your model for simulation, as discussed in "Controlling the Simulation" on page 5-11. Implement your visualization choices at any time by clicking **Apply** or **OK**.

- You can choose whether or not to visualize a specific machine in your model through the **Visualization** pane of its Machine Environment block dialog. A single window displays all selected machines in a model. By default, each machine is selected for visualization.

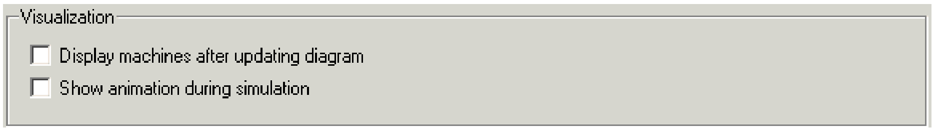


To enable visualization and animation of connected machine, select 'Visualize machine.' Visualization and/or animation must also be enabled in the SimMechanics node of the Configuration Parameters dialog.

[✓] Visualize machine

- You enter the visualization settings for an entire model in the **Visualization** area of the **SimMechanics** node of the **Configuration Parameters** dialog. To open visualization, you must select at least one of these check boxes.

  Model-wide visualization is turned off by default.


Visualization

[ ] Display machines after updating diagram

[ ] Show animation during simulation

Rendering Your Machines in Static Display

Select the **Display machine after updating diagram** check box if you want the SimMechanics visualization window to display the machines in your model in a static rendering. Once you select this check box, you can synchronize the display with changes in your model by selecting **Update Diagram** in the Simulink **Edit** menu. To open the static visualization the first time, you must update your model after selecting this check box.

This check box is not selected by default.

Animating Your Machines During Simulation

Select the **Show animation during simulation** check box if you want the SimMechanics visualization window to animate your machine as it moves during the simulation. If you select this check box, but not the static rendering, the visualization opens only when you start the simulation.

This check box is not selected by default.

Other SimMechanics Visualization Controls

All other visualization controls are located on the SimMechanics visualization window itself. You can access them once the window is open.

Using SimMechanics Visualization

This chapter guides you in making the appropriate visualization choices within SimMechanics. This section explains why you might want to visualize your machine's and animate its motion.

You can find more on the choice of body shapes in the next section:

- "Rendering Body Shapes in SimMechanics" on page 6-5

The three sections that follow explain the SimMechanics visualization controls:

- "Introducing the SimMechanics Visualization Window" on page 6-11
- "Controlling Machine Displays in SimMechanics" on page 6-18
- "Animating SimMechanics Simulations" on page 6-24

Rendering Versus Animation

SimMechanics visualization serves two distinct purposes, static and dynamic visualization. In both cases, you can change your observer viewpoint and navigate through the scene. You can change the body properties of the visualization only by changing the corresponding Body blocks in your model. Changing a body's mass, inertia tensor, and coordinate systems can change its visual rendering.

Static Rendering

Static rendering of machines in their initial state, during construction. Either choice is valid:

- Open the visualization before or while you build your model. You render each body as you add it to your model.

  Having the visualization window open during model building lets you keep track of your machine parts and how they are connected. You can see unphysical or mistaken constructions before you finish the model.
- Open the visualization after you finish the model. All the bodies in the model appear together.

Dynamic Animation

Animation of machines while the SimMechanics model is running. Use this feature to watch the model's dynamics in three dimensions and visualize motions and relationships more easily than is possible with Scope blocks alone. Chapter 5, "Running Mechanical Models"presents the steps for configuring and running SimMechanics models.

Creating an External Virtual Reality Client

You can bypass standard SimMechanics visualization by creating a virtual reality world of your own design to visualize your machine's bodies. With Virtual Reality Toolbox, you can build a custom interface from your model to the virtual world and animate its virtual bodies:

- "Custom Visualization with Virtual Reality" on page 6-28

Rendering Body Shapes in SimMechanics

The visualization window renders the bodies in either of two shapes:

- Equivalent ellipsoid for each body, based on its mass properties and center of gravity (CG) position, explained in “Equivalent Ellipsoids” on page 6-5
- Convex hull for each body, based on its Body coordinate systems (CSs), explained in “Convex Hulls” on page 6-8

Choosing the Body Shape

You choose the body rendering in the special SimMechanics toolbar of the SimMechanics visualization window. You can render the bodies of your machine with either convex hulls, or ellipsoids, or both. See “Introducing the SimMechanics Visualization Window” on page 6-11 for visualization control details.

Equivalent Ellipsoids

The inertia tensor $\boldsymbol{I}$ of a rigid body is real and symmetric, so it has three real eigenvalues (I_1, I_2, I_3) and three orthogonal eigenvectors. These eigenvectors are the principal axes of the body. In the coordinate system defined by those axes, the inertia tensor is diagonal. The *trace* of the inertia tensor, $\mathrm{Tr}(\boldsymbol{I}) = I_1 + I_2 + I_3$, is the same in any coordinate system with its origin at the body’s center of gravity (CG).

Every rigid body has a unique equivalent ellipsoid, a homogeneous solid ellipsoid of the same inertia tensor. The ellipsoid surface is given by

$$\left(\frac{x}{a_x}\right)^2 + \left(\frac{y}{a_y}\right)^2 + \left(\frac{z}{a_z}\right)^2 = 1$$

The three parameters (a_x, a_y, a_z) are the *generalized radii* of the ellipsoid. For axis i = 1,2,3,

$$a_i = \sqrt{5\left[Tr(\boldsymbol{I}) - 2I_i\right]/(2m)}$$

Triangle Inequalities

The *principal moments* (I_1, I_2, I_3) must satisfy the *triangle inequalities:*

$$I_2 + I_3 \geq I_1$$
$$I_3 + I_1 \geq I_2$$
$$I_1 + I_2 \geq I_3$$

Violation of the triangle inequality for I_i leads to an unphysical imaginary generalized radius a_i.

Caution Visualizing the equivalent ellipsoid of a body whose principal moments do not satisfy the triangle inequalities leads to a SimMechanics warning indicating that one or more triangle inequalities have been violated. The simulation continues, but the body in violation is not displayed.

Ellipsoids with Special Symmetry

In general, all three I_i, i = 1,2,3, are unequal. Such a body is an *asymmetric top*. If two of the three I_i are equal (double degeneracy), the body is a *symmetric top*. The third axis is the axis of symmetry. If all three I_i are equal (triple degeneracy), the body is a *spherical top* and dynamically equivalent to a homogeneous sphere.

Reduced-Dimension Ellipsoids

In special cases, the equivalent ellipsoid reduces to a planar, linear, or point figure.

Let (i,j,k) label the three axes (1,2,3) = (x,y,z) in any order.

- For a true *ellipsoid*, with nonzero volume, all the a_i are nonzero. The triangle inequalities are strict inequalities in this case:

  $$I_j + I_k > I_i$$
  $$I_k + I_i > I_j$$
  $$I_i + I_j > I_k$$

- For an *ellipse*, with zero volume but nonzero area, one $a_\mathrm{i} = 0$ and the other two a_j, a_k are nonzero. One of the triangle inequalities becomes an equality:

  $$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &> I_j \\ I_i + I_j &> I_k \end{aligned}$$

- For a *line*, with zero volume and area but nonzero length, two a_i, $a_\mathrm{j} = 0$ and the third a_k is nonzero. Two of the triangle inequalities become equalities:

  $$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &= I_j \\ I_i + I_j &> I_k \end{aligned}$$

  Equivalently, $I_\mathrm{i} = I_\mathrm{j}$ are nonzero and $I_\mathrm{k} = 0$.

- For a *point*, with no spatial size, all three a_i vanish. All three triangle inequalities become equalities:

  $$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &= I_j \\ I_i + I_j &= I_k \end{aligned}$$

  Equivalently, all three I_i vanish.

Example: Simple Pendulum Rod

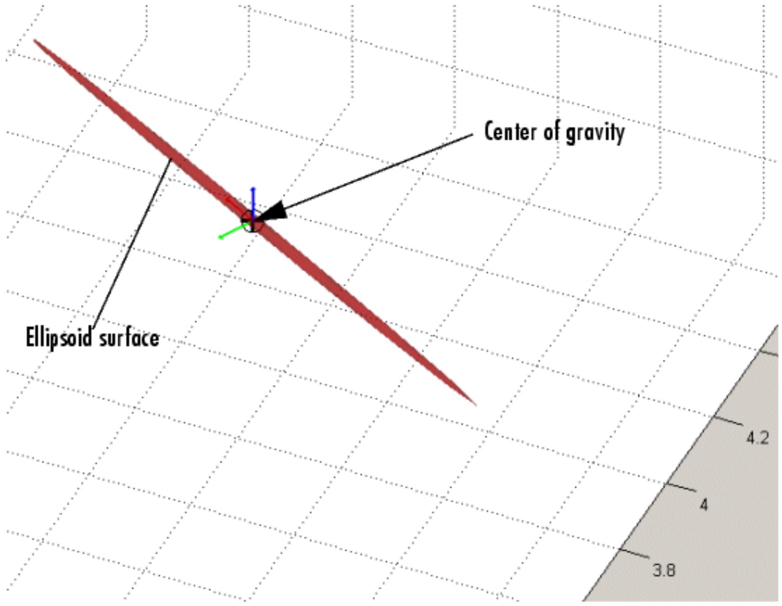
Consider the simple pendulum rod in “Visualizing a Simple Pendulum” on page 2-30. You can open the model by entering `mech_spen` at the command line.

The rod length $L = 1$ m, and its radius $r = 1$ cm. The inertia tensor is

$$\begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} = \begin{pmatrix} mr^2/2 & 0 & 0 \\ 0 & mL^2/12 & 0 \\ 0 & 0 & mL^2/12 \end{pmatrix}$$

Because the rod has an axis of symmetry, the *x*-axis in this case, two of its three principal moments are equal: $I_{yy} = I_{zz}$, and two of its three generalized radii are equal: $a_y = a_z$. The rod is a symmetric top and, since *r* is much smaller than *L*, its equivalent ellipsoid is almost a line of zero volume and area.

The generalized radii of the equivalent ellipsoid are $a_x = \sqrt{5/3}\,(L/2) = 0.646$ m and $a_y = a_z = \sqrt{5}\,(r/2) = 1.12$ cm. This is the rod so rendered:



Convex Hulls

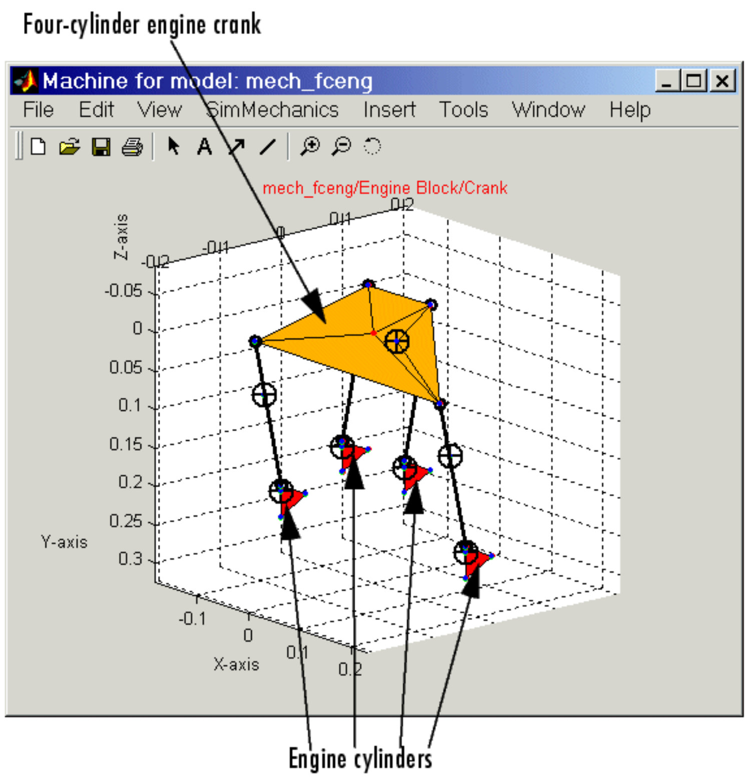
Every Body has at least one Body coordinate system (CS) at the CG. A Body also has one or more extra Body CSs for the attached Joints, as well as possible Actuators and Sensors. Each Body CS has an origin point, and the collection of all these points, in general, defines a volume in space. The minimum outward-bending surface enclosing such a volume is the convex hull of the Body CSs, and this is the alternative way that SimMechanics can render a body.

To enclose a nonzero volume, the set must have at least four non-coplanar Body CSs. Three non-collinear Body CSs are rendered instead by a triangle, and two non-coincident origins by a line. One is displayed just as a point. (The minimum one Body CS would be just the CG CS.) Four or more coplanar origins are rendered by a triangle, three or more collinear origins are rendered by a line, and two or more coincident origins are rendered by a point.

Example: Four-Cylinder Engine Crank

Refer to the four-cylinder engine model of the Demos library by entering `mech_fceng` at the command line.

Double-click the Engine Block subsystem and note the Crank block representing the engine crank. This Body block has six Body CSs. Visualize the engine as convex hulls with the SimMechanics visualization window. The large block in your visualization is this engine crank, and it encloses a nonzero volume.


Four-cylinder engine crank
Machine for model: mech_fceng
File Edit View SimMechanics Insert Tools Window Help
mech_fceng/Engine Block/Crank
Z-axis
Y-axis
X-axis
-0.2
-0.1
0
0.1
0.2
-0.05
0
0.05
0.1
0.15
0.2
0.25
0.3
-0.1
0
0.1
0.2
Engine cylinders

Four Cylinder Engine Example: Engine Crank Convex Hull (Yellow)

Introducing the SimMechanics Visualization Window

The MATLAB Graphics-based visualization window is part of SimMechanics. With it, you can customize how your machines are displayed, interact with your model, and watch and record animations.

The SimMechanics visualization window features special controls. It also uses a special set of symbols to draw bodies and Body coordinate systems (CSs). This section is an overview of what you can do with the SimMechanics window:

- "Interpreting the Special SimMechanics Symbols" on page 6-12
- "Using the Standard MATLAB Graphics Controls" on page 6-13
- "Accessing the Special SimMechanics Features" on page 6-14
- "Saving and Recalling Display Settings" on page 6-16

Note This section mainly focuses on features specific to SimMechanics MATLAB Graphics visualization. Certain standard MATLAB Graphics features are disabled or missing in the SimMechanics visualization window.

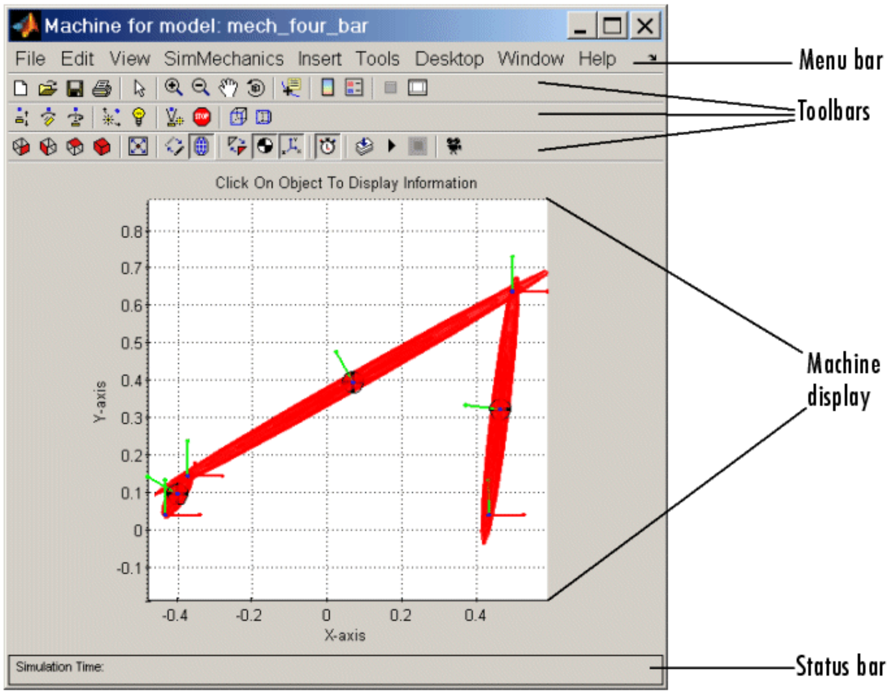
Refer to the MATLAB Graphics documentation for a full discussion of the graphics tools of MATLAB.

Opening and Updating the SimMechanics Visualization Window

Starting visualization and choosing the display options for the machines in your model are discussed in "Starting SimMechanics Visualization" on page 6-2.

Once you configure and start SimMechanics visualization, a customized MATLAB Graphics figure window opens and displays the machines in your model that you have chosen to visualize. One window displays all selected machines simultaneously.

To synchronize the static visualization rendering of your machines with your model, select **Update Diagram** from the model **Edit** menu at any time.


Machine for model: mech_four_bar
File Edit View SimMechanics Insert Tools Desktop Window Help
Click On Object To Display Information
Y-axis
X-axis
Simulation Time:
Menu bar
Toolbars
Machine display
Status bar

SimMechanics Visualization Window Displaying a Machine

Interpreting the Special SimMechanics Symbols

When SimMechanics opens a visualization window, it uses special conventions to render the bodies of your machine. You can control some of these conventions through the **SimMechanics** menu (see “Changing Machine Display Symbols” on page 6-19).

The machine bodies are rendered by one of the two SimMechanics body shapes (see “Rendering Body Shapes in SimMechanics” on page 6-5):

- The window displays all the machine’s bodies. If a body has one of these associated surfaces, the surface is shaded red:

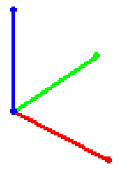
   - Equivalent ellipsoid
   - Convex hull of one or more surface patches (a triangle of coplanar points or an enclosing surface of four or more points)

  A line convex hull is a red stick figure.

- Surfaces used in marking out enclosed convex hull volumes or planar triangles are tiled with patches.

The rendering uses two special symbols:

- The center of gravity (CG) point of each body is marked by a circle-plus symbol ⊕.
- Each Body CS is marked by coordinate axis triads. The color coding is X-Y-Z axes = RGB = red-green-blue.

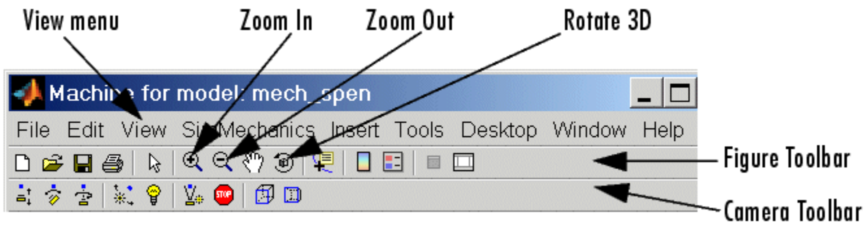


Using the Standard MATLAB Graphics Controls

The main controls of the **Figure Toolbar** are standard on all MATLAB Graphics windows. Refer to the MATLAB Graphics documentation to learn how to configure graphics windows and objects.

For SimMechanics visualizations, the **Zoom in**, **Zoom out**, and **Rotate 3D** buttons are especially useful. Select the button and click in the display area to activate the feature. Click, hold, and roll the figure to rotate the machine in three dimensions.

The useful **Camera Toolbar**, also discussed in the MATLAB Graphics documentation, is enabled by default in SimMechanics. You can enable or disable it, as well as the **Figure Toolbar**, from the **View** menu. "Changing Perspective and Window Size" on page 6-22 discusses its role in SimMechanics visualization.

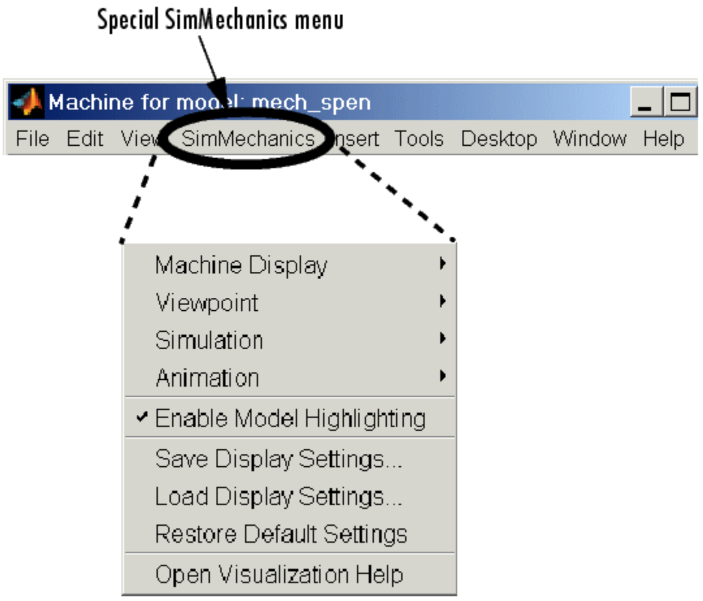

View menu
Zoom In
Zoom Out
Rotate 3D
Machine for model: mech_spen
File Edit View SimMechanics Insert Tools Desktop Window Help
Figure Toolbar
Camera Toolbar

Accessing the Special SimMechanics Features

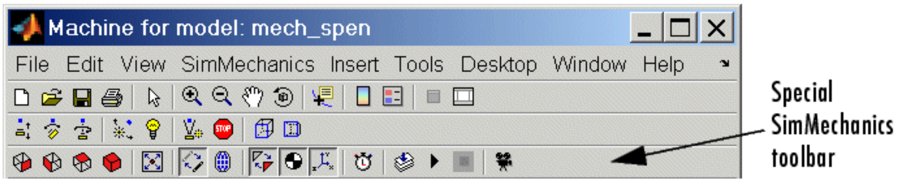
You can perform a variety of tasks with the special **SimMechanics** menu, submenus, and toolbar.

Once you open a visualization window in SimMechanics, you have two ways to control the machine display and carry out these tasks:

- Use the **SimMechanics** menu in the middle of the menu bar. The menu contains special items and submenus. See "The SimMechanics Visualization Menu" on page 6-15.


Special SimMechanics menu
Machine for model: mech_spen
File Edit View SimMechanics Insert Tools Desktop Window Help
Machine Display
Viewpoint
Simulation
Animation
✓ Enable Model Highlighting
Save Display Settings...
Load Display Settings...
Restore Default Settings
Open Visualization Help

- Use the buttons in the SimMechanics toolbar. Every feature on this toolbar occurs in the SimMechanics menu, although the reverse is not true. See "The SimMechanics Visualization Toolbar" on page 6-16.


Machine for model: mech_spen
File Edit View SimMechanics Insert Tools Desktop Window Help
Special SimMechanics toolbar

Certain options in the **SimMechanics** menu toggle between enabled and disabled; for example, **Enable Model Highlighting**. By selecting the item, you either enable it, and a check mark appears, or you disable it, and the check mark disappears.

Other menu items, when you select them, instead trigger an immediate action, for example, **Simulation > Start**.

The SimMechanics Visualization Menu

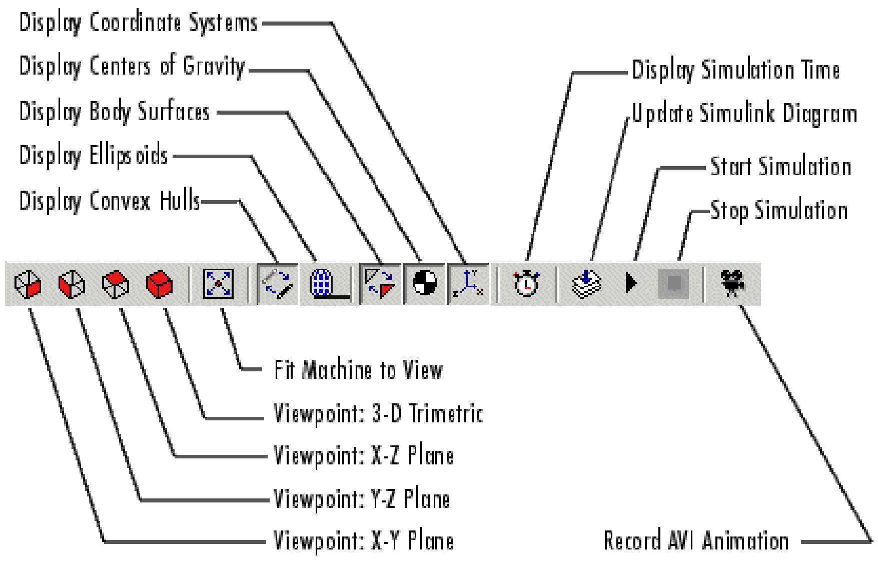
These are the top-level items in the **SimMechanics** menu.

Menu Item or Group	Feature
Machine Display	Control machine rendering
Viewpoint	Control perspective, axes, and window size
Simulation	Control model simulation
Animation	Record simulation animation
Enable Model Highlighting	Highlight Body blocks
Display Settings	Save, load, and restore SimMechanics display settings
Open Visualization Help	Open Help browser to this section of SimMechanics documentation

The SimMechanics Visualization Toolbar

You can activate many **SimMechanics** features by selecting buttons on the SimMechanics toolbar, instead of selecting items in the SimMechanics menu. The setting changes initiated by the toolbar are the same as the corresponding menu actions: either you enable or disable a feature, or you initiate an immediate action.

Hovering your mouse cursor over a toolbar button displays the button's tooltip indicating its function.



SimMechanics Toolbar Features

Saving and Recalling Display Settings

As you work with the **SimMechanics** menu, you might want to save the setting changes that you make. The main menu has three items that allow you to save, load, and reset display settings.

Saving Display Settings

Save Display Settings saves the current configuration of SimMechanics visualization settings. The settings are stored in a MAT-file. By default, this

MAT-file is placed in your current MATLAB directory. (To see your current directory, check the **Current Directory** browser or **Current Directory** field on the MATLAB desktop, or enter `pwd` at the command line.) If the name of your model is `modelname.mdl`, the default settings file is `modelname.mat`. When you save the settings, the **Save Display Settings** browser allows you to change the directory and name of the file. You can save multiple display settings MAT-files under different names. Using an existing MAT-file name overwrites the existing MAT-file.

The name of the *last* saved MAT-file is stored in the Simulink MDL-file itself. You must save your MDL-file in order to save this name.

If you do not choose to store your settings before you close SimMechanics visualization, your settings are not saved, and the MDL-file does not store a name for a MAT-file.

Reopening a Model Visualization

If the MDL-file lacks a name of a display settings MAT-file to load, SimMechanics visualization reverts to the default settings upon restarting.

If you do save a MAT-file of display settings, SimMechanics automatically loads the last saved MAT-file of display settings when you restart SimMechanics visualization after closing your visualization window or the entire MDL-file.

Reloading Display Settings Manually

If you want to load a different settings MAT-file, select **Load Display Settings** and choose a file in the **Load Display Settings** browser.

Resetting Display Settings to Default

If at any time you want to revert to the default display settings documented in this section, select **Restore Default Settings**.

Controlling Machine Displays in SimMechanics

You can change how machines look in the SimMechanics visualization window. This section explains how to control the machine display:

- "Highlighting Bodies and Body Blocks" on page 6-18
- "Changing Machine Display Symbols" on page 6-19
- "Changing Perspective and Window Size" on page 6-22

Highlighting Bodies and Body Blocks

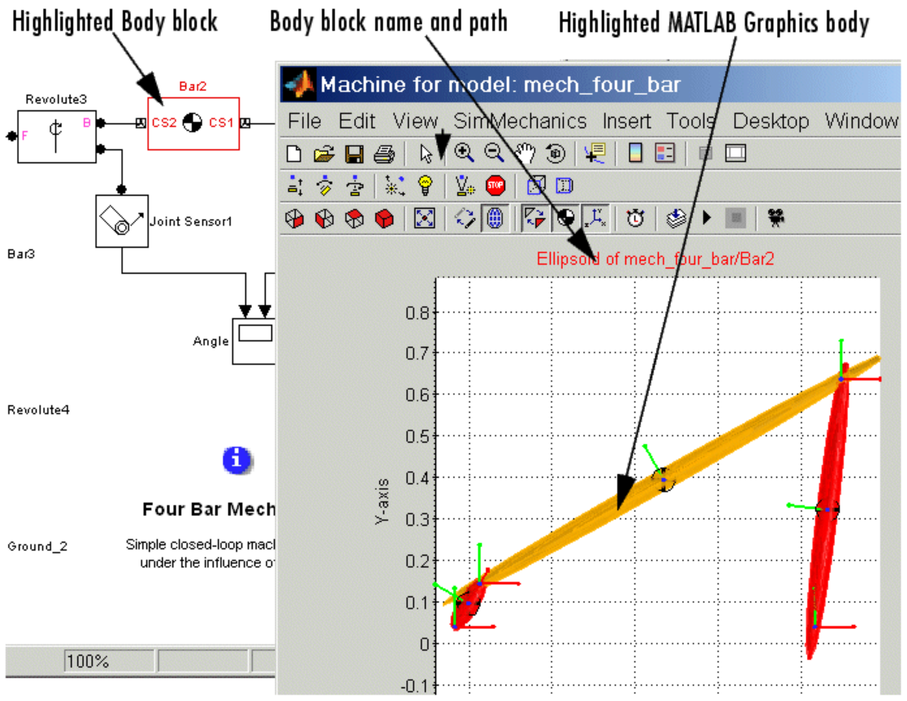
Clicking a body or body coordinate system (CS) in the SimMechanics visualization window causes the following:

- The rendered body surface or CS triad changes color to yellow.
- The visualization window displays the associated Body block or Body CS name and its path on the line just above the machine display.
- The model window comes back to focus with the associated Body block highlighted in red.

To unhighlight, click anywhere in the white area of the machine display. You cannot highlight a body while the simulation is running.

You can enable or disable the highlighting of model Body blocks from the **SimMechanics** menu item **Enable Model Highlighting**. The default is enabled. Whether model highlighting is enabled or disabled, clicking a body in the visualization window always highlights the body in yellow. If you disable model highlighting, the associated Body block is not highlighted when you click the rendered body.

The `mech_four_bar` model from the Demos library is shown with Bar2 (the middle bar) highlighted.



SimMechanics Visualization Window with Highlighted Body and Body Block

Changing Machine Display Symbols

The visualization window uses various symbols to render the machine. (See "Accessing the Special SimMechanics Features" on page 6-14.) You can enable or disable these symbols from the **Machine Display** submenu. Some options depend on which body rendering you choose, equivalent ellipsoids or convex hulls (see "Rendering Body Shapes in SimMechanics" on page 6-5).

Changing Machine Display Symbols

The **Machine Display** submenu has two groups of active items. You can disable or reenable any one or more of these by selecting the items. When an item is disabled, the corresponding symbols immediately disappear from the machine display in the window. When an item is reenabled, the corresponding symbols immediately reappear.

The first group allows you choose whether to display machine bodies as convex hulls, equivalent ellipsoids, or both. By default, the convex hull rendering is enabled, while the equivalent ellipsoids are disabled.

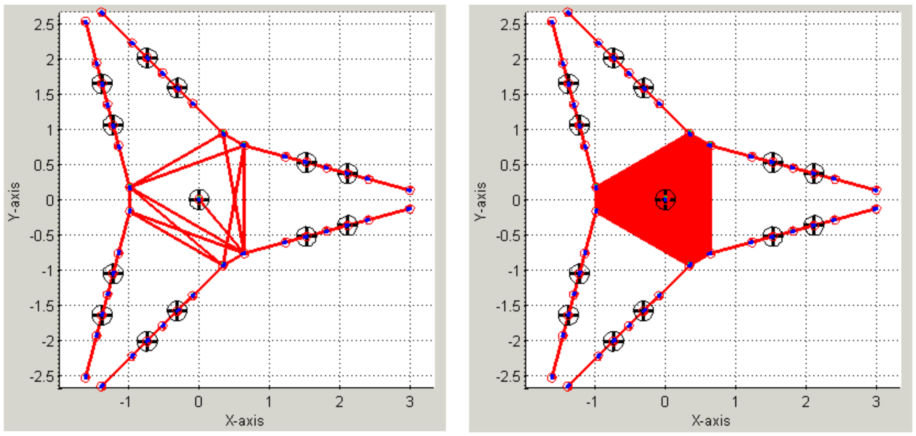
Convex Hulls	Display convex hull for each body
Equivalent Ellipsoids	Display equivalent ellipsoid for each body

The second group allows you to turn body-related symbols on or off. Display of all three is enabled by default.

Body Surfaces	Display convex hull or ellipsoid surfaces
Centers of Gravity (CGs)	Display center of gravity symbols
Coordinate Systems (CSs)	Display coordinate system triads

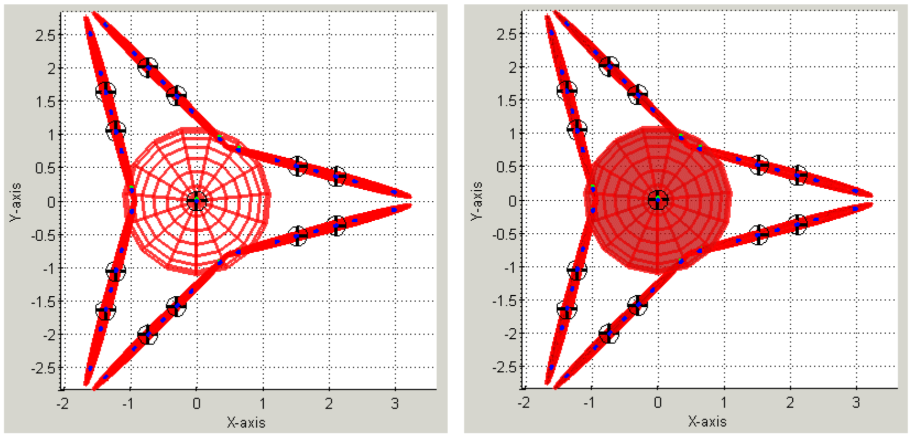
You can try displaying your machine with different combinations of symbols turned on or off. The Stewart platform of the `mech_stewart_trajectory` demo illustrates the possibilities.

- If body surfaces are turned off, convex hulls are rendered by red open wire frame figures. If the body surfaces are turned on, convex hull surfaces are rendered with red solid patches filling in the wire frame.



Convex Hulls Without and With Body Surfaces (Stewart Platform)

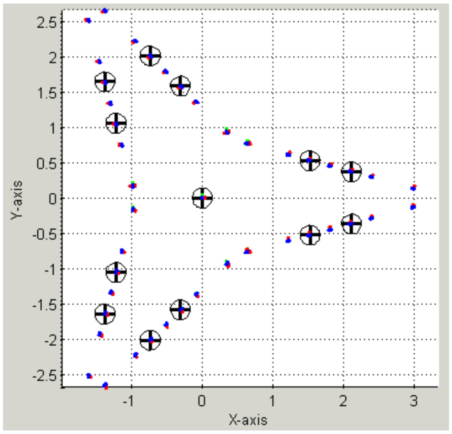
- If body surfaces are turned off, ellipsoids are rendered by red open wire frame figures. If the body surfaces are turned on, the ellipsoid wire frame is filled in by red surface patches.



Equivalent Ellipsoids Without and With Body Surfaces (Stewart Platform)

- If both convex hulls and ellipsoids are turned on, the machine display is a superposition of the two renderings.

- If you turn off both body shapes, bodies are outlined by their centers of gravity (CGs) and body coordinate systems (CSs).



Bodies Outlined by CGs and Body CSs (Stewart Platform)

Changing Perspective and Window Size

You can change your perspective view of the machine and the size of the visualization window with the **Viewpoint** submenu. Selecting a viewpoint immediately changes your view in the window.

Setting a Perspective Automatically

The available automatic perspectives are **X-Y Plane**, **Y-Z Plane**, **X-Z Plane**, and the trimetric **3-D (view([1 -1 1]))**. The coordinate axes in SimMechanics are

+*x* (rightward)

+*y* (upward)

+*z* (out of the screen)

A planar view projects the machine onto the selected plane. The trimetric view displays the machine from the viewpoint along the axis (1, -1, 1).

Setting a Perspective Manually

You can find any perspective you want manually with the features of the **Figure Toolbar** and the **Camera Toolbar**.

To manually rotate the view,

1. Activate **Rotate 3D** from the **Figure Toolbar** or the **Tools** menu.
2. Click and hold your mouse anywhere in the display area, then roll the mouse to roll the viewpoint. Release your mouse to stop the roll.

   To disable the 3-D roll, deactivate the menu item or button.

You can change the visualization window perspective in a more comprehensive way with the camera controls. Find these controls in the **Tools** menu or on the **Camera Toolbar**. With these controls, you can change the viewpoint angle, display projection, and scene lighting. You can also navigate through the machine display.

Note The **Figure Toolbar** and **Camera Toolbar** features are standard MATLAB features fully described in the MATLAB Graphics documentation.

Changing the Window and Axis Size

Fit Machine to View immediately resizes the visualization window and the machine to fit together. It does not change the display perspective.

Enable Automatic Axis Resize forces SimMechanics to resize the axes and window to fit the full range of machine motion during an animation. If this option is disabled, the axes instead shift to follow the machine motion. The default is disabled.

Animating SimMechanics Simulations

This section shows you how, from the SimMechanics visualization window, you can control certain aspects of machine simulation and animation. The SimMechanics controls also allow you to record machine animations.

Controlling the Simulation from the Window

Certain Simulink model features are mapped into the **SimMechanics** menu so you can use them from the SimMechanics visualization window. You find these features in the **Simulation** submenu.

You can start a model (if it is not running) and stop a model (if it is running) from this submenu by selecting **Start** or **Stop**. The keyboard shortcut **Crtl+T** initiates the same actions, if the SimMechanics visualization window or the Simulink model window is in focus.

You can update your Simulink model diagram by selecting **Update Diagram** or by entering **Ctrl+D** at the keyboard with the visualization window or Simulink model window in focus.

You can also enable or disable the display of the simulation running time by selecting **Display Simulation Time** in this submenu. The default is disabled. The simulation running time is shown in the left corner of the status bar at the bottom of the SimMechanics window as `Simulation Time:  ...  sec`.

Changing the Machine Display Refresh Rate

By default, the machine display is updated at every major time step during simulation. You can change this refresh rate by reconfiguring the Simulink output options. Taking more sample points makes the animation smoother. Using fewer sample points leads to a more disjointed animation, but a faster simulation and a smaller recorded animation file (see "Recording and Playing Animations" on page 6-26).

Open **Configuration Parameters** in the **Simulation** menu of your model window. On the **Data Import/Export** node, in the **Save options** area, change the **Output options** pull-down menu entry to `Produce specified output only`. You must now use the **Output times** field on the right to specify explicitly how often Simulink should capture the simulation output. In this field, enter a vector of sample times. The sample time range must be the same as or lie within the **Start time** and **Stop time** range of the **Simulation time** area in the **Solver** node of **Configuration Parameters**.

If you want a uniform output sampling, use the `linspace` command to specify the time range and number of sample points:

```
linspace(start-time, end-time, number-of-points)
```

See the Simulink documentation on the **Configuration Parameters** dialog for more about output options.

Example: Changing the Refresh Sampling Rate

To sample 200 points from zero to 10 seconds,

1. Open **Configuration Parameters** from the **Simulation** menu.
2. Locate **Output options** in the **Data Import/Export** node. Change the pull-down menu to `Produce specified output only`.
3. Enter `linspace(0,10,200)` in the **Output times** field. Click **Apply** or **OK**.

Speeding Up the Animation

As SimMechanics animates your machine's motion, it moves the bodies, whether rendered as convex hulls or as equivalent ellipsoids, and refreshes the body center of gravity (CG) and coordinate system (CS) triad symbols.

You can speed up the animation if you turn off one or more of these symbols:

- Body CGs
- Body CSs
- Ellipsoids or Convex Hulls
- Body Surfaces

The preceding section, "Changing Machine Display Symbols" on page 6-19, explains the full set of display controls and how to enable and disable them. Leave at least one type of symbol enabled in order to see the bodies' motions.

Recording and Playing Animations

The SimMechanics visualization window allows you to record animations of your machine simulations. The animations are stored in Audio Video Interleave (AVI) format. You control animation recording through the **Animation** submenu.

To activate animation recording, enable **Store in AVI File**. The default is disabled. By default, recorded AVI files are placed in your current MATLAB directory. (To see your current directory, check the **Current Directory** browser or **Current Directory** field on the MATLAB desktop, or enter `pwd` at the command line.) If the name of your Simulink model file is `modelname.mdl`, the name of the recorded AVI file is `modelname.avi`. When you enable animation recording, this name appears in the right corner of the status bar at the bottom of the visualization window as **`Animation File:`** `modelname`**`.avi`**.

SimMechanics first records a MATLAB movie by capturing the machine display at every major simulation or output time step. Then, in the termination phase of your simulation, it converts this movie to AVI format and stores it in the AVI file. A small **AVI Conversion** window opens to indicate that conversion and storage are complete and to display the path of the AVI file. Click **OK** to close this prompt.

Caution SimMechanics overwrites any existing file with the same name as the AVI file in the same directory. The AVI file write fails if a file of the same name that is locked by another application exists in the same directory.

Changing the Name and Storage Directory of the Animation File

If you want to change the directory in which your AVI files are stored, you must implement the change *before* beginning the simulation. Change the AVI storage directory by selecting **Choose AVI File Location**. The **AVI File**

Location browser appears. Change to the directory you want the AVI file stored in, then click **Save**.

In the same **AVI File Location** browser, you can also change the name of the stored AVI file before clicking **Save**.

Compressing the Animation File

You can reduce the size of your AVI file by compressing it. Select **Compress AVI File** to enable this feature. The default is enabled.

Caution AVI compression is available only on the Windows platform. MATLAB uses the Indeo 5 compression algorithm. You must have the corresponding video codec installed to

- Compress an animation while recording.
- Decompress and play back a compressed animation.

Check your operating system configuration for installed video codecs.

Playing Back the Animation File

You need an AVI-compatible video player to view the recorded file. You can use the internal MATLAB movie viewer or an external video player.

If you open an AVI file from MATLAB, an **Import Wizard** prompts you to load the AVI stream. Click the **Play Movie** button. The MATLAB **Movie Viewer** opens and runs the animation.

Custom Visualization with Virtual Reality

You can bypass the standard SimMechanics visualization and create a machine animation in a virtual world of your own design. This gives you the power to animate a more realistic visualization of your machine. You create a virtual world, populate it with bodies represented as virtual objects using Virtual Reality Modeling Language (VRML), then interface the virtual world with your SimMechanics model. Creating your own virtual animation requires a new or existing virtual world for your model and an interface between them.

This section explains how to use a separately created virtual world with SimMechanics:

- “Creating Virtual Worlds for SimMechanics Models” on page 6-28
- “Interfacing SimMechanics with Virtual Worlds” on page 6-32

Virtual Reality Visualization This section assumes that Virtual Reality Toolbox is installed on your system and that you are familiar with it and with creating virtual worlds. Refer to the Virtual Reality Toolbox documentation for full details on installing and using this toolbox.

We recommend you allocate generous central processor power, graphics card speed, and memory for the virtual reality feature, especially for animation.

Creating Virtual Worlds for SimMechanics Models

The Virtual Reality Toolbox User’s Guide and VRML books such as Marrin and Campbell [8] explain how to create virtual objects and assemble them into virtual worlds. The best way to become comfortable moving around in a virtual world is to practice with a variety of movement modes and viewpoints. This section highlights the special requirements to make a virtual world usable as a visualization for a SimMechanics model.

Editing Virtual Worlds and VRML Files

As you create a virtual world populated by virtual bodies, you must create each component body, then plan and implement the geometry of the machine’s initial state. Use a VRML authoring tool that can read VRML as a native format to create and edit virtual reality `.wrl` files. Virtual Reality Toolbox includes a VRML authoring tool called V-Realm Builder®. If you are familiar with raw VRML source code, you can use a plain text editor or the MATLAB editor to edit the files.

Representing Bodies as Virtual Objects

You represent each body by a virtual object encoded in a `.wrl` file. You also create a master `.wrl` file to represent the virtual world that refers to body `.wrl` files, placing and orienting these bodies in the larger scene. You can define a body's position and orientation with respect to,

- The overall virtual world, corresponding to the SimMechanics coordinate system World
- Another body in the machine, corresponding to Body coordinate systems in SimMechanics

You can nest body references to other bodies in VRML hierarchies, but you must define at least one body's position and orientation with respect to the overall virtual world. Place and orient the bodies in their initial states, corresponding with the initial state of the SimMechanics simulation.

Each body's `.wrl` file contains a hierarchical tree starting with the Transform node. Among Transform's fields must be translation and rotation fields to specify the body's position and orientation in space. If a body is nested below another body, its position and orientation are defined with respect to the next body up the hierarchy.

Creating your own virtual world gives you great flexibility in representing your machine:

- You can render bodies in as much or as little detail as you want, with shapes, colors, textures, etc., of your own choosing.
- You can include or omit bodies that you do not animate.
- You can create a computer-aided design (CAD) assembly of your machine and export it into VRML files.
- If you only translate a body, you can omit the rotation field from its Transform node.

- If you only rotate a body, you can omit the translation field from its Transform node.

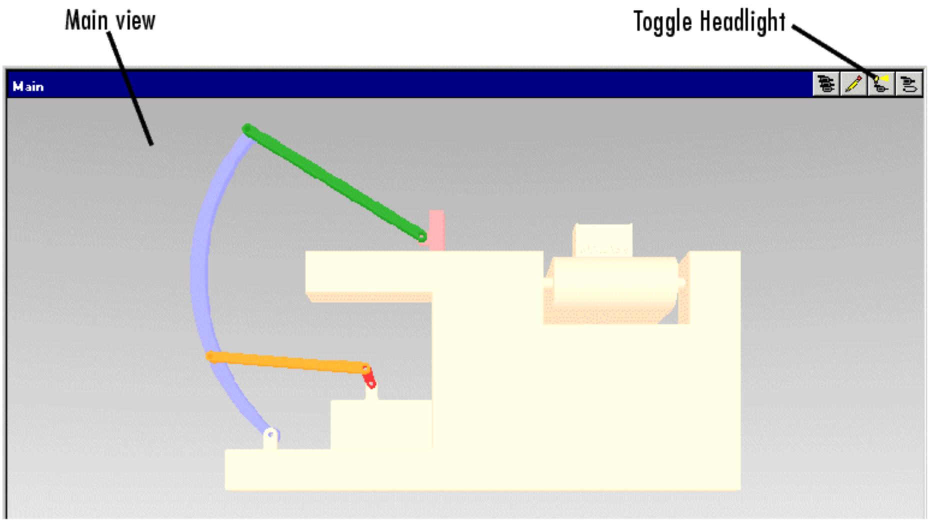
Example: Viewing Custom External VRML Files for the Conveyor Loader

The demo model `mech_conveyor_vr` is a modified version of the original conveyor model `mech_conveyor` and comes with external VRML files containing static renderings of the machine parts in their initial positions. This example uses the V-Realm Builder editor to view the files.

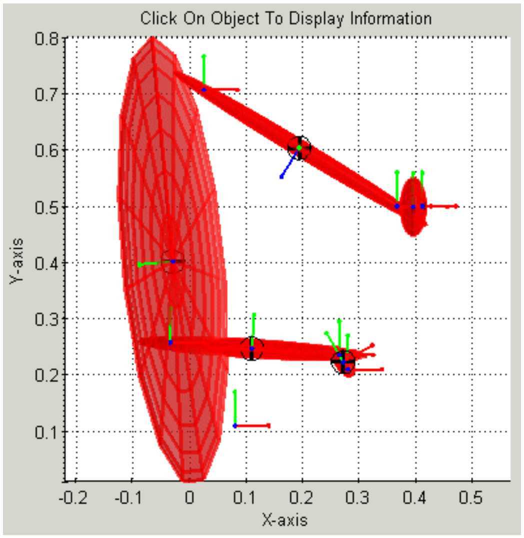
1 In the *`matlabroot`*`\toolbox\physmod\mech\mechdemos\` directory, open these external VRML files with V-Realm Builder:

```
base.wrl, convmech.wrl, link1.wrl, link2.wrl, link3.wrl,
link4.wrl, pusher.wrl
```

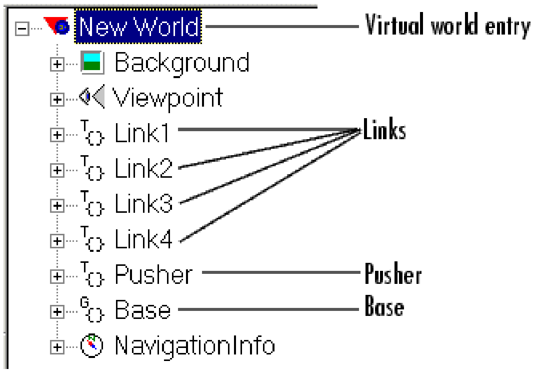
2 With the `conveyor.wrl` world in focus, click the **Test Mode** button in the V-Realm Builder toolbar and view the complete machine in the **Main** view. Right-click in this window to configure the navigation. If the colors seem washed out, toggle off the headlight.



These conveyor parts are more realistic than the equivalent ellipsoids or convex hulls available in SimMechanics visualization:

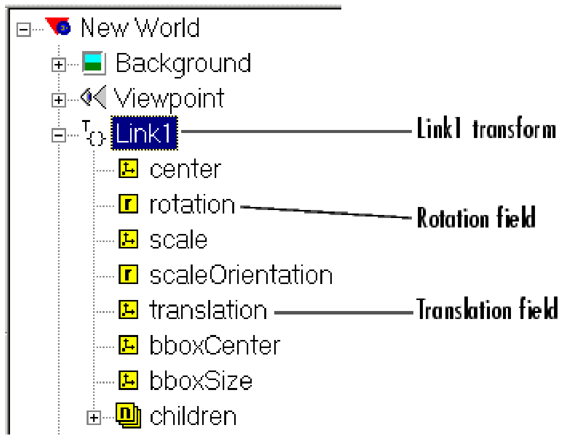


3 On the left side of your VRML editor window, examine the node tree of `convmech.wrl` that refers to the six VRML files representing each component body:



The hierarchy of body position and orientation references is flat in this model. Each body is separately referenced to the top level of the hierarchy, `New World`.

4 Expand one of the nodes. Each body node has, among others, rotation and translation fields:



The exception is the base. Being grounded and immobile, it has neither a translation nor a rotation field.

Interfacing SimMechanics with Virtual Worlds

To animate a body, you need to measure its motion in your SimMechanics simulation and export that information to the virtual world. This requires connecting Body Sensor blocks to the Bodies you want to animate in your model, then creating an interface that animates the virtual bodies with the body sensor motion signals. "Example: Interfacing the Conveyor Loader Model and Virtual World" on page 6-35 applies these steps to the `mech_conveyor_vr` demo.

Adding and Configuring Body Sensors

Refer to "Sensing Body Motions" on page 4-63 for general information on how to use Body Sensors. Connect the Body Sensors to Body coordinate systems (CSs) on the bodies whose motions you want to animate. The Body block reference discusses how to create and configure Body CSs. You need to take these extra steps to export the signals of a body sensor to your virtual world:

1 Make sure the Body Sensor's Body CS reference origin and orientation follow the body's defining VRML hierarchy.

Example: You define a new Body CS on a body to connect the Body Sensor. If you defined the VRML body's position with respect to the center of gravity (CG) of a second, neighboring body in your VRML files, you should set the **Translated from origin of** field of the new Body CS to the origin of the CG CS of the second body.

2 In the Body Sensor dialog, select the **[x; y; z] Position** check box if you want to animate the body's translational motion.

Select the **[3 x 3] Rotation matrix** check box if you want to animate the body's rotational motion.

3 Choose the coordinate system in which the body motions are measured in the **With respect to coordinate system** pull-down menu. You can pick `Absolute (World)` or `Local (Body CS)`. This coordinate system should be the same as the coordinate system used to define the body's position and orientation in the VRML files.

A Simulink output port > appears on the block for each of the motion signals. The translational signal is a 3-vector of spatial coordinates, *(x, y, z)*. The rotational signal is a 9-vector, column-wise representation of the 3-by-3 orthogonal rotation matrix R, $(R_{11}, R_{21}, R_{31}, R_{12}, ...)$.

Animating the Virtual World Bodies

Animating the virtual bodies requires interfacing the body sensor signals in the SimMechanics model with the VRML translation and/or rotation fields in the `.wrl` files. You accomplish this with the VR Sink block, which you can find in the Virtual Reality Toolbox block library. Enter

```
vrlib
```

at the command line. Drag a copy of the VR Sink block into your model.

Open the VR Sink dialog box. (The figure Conveyor Loader Model: VR Sink Dialog Box on page 6-37 displays an example of the dialog.) In the **Source file** field in the **World properties** area, enter the name of the VRML file that represents your model's virtual world. This is the file that refers to the other `.wrl` files representing the component bodies of your machine. If the virtual world VRML file is not in the same directory as your model, enter the file's path relative to the model. Click **Apply**.

In the **VRML tree** window, the node list of the virtual world `.wrl` file appears. Expand the tree of each component body in the list to view that body's check box list. Select the **rotation** and/or **translation** check boxes as needed for each body. A Simulink input port > appears on the block icon for each of these selected check boxes. The ports are labeled **`node`**.**`field`**. The **`node`** is the name for the body. The **`field`** is named either `rotation` or `translation`.

Converting Body Sensor Signals into VRML Format

You are now ready to connect the Body Sensor output signals to the VR Sink block. But you might need to modify those signals for valid use in VRML.

- You can connect the translational motion signal line directly from the output port of the Body Sensor to the **`node`**`.translation` input port on the VR Sink. The VRML node tree directly accepts translation motion as a 3-vector signal of rectangular coordinates *(x,y,z)*.

  Make sure that the translational motion signal refers to the same coordinate system used to define the body's position in the VRML files.

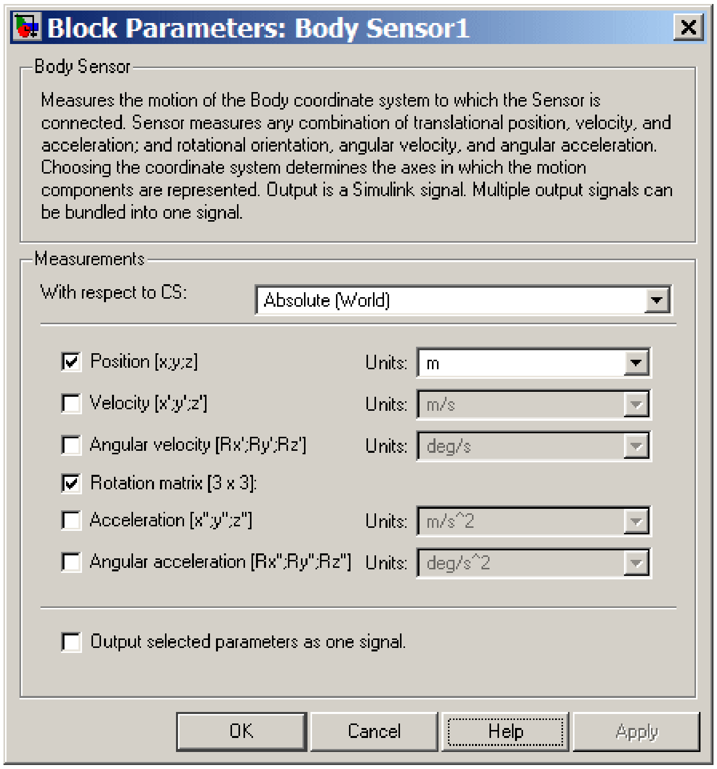
- You cannot directly connect the rotational motion signal line to the VR Sink. The Body Sensor output represents orientation with a 3-by-3 rotation matrix R, while VRML accepts orientation represented as the axis-angle 4-vector form [$\boldsymbol{n}$ θ], where $\boldsymbol{n} = (n_x, n_y, n_z)$ is a 3-vector representing the rotation axis and θ is the rotation angle.

  Open the SimMechanics Utilities library. For each rotational motion signal, drag a RotationMatrix2VR block into your model. Connect the rotation signal from the Body Sensor block to the RotationMatrix2VR block. Then connect the latter block to the corresponding **`node`**`.rotation` input port on VR Sink for that body. This block converts the 3-by-3 R matrix signal into the 4-vector VRML form.

Close the VR Sink block dialog. Your SimMechanics model now animates the virtual world.

Example: Interfacing the Conveyor Loader Model and Virtual World

In the `mech_conveyor_vr` demo model, open the Body Sensor1 block. The block measures the translational and rotational motion of Link3 in the conveyor:



Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

With respect to CS: Absolute (World)

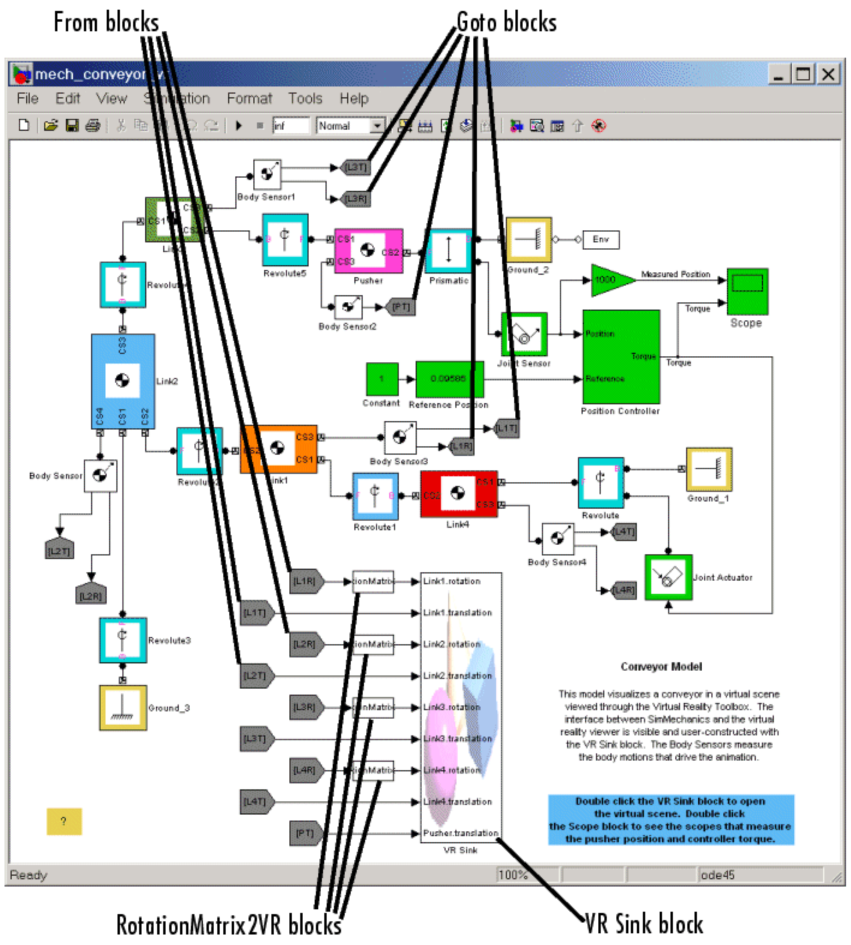
✓ Position [x;y;z]	Units:	m
☐ Velocity [x';y';z']	Units:	m/s
☐ Angular velocity [Rx';Ry';Rz']	Units:	deg/s
✓ Rotation matrix [3 x 3]:		
☐ Acceleration [x'';y'';z'']	Units:	m/s^2
☐ Angular acceleration [Rx'';Ry'';Rz'']	Units:	deg/s^2

☐ Output selected parameters as one signal.

The Body Sensor1 block has two Simulink output signals.

The other Body Sensor blocks are similar, except for Body Sensor2, which measures only the translational motion of the pusher. All the Body Sensors measure body motions with respect to World, the frame in which the conveyor base is at rest. Each motion signal represents the body's displacement relative to its initial position.

The `mech_conveyor_vr` model contains a Virtual Reality Toolbox interface to the model's custom VRML files.



Conveyor Model

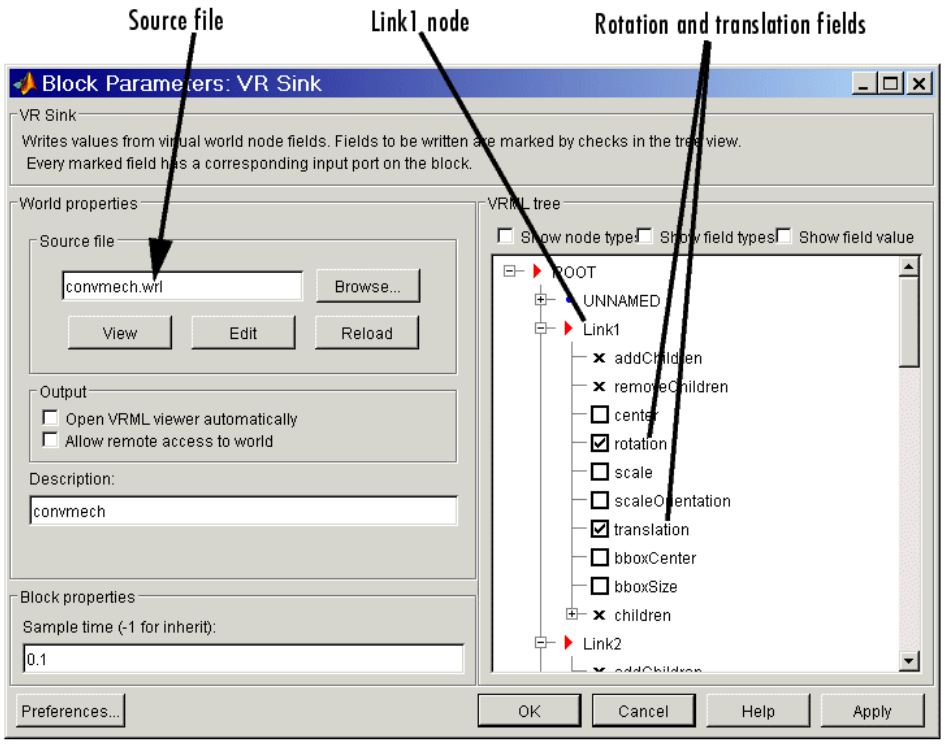
This model visualizes a conveyor in a virtual scene viewed through the Virtual Reality Toolbox. The interface between SimMechanics and the virtual reality viewer is visible and user-constructed with the VR Sink block. The Body Sensors measure the body motions that drive the animation.

Double click the VR Sink block to open the virtual scene. Double click the Scope block to see the scopes that measure the pusher position and controller torque.

Conveyor Loader Model with Custom Virtual Reality Interface

1 Trace each body sensor signal through the model. The signals are routed through pairs of Simulink Goto and From blocks.

2 Open the VR Sink block. The **Source file** is `convmech.wrl`, the master file for this virtual world. The **VRML tree** on the right reproduces the node tree visible in the VRML editor for `convmech.wrl`.



Writes values from virtual world node fields. Fields to be written are marked by checks in the tree view. Every marked field has a corresponding input port on the block.

Source file: convmech.wrl

[ ] Open VRML viewer automatically
[ ] Allow remote access to world

Description: convmech

Sample time (-1 for inherit): 0.1

[ ] Show node types [ ] Show field types [ ] Show field value

[ ] center
[✓] rotation
[ ] scale
[ ] scaleOrientation
[✓] translation
[ ] bboxCenter
[ ] bboxSize

Conveyor Loader Model: VR Sink Dialog Box

3 Expand and scroll down the VRML trees. The trees for Link1, Link2, Link3, Link4, and Pusher list the field inputs for accepting motion signals.

- The Link component bodies require both translational and rotational motions. All the Links have actively selected check boxes for their **rotation** and **translation** field inputs.
- The Pusher body requires only translational motion. Only the **translation** field check box is selected for the Pusher.

Each of the nine Simulink input ports on the VR Sink block is named **`node.field`**. The Base of the conveyor does not move, so its node has no motion input fields.

4 In the VR Sink dialog, click **View** in the **World properties / Source file** area.

   Your Virtual Reality Toolbox viewer opens, displaying the conveyor machine scene. The scene is identical to that visible in the VRML editor (see "Example: Viewing Custom External VRML Files for the Conveyor Loader" on page 6-30).

5 Close all the dialog boxes by clicking **OK**, leaving the viewer open.

6 Click the **Start** button in the model window.

   As in the original `mech_conveyor` demo, starting the model opens the **Reference Position** slider bar that you can move from side to side. As you do so, watch the pusher in the viewer move in parallel.

Modeling with Computer-Aided Design

Using SimMechanics with computer-aided design (CAD) extends your mechanical modeling and simulation capabilities, allowing you to create SimMechanics models from CAD assemblies.

Creating a CAD-Based Robot Arm Model (p. 7-39)	Generating and completing a full SimMechanics model from an exported robot arm assembly
Modeling a Stewart Platform in CAD (p. 7-46)	Constructing and translating a six-degree-of-freedom Stewart platform assembly

Introducing CAD Translation

Computer-aided design (CAD) tools allow you to model machines geometrically as collections of parts, or *assemblies*. Simulink and SimMechanics use a block diagram approach to model control systems around mechanical devices and simulate their dynamics. The block diagram approach does not include full geometric information, nor do CAD assemblies typically incorporate controllers or allow you to perform dynamic simulations. With CAD translation, you can combine the power of CAD and SimMechanics.

The translator transforms geometric CAD assemblies into Simulink block diagram models. The intermediary between a CAD assembly and its SimMechanics model is an XML file in a Physical Modeling format. This section covers what you need to get started with CAD translation:

- "CAD Translation Software Requirements" on page 7-3
- "Overview of the CAD Translation Steps" on page 7-3
- "Installing the CAD-to-SimMechanics Translator" on page 7-5

CAD Translation Software Requirements

Before starting to use CAD with SimMechanics, you must first

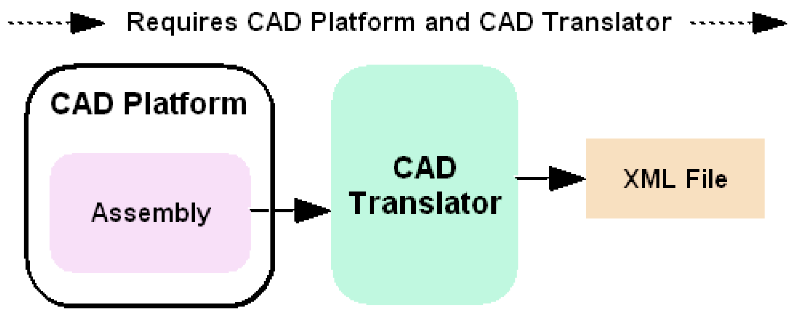
1. Install your CAD platform or CAD software application.
2. Install the CAD-to-SimMechanics translator appropriate to your CAD platform, using the translator installer. See "Installing the CAD-to-SimMechanics Translator" on page 7-5.

Overview of the CAD Translation Steps

Using the CAD-to-SimMechanics translator with a CAD assembly requires two major steps, exporting the CAD assembly into XML and importing the XML to create a SimMechanics model.

Exporting an Assembly

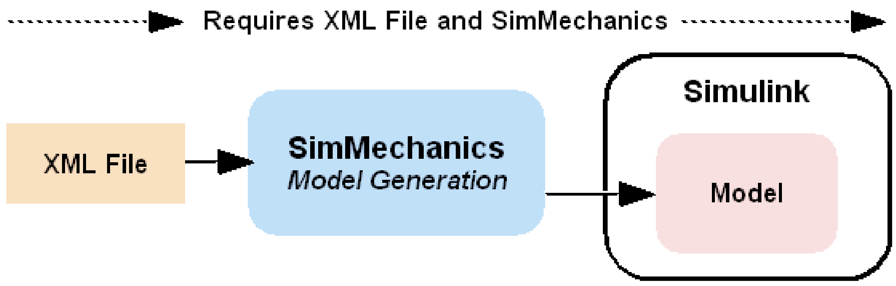
You export the assembly from the CAD platform into a Physical Modeling XML file that you can later use with SimMechanics. This step requires the CAD platform and the platform-specific CAD-to-SimMechanics translator, but not MATLAB. See “Exporting CAD Assemblies into Physical Modeling XML” on page 7-6.



Exporting a CAD Assembly into a Physical Modeling XML File

Importing a Model

You then convert the Physical Modeling XML file into a SimMechanics model in Simulink. This step requires the XML file and SimMechanics, but not the CAD platform or the translator. See “Creating Models from Physical Modeling XML” on page 7-14.



Importing a Physical Modeling XML File into SimMechanics

Installing the CAD-to-SimMechanics Translator

You do not need any MATLAB components to install the CAD-to-SimMechanics translator, but the target CAD platform must be installed.

Downloading the Translator

To obtain the translator, locate and download the self-extracting installer specific to your CAD platform and operating system from the SimMechanics product Web page, `www.mathworks.com/products/simmechanics/`.

Installing the Translator

Once you have obtained the translator download, run the installer and follow the instructions in the README file provided. Print this file for future reference.

Installing the Translator over a Network

You can install the translator with the installer, your CAD platform, or both, on your network. Use the correct paths to specify the locations. If you can, map these network locations as drives on your computer.

Linking the Translator to Your CAD Platform

To configure your CAD platform to work with the translator, you need to link the translator to the CAD platform so that it is enabled and available as you work with an assembly.

Consult the translator's README instructions file and your CAD platform's documentation for further information about linking.

Finding the Translator Help and Example Files

Once your translator is installed, the target installation directory includes subdirectories containing help files for your CAD platform and example files containing preconstructed assemblies to learn with.

Exporting CAD Assemblies into Physical Modeling XML

The CAD-to-SimMechanics translator converts an existing computer-aided design (CAD) assembly into a Physical Modeling XML file that is portable and independent of SimMechanics. From this XML file, you can generate a SimMechanics model, as discussed in "Creating Models from Physical Modeling XML" on page 7-14.

Note You do not need MATLAB or any MATLAB component to use the CAD-to-SimMechanics translator, but you do need to have the target CAD platform installed.

When you export from your CAD platform, you must export a complete assembly into XML, not just a part.

This section explains how to export CAD assemblies into the Physical Modeling XML format.

- "Building a CAD Assembly for SimMechanics" on page 7-6 explains the requirements for a CAD assembly to produce a valid SimMechanics model.
- "Translating CAD Assemblies into XML" on page 7-10 walks you through the steps to export the XML file representing the CAD assembly.
- "Troubleshooting Assembly Export Problems" on page 7-12 examines some of the problems you might encounter when you export a CAD assembly into a Physical Modeling XML representation.
- "Getting Help in the Translator Window" on page 7-12 shows you how to get online help while you are working with the translator.

Building a CAD Assembly for SimMechanics

The CAD-to-SimMechanics translator creates a Physical Modeling XML file that represents the assembly's parts as bodies and maps the constraints between the parts into joints. You need to specify enough information in your CAD assembly for SimMechanics to construct a dynamically meaningful model from the XML file.

CAD Assembly Component	Corresponding SimMechanics Blocks
Part	Body
Constraints *	Joints
Fundamental Root	Ground – Root Weld – Root Body
Subassembly	Subsystem
Subassembly Root: [[*trunk*]] – constraint(s) – *subassembly*	[[Root Body – Root Weld – Fixed Body]] – Joint – *subsystem* ....
Fixed Part (in a subassembly)	Root Body – Weld – Body

* Constraints on parts in a CAD assembly are sometimes called *mates*.

Roots and Root Bodies

Every CAD assembly has a single *fundamental root*, a fixed point that does not move. The positions and orientations of all parts refer directly or indirectly to this fundamental root. The translator converts the fundamental root into a unique Ground – Weld – Body combination in SimMechanics.

A *root body* is a zero-mass, zero-inertia body used in the generated SimMechanics model to represent a CAD root. A root body is always welded to ground, so that its zero mass and zero inertia do not affect the model's dynamics. A root body is necessary, in general, to represent a fixed anchor for part constraints in the original assembly. This body can carry multiple coordinate systems for this purpose, while the single Ground block in the generated model can carry only one.

Subassemblies and Hierarchies

You can isolate a collection of CAD components (parts and their constraints) into a *subassembly*. The translator converts subassemblies into subsystems in SimMechanics.

The main assembly is like the trunk of a tree, and its subassemblies are like the branches of the tree. Subassemblies can have subassemblies, and so on. This tree is the assembly's *hierarchy*. Each CAD subassembly has its own *subassembly root*. A *fixed part* of a CAD subassembly is a part that is welded to the subassembly root. It cannot move relative to the subassembly root.

See "Creating a CAD-Based Robot Arm Model" on page 7-39 for an example of subassembly hierarchy.

Improving Your Assembly with Subassemblies

Use subassemblies to organize your assembly hierarchically. This will simplify your subsequent SimMechanics model by grouping blocks into corresponding subsystems. Follow these guidelines to ensure that your CAD assembly translates into a functioning SimMechanics model:

- You must have at least one fixed part inside each subassembly. (See "Subassemblies and Hierarchies" on page 7-7 for more on fixed parts.)
- Put as many welded components as you can inside rigid subassemblies or combine welded components into a single equivalent part.

  If your assembly has a group of parts that do not move relative to one another, model them so that the translator treats this group as a single part, eliminating unnecessary Body and Joint blocks from your subsequent SimMechanics model.
- Avoid imposing constraints between a subassembly part and geometrical abstractions such as the *x-y*, *y-z*, and *x-z* planes of the subassembly. Instead, impose constraints between the part and a fixed part in the subassembly. Normally, this fixed part is the subassembly root and translates to the Fixed Body in the Root Body – Root Weld – Fixed Body sequence.

Mass Properties of Assembly Parts

The CAD assembly's parts need to have masses and inertia tensors. When you generate the SimMechanics model, this mass property information is used to specify the properties of the SimMechanics Body block corresponding to each assembly part.

Your CAD platform might compute masses and inertia values from the mass density and geometry of the assembly parts. Otherwise, you must specify the mass and inertia tensor with respect to the part's center of gravity. The translator computes the center of gravity of each part automatically.

See "Exporting a CAD Part" on page 7-23 for an example.

Constraint Geometries

The constraints in your CAD assembly restrict how the assembly's parts can move with respect to each other. Without any constraints, a pair of CAD parts can move with six unrestricted degrees of freedom (DoFs) relative to one another. Constraints between pairs of parts reduce the six to fewer DoFs. In SimMechanics, joints express DoFs between bodies because bodies by themselves carry no DoFs. Constraints and joints are complements of one another.

You must specify the constraint geometry in the CAD assembly consistently and in enough detail for SimMechanics to reconstruct the assembly's DoFs as joints. The relationship between constraints in CAD and joints in SimMechanics is not, in general, a simple mapping. Some SimMechanics joints have only one DoF, while others represent more than one DoF. The translator often combines multiple DoFs into one joint. Constraint specification details often depend on the specific CAD platform.

Each joint is connected to each of two bodies at a body coordinate system (CS). The constraint geometry determines the joints into which the translator transforms the constraints and controls the position and orientation of the body CSs. Each of these body CSs has an origin and axis triad fixed relative to its body. The translator creates body CSs on the bodies as necessary for connecting joints.

See "Creating a CAD-Based Robot Arm Model" on page 7-39 for an example of configuring constraints.

Avoiding Redundant Constraints

Keep constraints simple and few enough to avoid creating unnecessary joints in your SimMechanics model.

For example, consider three parts, P1, P2, and P3, in an assembly. Suppose P1 and P2 are constrained so that there is no movement possible between them. When you attach P3, you could put one constraint between P3 and P1 and the other between P3 and P2. This leads to a redundant joint in the SimMechanics model, making it harder to understand and troubleshoot than if you created only one constraint. In this example, it is better to create a constraint just between P3 and P2, since P2 cannot move with respect to P1 anyway.

Translating CAD Assemblies into XML

To translate the CAD assembly into a form that SimMechanics can use, you must check and configure the translator settings, then save the assembly through the translator into the Physical Modeling XML format.

Applying the Translator Settings

Open your CAD assembly.

1. Open the SimMechanics settings interface for your CAD platform.

   In this interface, you set the tolerance configurations. See "Configuring Tolerances" on page 7-10.

2. At any time, you can

   - Apply your settings.
   - Cancel your settings. You lose whatever new settings you have entered.

Configuring Tolerances

In the **Settings** area, you can configure one or more of the translation tolerances. Geometrical and numerical differences smaller than the tolerances are treated as zero. The entries implicitly have the same units selected in your CAD platform settings.

- **Linear tolerance** specifies the smallest significant difference in length.
- **Angular tolerance** specifies the smallest significant difference in angle.
- **Relative roundoff** specifies the smallest significant numerical difference.

Creating the XML File

To complete translation of the assembly into a Physical Modeling XML file,

1. Apply your translator settings.

2. If you changed the assembly or any subassemblies, you need to rebuild the assembly and resave it in its native format before exporting it to XML.

3. Export the assembly into XML format. The default name is the same as that of the CAD assembly file. You can change the XML filename and directory at this point.

4. Click **OK**.

   The assembly is saved in the new form as an XML file.

Note To use the exported XML file to automatically generate a SimMechanics model, you need to move or copy the file into a MATLAB working directory. See "Creating Models from Physical Modeling XML" on page 7-14.

Troubleshooting Assembly Export Problems

The CAD-to-SimMechanics translator can encounter difficulties when it attempts to represent your assembly in the XML file as SimMechanics bodies and joints.

Constraint Translation Errors

Constraint translation errors occur when you specify a constraint in your CAD assembly that is not supported for export. Constraints (or mates) supported for a specific CAD platform are listed in its help page (see "Getting Help in the Translator Window" on page 7-12).

If the translator fails to map one or more constraints into joints, it issues one or more error dialogs and logs the errors into a text file. The error dialog indicates the name of this error log file, which is located in the same directory as the exported Physical Modeling XML file. The XML file is generated regardless of constraint translation errors.

The XML file itself contains the same errors, each paired with the corresponding failed joint. The constraints that failed to translate properly are converted instead into welds. These errors reappear as MATLAB command line warnings if you generate a SimMechanics model from the XML file.

Subassembly Configuration Errors

Be sure that you configure your subassemblies' positions and orientations to be consistent with the main assembly's configuration *before* you export the assembly.

If subassemblies are not consistent with their main assembly, the resulting model will not be valid and will encounter simulation errors.

Getting Help in the Translator Window

The translator installation for your platform includes online help files that are independent of the MATLAB Help system.

Getting HTML Help

To obtain online translator help in your CAD platform, select the help option available in the SimMechanics interface for your CAD platform.

Your default Web browser opens and displays an HTML help file specific to your CAD platform.

Getting PDF Help

You can also access a translator guide in PDF format by looking in your CAD platform's CAD-to-SimMechanics translator installation directory. The PDF guide contains the CAD-related sections of this user's guide.

Creating Models from Physical Modeling XML

A CAD assembly can contain enough part and constraint information that you can generate a SimMechanics model consisting of Body and Joint blocks representing the assembly. Once you have exported a CAD assembly into a Physical Modeling XML file, you can generate a SimMechanics block diagram model with this file. Although the generated model will run, you often need to manually simplify and complete it to faithfully represent the original dynamic system.

- “Generating Body-Joint CAD-Based Models” on page 7-14 shows how to import the XML file into SimMechanics using the command `import_physmod`, which constructs the model.
- “Common Features of CAD-Based Models” on page 7-16 explores the characteristics of an automatically generated CAD-based model.
- “Editing and Completing Generated Models” on page 7-16 explains when and how to edit your model and expand it with additional blocks, such as Constraints, Drivers, Actuators, and Sensors.
- “Troubleshooting CAD-Based Models” on page 7-19 points out major model-generation and simulation errors and problems that can arise from CAD-based models, and techniques for solving them.

Note This section assumes that you have SimMechanics installed locally or remotely, and that you have the XML file representing a CAD assembly in your current MATLAB directory.

To generate the SimMechanics model, you do not need the CAD platform from which the XML file was exported or the CAD-to-SimMechanics translator.

Generating Body-Joint CAD-Based Models

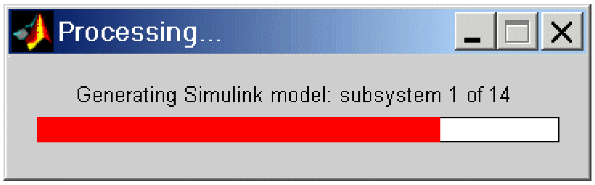
You generate the CAD-based SimMechanics model with the `import_physmod` command.

1 Move or copy the Physical Modeling XML file you want to use into your MATLAB working directory.

2 To start the model generation from an XML file called `cad_assembly.xml`, enter

```
import_physmod('cad_assembly.xml')
```

at the command line. A progress bar appears and is updated as the model is imported.



Generating Simulink model: subsystem 1 of 14

A Simulink model window opens. The model is populated by Bodies and Joints corresponding to the assembly parts and constraints saved in `cad_assembly.xml`. (The corresponding connections and hierarchy are explained in "Exporting CAD Assemblies into Physical Modeling XML" on page 7-6.) The name of the generated model is the name of the original assembly file, regardless of the name chosen for the XML file.

Changing the Appearance of a Generated Model

You can change the appearance of your generated model by using optional import settings. See the `import_physmod` command reference for more details.

For example, entering

```
import_physmod('cad_assembly.xml','FontSize',18)
```

at the command line generates a model with a block label font size of 18 pixels.

Using the Physical Modeling XML File Import Dialog

You can also select a Physical Modeling XML file and generate a model from it through the **Physical Modeling XML File Import** dialog. Open it by entering `import_physmod` at the command line with no arguments.

See the `import_physmod` command reference for complete information about the XML file import dialog.

Common Features of CAD-Based Models

Most of the properties of models that you generate from an XML file are the same as the default properties of all Simulink models. See the Simulink documentation for general information on working with Simulink models.

Models generated from a CAD-based XML file have certain common features:

- Exactly one Ground block and connected Machine Environment block
- Fundamental root, represented by Ground – Root Weld – Root Body
- Subassembly roots, represented by Root Body – Root Weld – Fixed Body
- Joints with degrees of freedom (DoFs) containing the correct joint primitives for the translational and rotational DoFs between any two bodies
- Joints without DoFs, represented by Welds

Joints and Subsystem Hierarchy

Joints that directly connect a subsystem's Body to the next higher level in the hierarchy appear in that next higher level, not within the subsystem. This means that the boundary between subassembly and main assembly in CAD is drawn differently from the boundary between subsystem and top-level system in SimMechanics. This difference does not change the degrees of freedom of the model. They are the same in a CAD assembly and in the SimMechanics model generated from the assembly.

Editing and Completing Generated Models

The `import_physmod` command generates a model containing only Machine Environment, a Ground, Bodies, and Joints. A complete SimMechanics model typically contains other blocks, including Constraints, Drivers, Sensors, and Actuators (from the `mechlib` block library), as well as blocks from Simulink, such as Scopes. Creating a complete SimMechanics model requires inserting and connecting these additional blocks in your generated model. Use the relevant sections of Chapter 4, "Modeling Mechanical Systems" to learn how to use these blocks.

Note When you first generate a model from Physical Modeling XML, you might want to save this original CAD-based model before you create later versions by eliminating unnecessary blocks and adding new ones.

Deleting Unnecessary Blocks

The constraint-to-joint mapping creates whatever blocks are needed to correctly isolate the degrees of freedom (DoFs). In some models, certain generated blocks are not needed for simulation. You can delete these unnecessary blocks to simplify your model without affecting its dynamics, as long as you reconnect the remaining blocks properly.

In particular, you can simplify the fundamental root, Ground – Root Weld – Root Body, in certain cases. (The same holds for subassembly roots. However, the corresponding subsystem's Root Body takes the place of Ground, and the subsystem's Fixed Body plays the role of Root Body.)

- If the Root Body is connected to the rest of the block diagram by a single Joint, you can delete the Root Weld – Root Body blocks and reconnect the Joint directly to Ground.
- If the Root Body is connected to the rest of the block diagram by multiple Joints, you cannot delete the Root Weld – Root Body sequence because a Ground can connect to only one Joint, and the generated model contains only one Ground. You have two alternatives in this case.
  - Determine where the Root Body coordinate systems (CSs) are. Delete the Root Weld – Root Body sequence. Then manually add more Grounds at the spatial points representing each of the deleted Root Body CS origins. Reconnect each Joint to its corresponding Ground at the correct spatial point.
  - Leave the Root Weld – Root Body sequence as it is.

See "Building a CAD Assembly for SimMechanics" on page 7-6.

Caution Not all Welds are redundant. Some are needed to connect a Body that would otherwise be isolated from the rest of the machine. Such Welds represent rigid connections between distinct assembly parts.

If you want to reduce or eliminate such Welds, return to your original assembly and reexport it with rigid subassemblies. See "Improving Your Assembly with Subassemblies" on page 7-8.

Constraining and Driving Degrees of Freedom

Constraints reduce the number of independent degrees of freedom in a machine by preventing certain movements. Drivers affect DoFs, not by eliminating them completely, but by forcing their motions to follow an external time-dependent signal. A constrained or driven DoF is not free to respond to forces and torques independently.

These sections discuss how to add constraints and drivers to your model:

- "Modeling Constraints and Drivers" on page 4-38
- "Actuating a Driver" on page 4-57

Note You insert these constraints into the SimMechanics model as blocks, and they act in addition to the part constraints in the original CAD assembly.

Actuating Bodies and Joints with Motions and Forces

SimMechanics gives you the ability to actuate bodies and joints in your models with external forces and motions, and to set up internal forces between bodies. You can also vary the mass or inertia (or both) of a body in time. Consult these sections for more details:

- "Actuating a Body" on page 4-46
- "Varying a Body's Mass and Inertia Tensor" on page 4-49
- "Actuating a Joint" on page 4-52

- "Modeling Force Elements" on page 4-69

Setting the Model's Initial Conditions

When you import a CAD assembly into SimMechanics, the XML file determines the initial geometric configuration of the model's bodies. By default, the initial velocities of the bodies are zero.

If you want to change the initial positions and/or velocities of the bodies in your model to be different from the CAD-determined configuration, you need to add initial condition actuators, as discussed in "Specifying Initial Positions and Velocities" on page 4-57.

Sensing Forces and Motions

To detect the motions of bodies and joints and measure the forces acting on or through them, you can add sensors to your model. See "Modeling Sensors" on page 4-63.

Satisfying General SimMechanics Requirements

You can find the general instructions for building valid SimMechanics models in these sections.

- "Modeling Machines" on page 4-3
- "Checking Model Validity" on page 4-74

Troubleshooting CAD-Based Models

Problems might arise in your generated SimMechanics model if your CAD assembly was not constructed or exported properly.

General Guidelines

- See "Building a CAD Assembly for SimMechanics" on page 7-6 instructions on constructing a CAD assembly specifically for SimMechanics.
- See "Translating CAD Assemblies into XML" on page 7-10 for details about exporting a CAD assembly.

Troubleshooting Errors During Model Generation

Errors in the Physical Modeling XML file appear as warnings at the MATLAB command line during model generation.

These warnings arise from CAD constraint translation errors encountered when the XML file was originally exported. Such errors occur when the translator fails to map one or more CAD constraints, which restrict the degrees of freedom between parts, into their corresponding SimMechanics joints. The translator warns you at the export step if such errors occur.

The failed Joint appears in your generated model as a Weld. You can fix such an error in two ways:

- In the generated model, manually replace the Weld with the proper Joints.
- Return to the original CAD assembly, reconfigure the constraints, and export it again.

Consult “Constraint Translation Errors” on page 7-12 for more about export errors.

Troubleshooting Model Dynamics Errors

Certain problems with CAD-based models appear only when you run the model.

- You must “fix” at least one part in every CAD subassembly by mating it to the subassembly root. Otherwise, the massless root body is dynamically active and experiences infinite acceleration when forces or torques are applied to it. See these sections for more details:
  - “Roots and Root Bodies” on page 7-7
  - “Subassemblies and Hierarchies” on page 7-7
  - “Improving Your Assembly with Subassemblies” on page 7-8
- If you find constraints are violated while SimMechanics is running your model, try the following:
  - Examine your original CAD assembly for redundant constraints.

  - Check and possibly increase the assembly tolerances at the original CAD export step.
  - Check and possibly increase the translated model's assembly tolerances in the machine's Machine Environment block, in the **Parameters** pane.
  - On the **Constraints** panel in the Machine Environment dialog, select the **Use robust singularity handling** check box.
- Never *decrease* assembly tolerances in a CAD-based SimMechanics model. Instead, decrease the assembly tolerances at the original CAD export step.

Troubleshooting SimMechanics and Simulink Problems

You might also encounter general Simulink or SimMechanics problems while running your model.

- For problems specific to SimMechanics, see "Troubleshooting Simulation Errors" on page 5-17.
- For general Simulink problems, consult the Simulink documentation.

Overview of CAD Translation Examples

Note The following CAD assembly and translation examples are based on SolidWorks® and the SolidWorks-to-SimMechanics translator.

These examples can be recreated with other CAD platforms. The assembly, geometric, kinematic, and part details differ from platform to platform. In SolidWorks, constraints on CAD parts are called *mates*.

The following sections of this chapter illustrate the stages of converting CAD assemblies into SimMechanics models in Simulink. Each section presents an example or set of examples based on a specific machine type represented in CAD and demonstrates how to apply in specific situations the procedures outlined in the chapter's preceding sections. The SimMechanics models in each section are generated with default settings.

- "Exporting a CAD Part" on page 7-23 presents the simplest case, translating an assembly with a single part or body.
- "Designing and Exporting CAD Constraints" on page 7-26 examines in detail how you translate assemblies with constrained parts into bodies with degrees of freedom represented by SimMechanics Bodies and Joints.
- "Creating a CAD-Based Robot Arm Model" on page 7-39 illustrates the translation of an assembly representing a simple machine, including subassembly-subsystem hierarchy. This section also introduces post-translation additions to the model.
- "Modeling a Stewart Platform in CAD" on page 7-46 illustrates the translation of a moderately complex machine assembly with many degrees of freedom, subassembly-subsystem hierarchy, and visualization of its motion.

Exporting a CAD Part

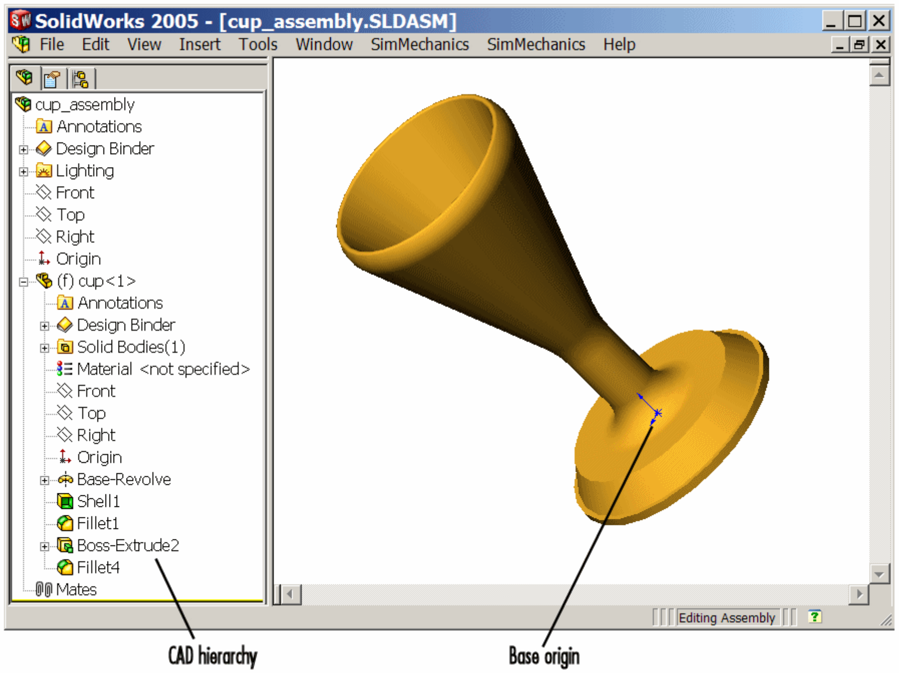
In this example, you export an assembly with one part and no constraints. Locate the two example CAD files:

- The full assembly file, `cup_assembly.`*`ASSEMBLYFILETYPE`*
- The part, a cup, in a file called `cup.`*`PARTFILETYPE`*

Although it has only one part, you must export the full assembly into XML, not just the cup part.

Viewing the CAD Assembly

Open the cup assembly file in your CAD platform and check its geometry and mass properties.



Cup Assembly in a CAD Platform

Property	**Value**
Volume	0.0001 cubic meters (m^3)
Surface area	0.0381 square meters (m^2)
Density	3.0 grams/cm^3 = 3000 kg/m^3
Mass	0.2906 kilograms (kg)
Principal moments of inertia at the center of gravity	I_x = 0.00015, I_y = 0.00067, I_z = 0.00067 kg-m^2

The inertia tensor is computed with the origin at the center of gravity and the coordinate axes aligned with base-origin axes, indicated in the figure Cup Assembly in a CAD Platform on page 7-23. The *x*-axis is the cup's axis of symmetry, and the *y*- and *z*-axes point across the cup.

Exporting the CAD Assembly

Now export the assembly into Physical Modeling XML format. The XML file `cup_assembly.xml` appears in your working CAD directory.

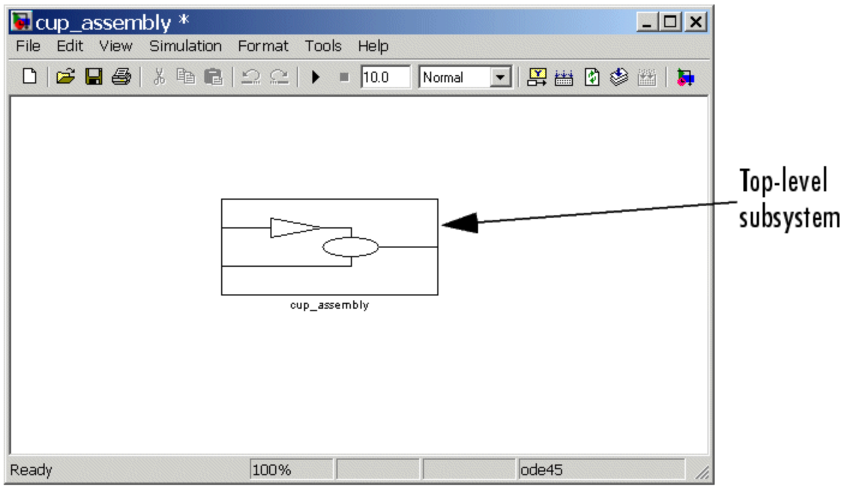
See "Exporting CAD Assemblies into Physical Modeling XML" on page 7-6 for a discussion of converting CAD assemblies into XML format.

Generating the SimMechanics Model

See "Creating Models from Physical Modeling XML" on page 7-14 for complete instructions on how to generate models from XML files.

1 Move or copy the exported XML file into a MATLAB working directory to generate a SimMechanics model from the file.

2 Generate the model from `cup_assembly.xml` with the `import_physmod` command.



Once you generate the SimMechanics model, it has six blocks, a combination

Machine Environment – Ground – Weld – Root Body – Weld – Cup

inside a subsystem, called `cup_assembly`, representing the entire assembly.

- The Root Body is the nondynamical zero-mass/zero-inertia body inserted between ground and the cup.
- The second joint is a Weld because the original CAD assembly has no degrees of freedom.

Deleting the Root Body and one of the Welds does not physically change the model, as long as you reconnect the remaining blocks.

Designing and Exporting CAD Constraints

In “Exporting a CAD Part” on page 7-23, you create and export an assembly composed of a single part. The CAD-to-SimMechanics translator converts the part into a body, with a mass, an inertia tensor, and body coordinate systems (CSs). Because there are no other parts in that CAD assembly, the SimMechanics body is welded to ground and has no degrees of freedom (DoFs). This lack of DoFs is not realistic for most assemblies.

Restricting Degrees of Freedom with Constraints

CAD platforms normally assume that two parts with no constraints between them have the complete six relative DoFs possessed by any rigid body relative to another body. You restrict the DoFs between parts by connecting them with constraints in the CAD assembly. Constraints restrict relative body motion and reduce the number of relative DoFs between body pairs. There is always one assembly part welded to ground.

Part-Constraint Assembly Examples in This Section

This section presents a set of complete CAD assemblies with both parts and constraints. Each example assembly consists of two instances of the same part file, representing two identical cubes.

- “Common Steps for Generating the Two-Part Models” on page 7-27 presents the essential steps for generating these models.
- “Block Structure of the Two-Part Models” on page 7-28 discusses the common structure of all the generated models in this section.

In different assemblies, the two cubes are constrained with different constraint combinations to create different relative DoFs between the cubes. You can typically represent a set of DoFs with a large number of different constraint combinations. Each constraint combination here, in general, is *not* the unique way to create the corresponding set of DoFs.

- “Modeling a Six-DoF Joint” on page 7-29 assembles the two cubes with no constraints so that the cubes have the full six degrees of freedom relative to one another.

- "Modeling a Prismatic Joint" on page 7-30 shows how to constrain the two cubes in two different ways so as to produce the same result, a single prismatic (translational) DoF between them.
- "Modeling a Revolute Joint" on page 7-34 constrains the two cubes so as to allow only a single revolute (rotational) DoF between them.
- "Modeling an Inplane Joint" on page 7-35 constrains the two cubes so as to allow two prismatic (translational) DoFs between them.
- "Modeling a Spherical-Spherical Massless Connector" on page 7-36 constrains the cubes so as to allow relative spherical joint motion, with the two cubes separated by a constant nonzero distance.

Locating the Example Assembly Files

Locate the CAD assembly files used as examples for this section. The assemblies have the generic name `<assembly-name>.`*`ASSEMBLYFILETYPE`*. The cube part is contained in `magic_cube.`*`PARTFILETYPE`*.

Assembly Name	Assembly Configuration
`sixDOF`	Two cubes with no constraints
`prismatic1`	Two cubes with planar and cylindrical constraints
`prismatic2`	Two cubes with planar constraints
`revolute`	Two cubes with planar and cylindrical constraints
`inplane`	Two cubes with planar constraints
`spherical_spherical_ massless_connector`	Two cubes with a distance constraint

Common Steps for Generating the Two-Part Models

The steps for exporting a two-part assembly and generating SimMechanics models based on it are essentially the same for all the examples of this section.

Viewing and Exporting an Assembly

To see a two-part assembly and export it into Physical Modeling XML, 1 Open the assembly `<assembly-name>.ASSEMBLYFILETYPE`. The two parts are `magic_cube-1` and `magic_cube-2`.

   In the CAD hierarchy, note any constraints imposed on the parts. These constraints define the relative DoFs between the parts.

2 Now translate this CAD assembly into Physical Modeling XML format. The XML file is saved in your current working CAD directory.

   See "Exporting CAD Assemblies into Physical Modeling XML" on page 7-6 for a discussion of converting CAD assemblies into XML format.

Generating a Model

Now you can generate a model based on this assembly.

1 Move or copy the XML file to a working MATLAB directory. Then open MATLAB in that directory.

2 At the command line, enter `import_physmod('<assembly_name>')`. SimMechanics automatically generates a model, `<assembly_name>.mdl`, based on `<assembly_name>.xml`.

   The entire assembly is translated into a subsystem, also called `<assembly_name>`, within the model.

3 Open the subsystem. The blocks are arranged in the common structure described in "Block Structure of the Two-Part Models" on page 7-28. A set of Joints represents the DoFs between the two cubes.

Block Structure of the Two-Part Models

All the models that you generate in this section from the example CAD assemblies have a common structure because each assembly has a fundamental root and two moving parts. Each model has eight blocks.

- *The assembly's fundamental root.* As in any generated CAD-based model, the four-block combination Machine Environment – RootGround – RootWeld – RootPart represents the assembly's fundamental root. The RootPart is a nonmoving, zero-mass/zero-inertia body.

For more about roots, see "Building a CAD Assembly for SimMechanics" on page 7-6 and "Common Features of CAD-Based Models" on page 7-16.

- *The moving bodies.* The bodies representing the assembly's parts are `magic_cube-1` and `magic_cube-2`.
- *The joints.* In all the models, the first cube is connected by a Weld to RootPart and cannot move. The second cube is connected to RootPart by a Joint that represents the appropriate degrees of freedom (DoFs).

  Depending on the DoFs in question in a particular assembly, the translator configures the Joint to represent different DoFs with combinations of prismatic, revolute, and spherical primitives. The second cube can move with respect to the first through the DoFs represented by the Joint.

Some of the blocks in the generated models are redundant. You can manually edit and simplify the models without changing their physical properties. For more about manual editing of generated models, see "Editing and Completing Generated Models" on page 7-16.

Modeling a Six-DoF Joint

The simplest assembly with two parts has no constraints between the parts. The parts can move with respect to one another with all six degrees of freedom (DoFs).

Exporting the Assembly

To see and export such an assembly,

1. Open the assembly `sixDOF.`*`ASSEMBLYFILETYPE`*.

   Note, in the CAD hierarchy, that the constraints (Mates) node has no entries. Therefore, relative to one another, the cubes are unconstrained in their motion and have six relative DoFs.

2. Translate this CAD assembly into `sixDOF.xml`.

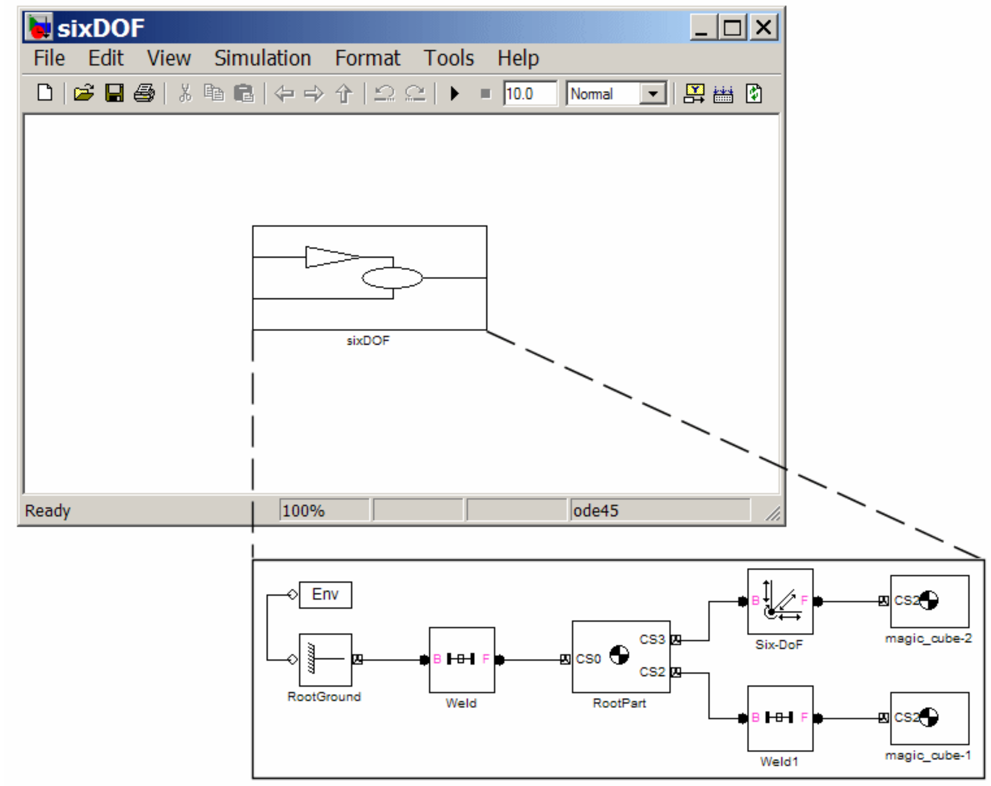
Generating the Model

Now generate a model based on this assembly.

1 At the MATLAB command line, enter `import_physmod('sixDOF')`. SimMechanics automatically generates a model, `sixDOF.mdl`.

2 Open the subsystem `sixDOF`. There are eight blocks.

   The Six-DoF Joint represents the six DoFs between the two cubes with one spherical and three prismatic primitives.



Modeling a Prismatic Joint

In the following two assemblies, the two cubes are constrained to have only a single translational degree of freedom (DoF) between them. These assemblies illustrate two ways to accomplish this; you can experiment with other constraints to find more. In the translated SimMechanics models, this single DoF is a prismatic joint.

Prismatic as a Planar Constraint and a Cylindrical Constraint

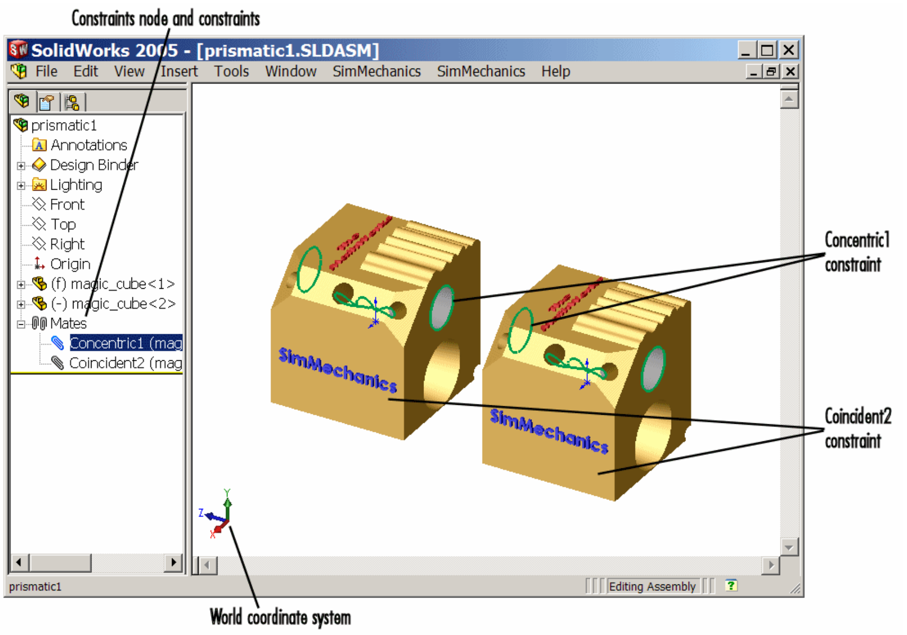
To see the first way of constraining the DoFs to produce a prismatic joint,

1 Open the assembly file `prismatic1.`*`ASSEMBLYFILETYPE`* and examine the CAD hierarchy.

2 Locate and expand the constraints (Mates) node. There are two constraints on the two cubes.

   - Highlight the first constraint, Concentric1. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes only to slide along and rotate about the *z*-axis running through the center of the parallel and concentric upper holes of each cube.
   - Highlight the second constraint, Coincident2. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along the *y*-*z* plane, with the two sides marked "SimMechanics" sharing a common plane, representing two translational DoFs. It also allows the two cubes to rotate about the *x*-axis. The cubes are not allowed to rotate about any other axis, or to translate perpendicular to the *y*-*z* plane.

These two constraints mean that the two cubes can only slide along the *z*-axis common to the two upper concentric holes. The second constraint prevents rotation about this axis, leaving the whole assembly with only one translational DoF.



Planar and Cylindrical Constraints on Two Cubes

Prismatic as Two Orthogonal Planar Constraints

To see the second way of constraining the DoFs to produce a prismatic joint,

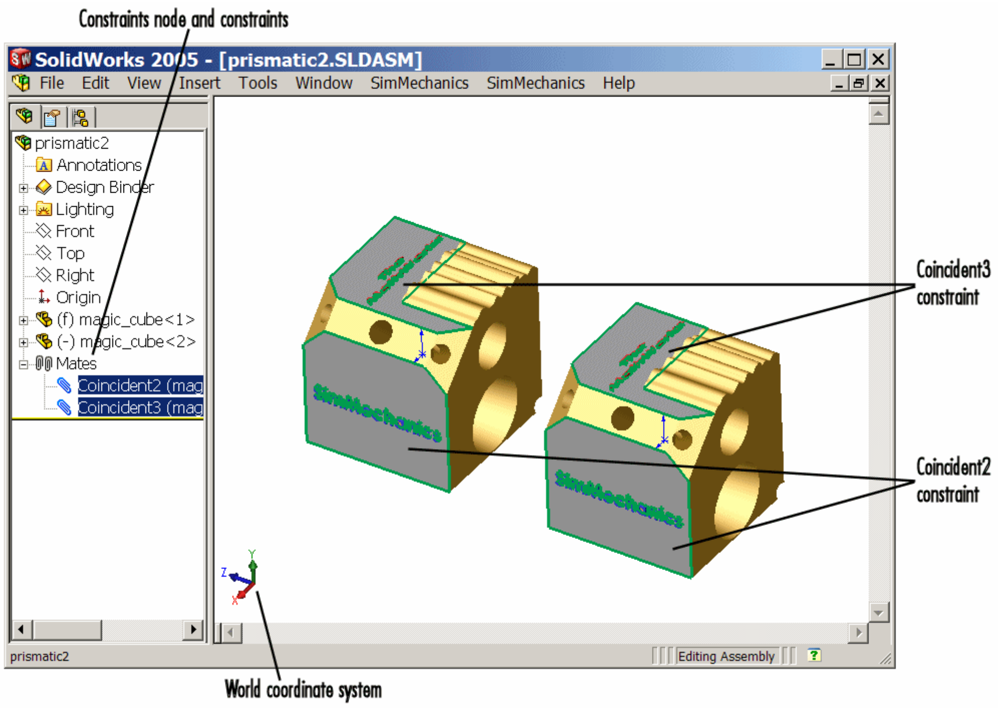
1 Open the assembly file `prismatic2.`*`ASSEMBLYFILETYPE`* and examine the CAD hierarchy.

2 Locate and expand the constraints (Mates) node. There are two constraints on the two cubes.

   - Highlight the first constraint, Coincident2. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along the *y*-*z* plane, with the two sides marked “SimMechanics” sharing a common plane, representing two translational DoFs. It also allows the two cubes to rotate about the *x*-axis. The cubes are not allowed to rotate about any other axis, or to translate perpendicular to the *y*-*z* plane.

- Highlight the second constraint, Coincident3. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along the *x*-*z* plane, with the two sides marked "The MathWorks" sharing a common plane, representing two translational DoFs. It also allows the two cubes to rotate about the *y*-axis. The cubes are not allowed to rotate about any other axis, or to translate perpendicular to the *x*-*z* plane.

These two constraints mean that the two cubes can only slide along the *z*-axis common to the two planes *y*-*z* and *x*-*z*, leaving the whole assembly with only one translational DoF.



Two Planar Constraints on Two Cubes

Exporting the Assemblies and Generating SimMechanics Models

To create models from the assemblies,

1 Export the two assemblies into the XML files `prismatic1.xml` and `prismatic2.xml`.

2 Copy or move them to a MATLAB working directory. At the MATLAB command line, generate SimMechanics models using `import_physmod`.

In both models, the assemblies are translated into the `prismatic1` and `prismatic2` subsystems, respectively. Each subsystem has eight blocks. The Prismatic Joint represents the single translational DoF between the two cubes with one prismatic primitive along the *z*-axis.

Modeling a Revolute Joint

In the following assembly, the two cubes are constrained to have only a single rotational degree of freedom (DoF) between them. In the translated SimMechanics model, this single DoF is a revolute joint.

Viewing the Assembly

To see an assembly with one rotational DoF,

1 Open the assembly file `revolute.`*`ASSEMBLYFILETYPE`* and examine the CAD hierarchy.

2 Locate and expand the constraints (Mates) node. There are two constraints on the two cubes.

   - Highlight the first constraint, Concentric1. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along and rotate about the *z*-axis running through the center of the parallel and concentric upper holes of each cube.
   - Highlight the second constraint, Coincident1. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along the *x*-*y* plane, with the parallel sides sharing a common plane. It also allows the two cubes to rotate about the *z*-axis. The cubes are not

allowed to rotate about any other axis, or to translate perpendicular to the *x*-*y* plane.

These two constraints mean that the two cubes can only rotate about the *z*-axis orthogonal to the *x*-*y* plane, leaving the whole assembly with only one rotational DoF.

Exporting the Assembly and Generating the Model

Now generate a model based on this assembly.

1. Export the assembly as `revolute.xml`. Copy or move it to a MATLAB working directory.

2. At the MATLAB command line, generate a SimMechanics model using `import_physmod`.

The assembly is translated into an eight-block subsystem called `revolute`. The Revolute Joint represents the single rotational DoF between the two cubes with one revolute primitive about the *z*-axis.

Modeling an Inplane Joint

In the following assembly, the two cubes are constrained to have only two translational degrees of freedom (DoFs) between them. In the translated SimMechanics model, these two DoFs are two prismatic joints.

Viewing the Assembly

To see an assembly with two translational DoFs,

1. Open the assembly file `inplane.`*`ASSEMBLYFILETYPE`* and examine the CAD hierarchy.

2. Locate and expand the constraints (Mates) node. There are two constraints on the two cubes.

   - Highlight the first constraint, Coincident2. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide along the *y*-*z* plane, with the two sides marked “SimMechanics” sharing a common plane. It also allows the two cubes to rotate about the *x*-axis. The cubes are not allowed to rotate about any other axis, or to translate perpendicular to the *y-z* plane.

- Highlight the second constraint, Parallel1. The constraint geometry is highlighted in the assembly. This constraint allows the two cubes to slide parallel to the *x-z* plane, with the two sides marked "The MathWorks" parallel but not necessarily in the same plane. It also allows the two cubes to translate perpendicular to the *x-z* plane and to rotate about the *y*-axis. The cubes are not allowed to rotate about any other axis.

These two constraints mean that the two cubes can only slide in the *y-z* plane, leaving the whole assembly with only two translational DoFs.

Exporting the Assembly and Generating the Model

Now generate a model based on this assembly.

1. Export the assembly as `inplane.xml`. Copy or move it to a MATLAB working directory.

2. At the MATLAB command line, generate a SimMechanics model using `import_physmod`.

The assembly is translated into a subsystem, `inplane`, having eight blocks. The In-Plane Joint represents the two translational DoFs between the two cubes with two prismatic primitives, along the *y*-axis and the *z*-axis.

Modeling a Spherical-Spherical Massless Connector

In the following assembly, the two cubes are constrained to have six rotational degrees of freedom (DoFs) between them, represented by two spherical primitives. The spherical primitives pivot independently about two pivot points at a fixed relative distance. In the translated SimMechanics model, a spherical-spherical massless connector represents these six DoFs.

Viewing the Assembly

To see an assembly with three rotational DoFs separated from three other rotational DoFs,

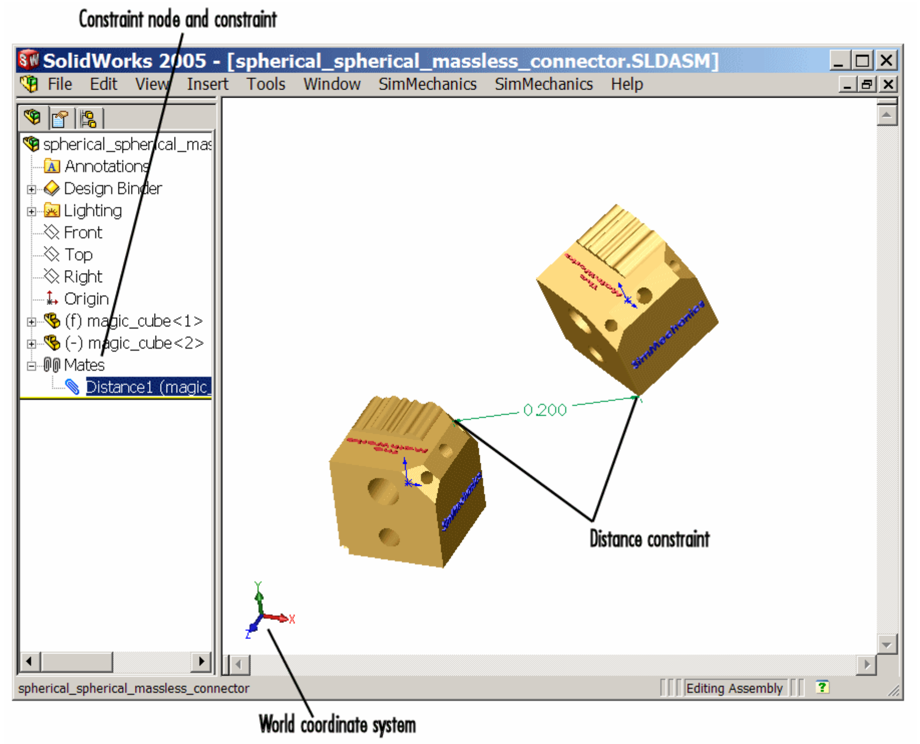
1. Open the assembly file

```
spherical_spherical_massless_connector.ASSEMBLYFILETYPE
```

and examine the CAD hierarchy.

2 Locate and expand the constraints (Mates) node. There is one constraint on the two cubes.

Highlight this constraint, Distance1. The two spherical pivot points are highlighted as small red or green squares, one on each cube. These points are the endpoints of the rigid massless connector. The cubes can move such that the distance between these two points (the length of the massless connector) does not change. The constraint allows the two cubes to pivot independently about their connector endpoints.


Constraint node and constraint
SolidWorks 2005 - [spherical_spherical_massless_connector.SLDASM]
File Edit View Insert Tools Window SimMechanics SimMechanics Help
spherical_spherical_mas
Annotations
Design Binder
Lighting
Front
Top
Right
Origin
(f) magic_cube<1>
(-) magic_cube<2>
Mates
Distance1 (magic
0.200
Distance constraint
spherical_spherical_massless_connector
Editing Assembly
World coordinate system

Distance Constraint on Two Cubes

Exporting the Assembly and Generating the Model

Now generate a model based on this assembly.

1 Export the assembly as

   ```
   spherical_spherical_massless_connector.xml
   ```

   Copy or move it to a MATLAB working directory.

2 At the MATLAB command line, generate a SimMechanics model using `import_physmod`.

The assembly is translated into an eight-block subsystem called `spherical_spherical_massless_connector`, arranged in the common structure described in “Block Structure of the Two-Part Models” on page 7-28.

The Spherical-Spherical massless connector Joint block represents the two spherical primitives, each with three rotational DoFs, independently pivoting at each end of the massless, rigid connector connecting the two cubes.

Creating a CAD-Based Robot Arm Model

The example of this section is based on a more complex CAD assembly, a robot arm. It includes multiple parts, multiple constraints, and a subassembly.

Locate the 11 CAD files for the robot arm are. They are

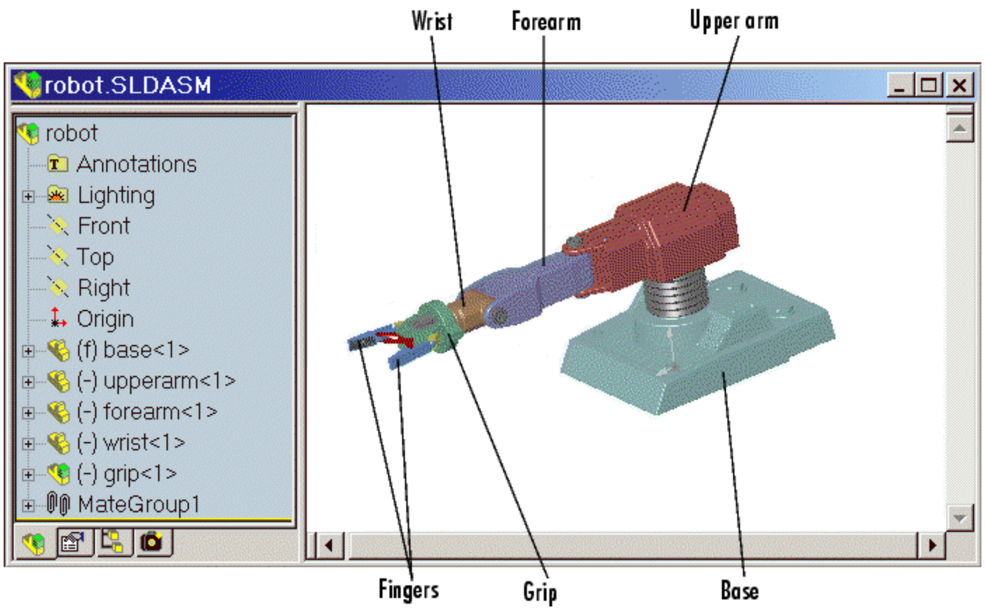
Filename	CAD Filetype
`robot.`*`ASSEMBLYFILETYPE`*	Assembly
`grip.`*`ASSEMBLYFILETYPE`*	Subassembly (flexible)

`base.`*`PARTFILETYPE`*
`forearm.`*`PARTFILETYPE`*
`upperarm.`*`PARTFILETYPE`*
`wrist.`*`PARTFILETYPE`*
	Parts (main assembly)

`fingertips.`*`PARTFILETYPE`* (twice)
`firstfingerlink.`*`PARTFILETYPE`*
`firstfingerlinkL.`*`PARTFILETYPE`*
`metacarpal.`*`PARTFILETYPE`*
`secondfingerlink.`*`PARTFILETYPE`* (twice)
	Parts (subassembly)

Viewing the Robot Arm Assembly

Open the assembly file for the whole robot.



Robot Arm Assembly in a CAD Platform

Then examine the CAD hierarchy:

- Five of the part files are grouped into the subassembly `grip`. The subassembly uses two instances each of `fingertips` and `secondfingerlink`.
- The subassembly has its own group of 18 constraints, MateGroup1.

  Two constraints, Angle1 and Angle2, are not active. If they were, they would lock the grip fingers into the open position. Here, each grip finger can move separately.
- The other four part files are separate and grouped into the main assembly.
- The main assembly has its own MateGroup1, consisting of seven constraints.

The whole assembly has eight DoFs. The `grip` subassembly alone contains two, allowing each finger to open and close separately. The main assembly has six DoFs:

- The upper arm can move relative to the base by pitching, yawing, and rolling (three DoFs).
- The forearm can yaw relative to the upper arm (one DoF).
- The wrist can pitch relative to the forearm (one DoF).
- The grip can rotate about its symmetry axis (one DoF).

Exporting the Robot Arm Assembly

Apply any changes you want to the assembly configuration or settings. If you change the assembly or any subassemblies, you need to rebuild the assembly before exporting it to XML.

Now export the assembly into Physical Modeling XML. The XML file `robot.xml` appears in your working CAD directory.

Generating and Completing the Robot Arm Model

Now generate a model for a robot arm based on the file `robot.xml`. You can use this preconfigured demo file or export your own version of the XML file from the robot arm CAD assembly. In either case, copy or move the XML file to your MATLAB working directory.

See “Creating Models from Physical Modeling XML” on page 7-14 to learn more about how to generate models from XML files.

Generating the Initial Model

The preconfigured `robot.xml` file is located in the MATLAB directory

```
toolbox/physmod/mech/mechdemos/
```

1 Generate the model by entering

```
import_physmod('robot')
```

at the command line. The status bar appears and indicates the progress of model generation.

A model window, named **robot**, opens and is populated with blocks.

2 Save this initial body-joint model as robot, and note these properties:

- The whole robot arm assembly is contained in the subsystem `robot`.
- The top level of the assembly has 13 blocks and the `grip-1` subsystem.
- The `grip-1` subsystem has 18 blocks.

  The original robot arm assembly has eight DoFs, with two in the grip subassembly and six at the top level. These translate into eight DoFs in the SimMechanics model:

  - Six DoFs occur at the top level. These include the upper arm relative to the base, the forearm relative to the upper arm, the wrist relative to the forearm, and the grip relative to the wrist.
  - Two DoFs occur in the subsystem. These are the rotational DoFs of the two grip fingers.

    There are eight revolute primitives in the subsystem. They occur in two closed loops as two independent four-bar mechanisms. Each four-bar mechanism actually has only one independent DoF because each four-bar loop closes on itself. (See "Four Bar Mechanism" on page 2-36 and "Counting Degrees of Freedom" on page 4-77.)

Obtaining Simulink and Additional SimMechanics Blocks

To modify and extend the robot arm model, you need blocks from the SimMechanics and Simulink block libraries. Open these libraries by entering `mechlib` and `simulink`, respectively, at the command line.

You can also open the Simulink library from the MATLAB window menu or toolbar.

Editing the Bodies

Some of the bodies in the generated robot arm model are redundant. You can remove them without affecting the model's dynamics, as long as you properly reconnect the remaining blocks.

- At the top level, the SimMechanics_RootPart block is a zero-mass, zero-inertia Root Body. You can delete it, along with the connected Weld1 block, then reconnect the Root Ground to the base-1 block through the Weld block.
- In the `grip-1` subsystem, you can delete the grip-1 (Root Body) block and the connected Weld block because they are unnecessary. You can also delete the associated body coordinate system on the metacarpal-1 Body block.

See the reference page for more details about the Body block.

Editing the Joints

The non-Weld Joint blocks, those that carry DoFs, are Revolute and Spherical Joints configured with the proper primitives to represent the original CAD assembly's DoFs.

- The first non-Weld Joint encountered as you move away from the Ground block is a Spherical, representing three DoFs.
- Each Revolute block contains a single revolute primitive, representing one rotational DoF.

Save this intermediate model as robot2.

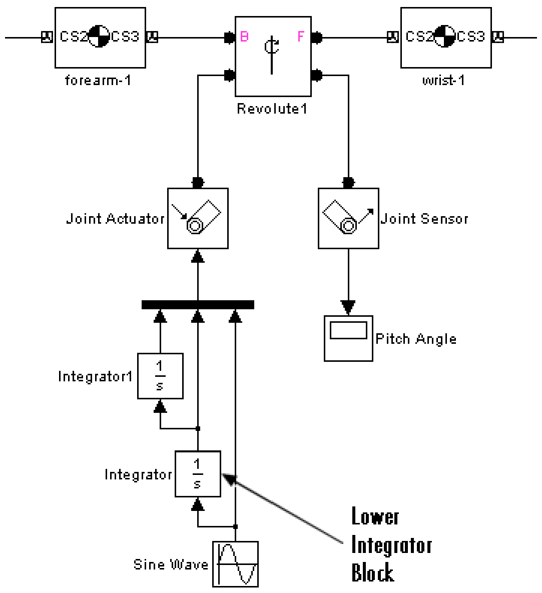
Adding an Actuator and a Sensor

You can motion-actuate the wrist relative to the forearm.

1. Double-click the Revolute Joint that connects the forearm-1 and wrist-1 Body blocks. Change the **Number of sensor/actuator ports** to `2`.
2. Click **OK**. Two new ports appear on the Joint.
3. From the SimMechanics Sensors & Actuators library, insert and attach a Joint Actuator and a Joint Sensor to these new ports.
4. Configure the Joint Actuator to accept motion signals. Be sure the angular units are `deg` (degrees).

5 From the Simulink library, insert a Sine Wave, a Mux, two Integrator blocks, and one Scope block. Connect them to the previous blocks as shown in the following figure. Rename the Scope block to Pitch Angle.

Consult the Simulink documentation for more about these Simulink blocks.



6 In the Sine Wave block, set the **Amplitude** to `60*pi*pi` and the **Frequency** to `60`. Leave all other defaults unchanged.

7 In the lower Integrator block, set **Initial condition** to `-60*pi`. Leave all other defaults unchanged.

Configuring Tolerances

The original robot arm CAD assembly requires looser tolerances than the SimMechanics defaults, and its motion can lead to singularities. To avoid simulation errors or slowdown, you need to reconfigure the assembly tolerances and constraint solver.

1 Open the Machine Environment block.

2 On the **Parameters** panel, reset the **Linear assembly tolerance** to `1e-2 m` (meters) and the **Angular assembly tolerance** to `1e-1 rad` (radians).

3 On the **Constraints** panel, select the **Use robust singularity handling** check box. Leave all other defaults. Click **OK**.

4 Resave your finished model as robot3.

Simulating and Observing the Robot Arm Motion

You can now run `robot3` and examine its motion.

To use the motion sensor,

1 Double-click the Pitch Angle block to open a scope.

2 Click the **Start simulation** button. The scope plot displays a trace of the pitch angle motion.

To visualize the body motions,

1 From the **Simulation** menu, select **Configuration Parameters**, then the **SimMechanics** node.

2 Select **Display machines after updating diagram** and **Show animation during simulation**. Click **OK**.

3 Select **Update Diagram** from the **Edit** menu. The SimMechanics visualization window opens.

4 In the **SimMechanics** menu of the visualization window, select **Machine Display**, then **Ellipsoids**. The display now shows the robot arm's component bodies as ellipsoids.

5 Click the **Start** button. The simulation begins. Observe the robot arm motion in the SimMechanics window.

Modeling a Stewart Platform in CAD

This section introduces a complex computer-aided design (CAD) assembly that models the Stewart platform, a six-degree-of-freedom (DoF) mechanical system used for accurate positioning applications.

Note The Stewart platform assembly of this section is an advanced example of computer-aided design. You should work through the previous examples of this chapter before attempting to work with this assembly.

To learn more about the Stewart platform, see Chapter 9, "Case Studies".

Locate the 45 CAD files for the Stewart platform. The master assembly file is

```
stewart_platform.ASSEMBLYFILETYPE
```

What the Stewart Platform Does

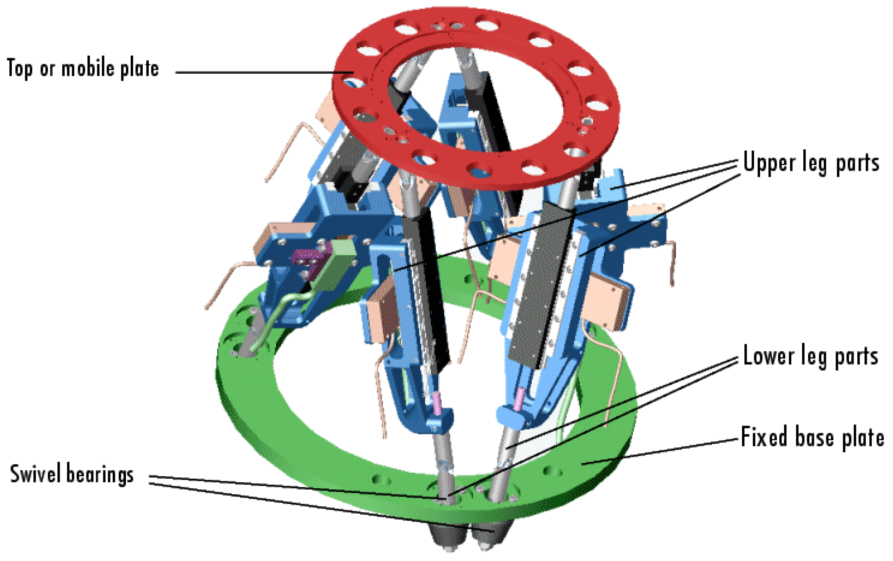
The Stewart platform consists of two plates connected by six mobile and extensible legs. The lower or base plate is immobile. The upper or mobile plate has six degrees of freedom, three rotational and three translational. The platform is highly stable and easy to control.

The platform's six legs each have two parts, an upper and a lower leg, with a piston-like cylindrical DoF between each pair of parts. The legs are connected to the base plate and the top plate by universal joints at each end of each leg. (These universals are not just sets of abstract DoFs. Each also contains a spider-like body, while also having two DoFs.) The upper part of each leg can slide into and out of the lower leg, allowing each leg to be varied in length. The position and orientation of the mobile platform (top plate) varies depending on the lengths to which the six legs are separately adjusted.

Once the top is connected to the legs, the entire Stewart platform assembly has 36 DoFs. Only six DoFs are *independent*, the same as the top plate would have if it were disconnected. You can think of these independent DoFs as the six adjustable leg lengths or as equivalent to the six DoFs of the mobile plate. See “Counting Degrees of Freedom” on page 4-77.

Viewing the Stewart Platform Assembly

Open the master assembly file, `stewart_platform.`*`ASSEMBLYFILETYPE`*. Click the assembly and rotate it to view the top and bottom plates and the legs.



Stewart Platform CAD Assembly

The CAD hierarchy for the Stewart platform contains assemblies for the top and base plates, as well as assemblies for the six legs. All the constraints on the assembly parts are grouped into one group, containing 30 constraints. There are 448 component parts and 38 subassemblies, which you can open individually to examine the separate parts.

The base plate is about 24 centimeters (cm) in diameter, the top plate about 16.5 cm. When centered and oriented flat, the top plate is about 20 cm above the base. The assembly models the platform material as aluminum (about 2.7 grams per cubic cm).

Exporting the Stewart Platform Assembly

Apply any changes you want to the assembly configuration or settings. If you change the assembly or any subassemblies, you need to rebuild the assembly before exporting it to XML.

Now export the assembly into Physical Modeling XML. Because the assembly is so complex, the export process takes longer than it does for simpler assemblies. As the export proceeds, the translator highlights various parts and subassemblies. When the highlighting stops, the export is finished.

The exported model appears as the XML file `stewart_platform.xml` in your working CAD directory.

Generating the Stewart Platform Model

Now move or copy the `stewart_platform.xml` file into your working MATLAB directory.

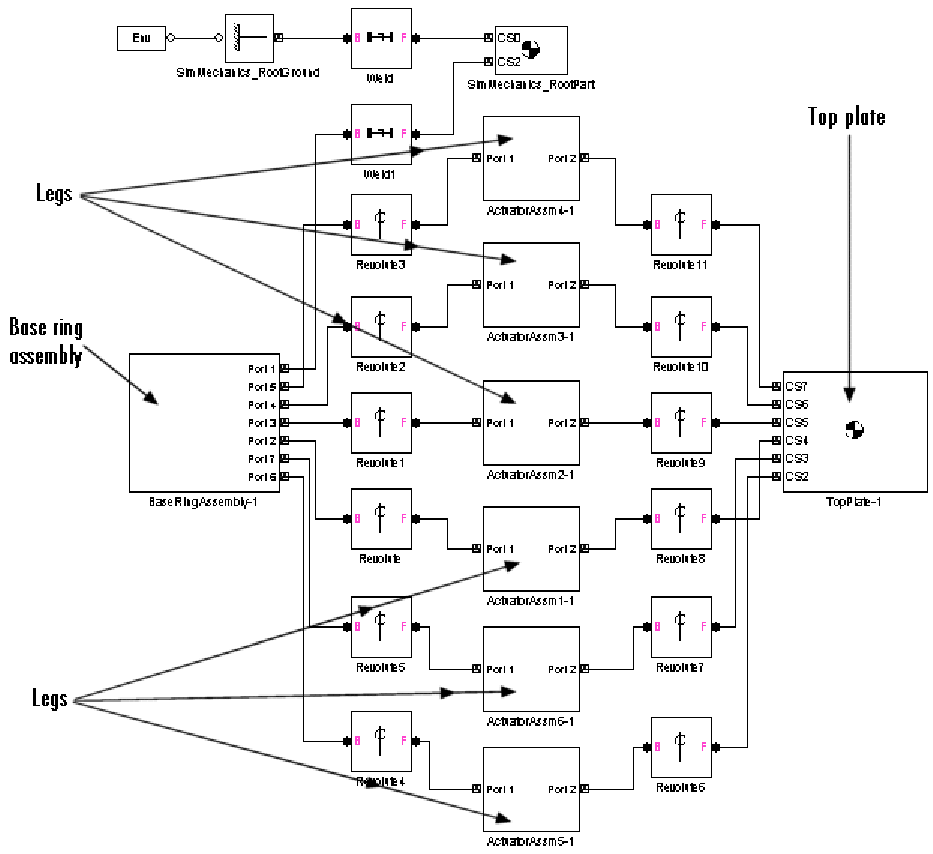
At the MATLAB command line, enter

```
import_physmod('stewart_platform')
```

and wait for SimMechanics to complete the generation of the new Simulink model `stewart_platform`.

Inspecting the Generated Model and Counting Its DoFs

The entire Stewart platform model is contained in a subsystem, also called `stewart_platform`. This subsystem itself contains seven subsystems.



Stewart Platform Model: Base, Legs, and Top Plate

The subsystems correspond to the subassemblies of the original CAD assembly: the base plate and the six platform legs.

- The base plate subassembly `BaseRingAssembly-1` contains six subassemblies, modeling a base swivel bearing for each leg.
- The six leg subassemblies, `ActuatorAssm`, model the upper and lower halves of each leg and represent part of their DoFs.

For each leg, there are six DoFs. Two pairs of revolutes associated with each leg represent the two universal joints connecting each leg to the top and base plates, respectively. Each of these universals has two DoFs.

- At the top level, there are two revolutes, one attached to either end of a leg subassembly, connecting each leg to the base and top plates, respectively.
- Within each leg subassembly, there are two other revolutes, each one connecting the leg to the top and base plates, respectively.

One of the revolutes inside the leg subassembly pairs with one of the revolutes outside the leg assembly to make up a two-DoF universal. These pairs occur twice on each leg, one connecting the leg to the top plate, the other connecting the leg to the base plate.

- Within each leg subassembly, there is one prismatic, representing the leg's freedom to expand or contract along its shaft.
- Within each swivel bearing subassembly, itself located within the base ring assembly, is another revolute representing each leg's freedom to rotate about its shaft.

Each leg has six DoFs. However, the constraints imposed by attaching each leg to fixed points on the base and top plates, respectively, reduce these to one independent DoF for each leg: the freedom to expand or contract along its shaft.

- The rotational DoFs associated with the universals at the attachment points are completely dependent on the leg's prismatic DoF.
- The rotational DoFs associated with the cylindricals in each leg are completely dependent on the universals at the top and bottom of each leg.

Deleting Unnecessary Bodies and Joints

The generated model contains a large number of redundant Root Weld and zero-mass Root Part blocks. You can delete these and not affect the model's dynamics, if you take care to reconnect the remaining bodies properly after deleting each Weld.

Adding Actuators and Sensors

If you want the motion of the platform to be controlled by something other than gravity, you need to add the appropriate Actuators to the model. To quantify the model's motion, you need to make precise measurements with Sensors. You can drive the actuators with external control signals to model an open-loop controller for the Stewart platform. If you introduce feedback from the sensors to the actuators, you can model a closed-loop controller.

You can find out more about using actuator and sensor blocks in Chapter 4, "Modeling Mechanical Systems".

Visualizing the Stewart Platform Motion

Note Find more information about SimMechanics visualization in Chapter 6, "Visualizing and Animating Machines".

Without any external forces acting, apart from gravity, the platform collapses under its own weight. You can verify this by running and visualizing your Stewart platform model.

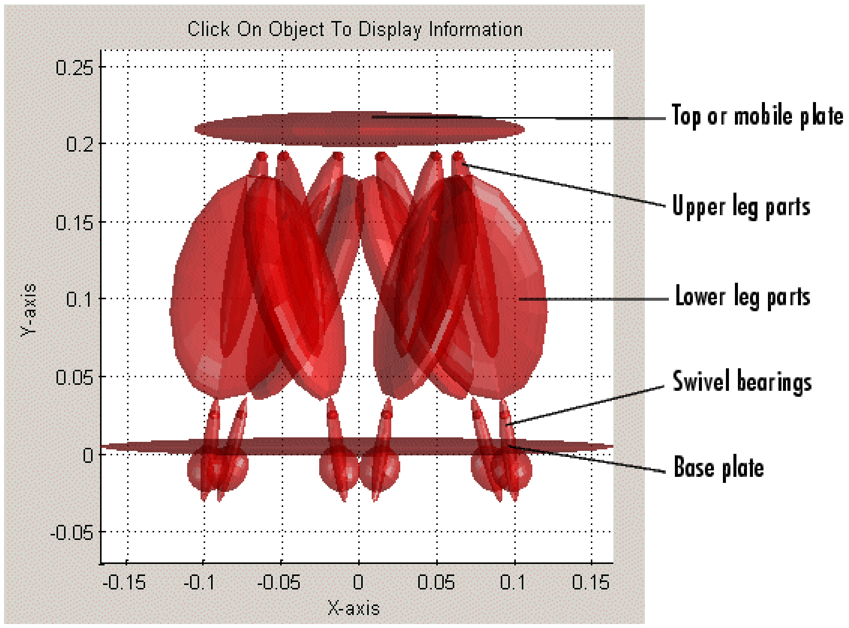
1. From the **Simulation** menu, select **Configuration Parameters**. The **Configuration Parameters** dialog opens. Choose the **SimMechanics** node.

2. Select **Display machines after updating diagram** and **Show animation during simulation**. Click **Apply** or **OK**.

3. From the **Edit** menu, select **Update Diagram**. The SimMechanics visualization window opens with the SimMechanics controls. The window displays the Stewart platform in its initial position.

4. Start the simulation by clicking the **Start** button in the toolbar of either the visualization window or the model window.

The mobile plate falls under its own weight and reaches the base plate in about 0.2 seconds. Because there is nothing to stop the legs or the top plate, the platform continues to collapse: the mobile plate falls below the base plate, and the upper and lower parts of each leg come apart.



SimMechanics Visualization of the CAD-Based Stewart Platform

Analyzing Motion

SimMechanics features analysis modes for studying machine motion beyond the simple forward dynamics integration of forces. This chapter explains how to specify machine motion, then deduce the necessary forces and torques, with the inverse dynamics and kinematic analysis modes. You can also specify a machine steady state and analyze perturbations about any machine trajectory by trimming and linearizing your model, respectively.

Chapter 9, “Case Studies” covers more sophisticated motion analysis and control design techniques applied to more complex systems.

Dynamics of Mechanical Systems

As explained in Chapter 3, “Representing Motion”, kinematics describes the motion of bodies, while *dynamics* explains the motion in terms of forces and torques. By Newton’s laws of motion, the accelerations of the bodies’ positions are directly related to the forces and torques applied to the bodies.

- You can predict accelerations if you are given the applied forces/torques, or relate known accelerations to the forces/torques that cause them, as explained in “Forward and Inverse Dynamics” on page 8-2.
- “Forces and Torques Determine Accelerations” on page 8-3 presents Newton’s laws of dynamics for translational and rotational motion.

The books of Goldstein [2] and José and Saletan [6] present rigid body mechanics in great detail.

Forward and Inverse Dynamics

Dynamical equations such as Newton’s laws of motion relate cause and effect. In mechanics, the cause is a set of forces and torques applied to the bodies of a mechanical system; the effect is the set of resulting motions. Dynamical equations allow you to analyze motion in either direction:

- In *forward dynamics*, you apply a given set of forces/torques to the bodies to produce accelerations. SimMechanics integrates the accelerations twice to yield the velocities and positions as functions of time.

  A set of *initial conditions* is needed to specify the initial positions and velocities and produce a complete solution for the motion. Initial conditions must be checked for consistency with constraints.
- *Inverse dynamics* starts with given motions as functions of time and differentiates them twice to yield the forces and torques needed to produce the given motions. The given motion functions of time must be checked for consistency with constraints.

You can use SimMechanics to analyze mechanical motion in both cases by choosing an analysis mode. The mode you choose can depend on the topology of your system.

Analysis Mode	Type of Analysis
Forward Dynamics	Forward dynamics (any topology)
Trimming	Forward dynamics (steady-state motion)
Inverse Dynamics	Inverse dynamics (open topology)
Kinematics	Inverse dynamics (closed topology)

Applying the Motion Modes

For more about motion modes, see these other sections.

- "Simulating and Analyzing Mechanical Motion" on page 1-20 is an overview of the SimMechanics analysis modes.
- "Analyzing the Motion" on page 5-7 contains detailed steps to implement these modes in your model.
- The case study "Finding Forces from Motions" on page 8-7 applies inverse dynamics to SimMechanics models.

Forces and Torques Determine Accelerations

Newton's second law of motion relates the force on a body, its mass, and the acceleration it experiences as a result of that force. The equivalents for rotational motion are the Euler equations.

Newton's Equations for Translational Dynamics

Let $\boldsymbol{F}_\mathrm{A}$ be the net force acting on a body A that has a constant mass m_A and a center of gravity (CG) position $\boldsymbol{x}_\mathrm{A}$. Newton's second law, valid for an inertial observer, relates the force on A to the translational acceleration of its CG.

$$\boldsymbol{F}_A = m_A \frac{d^2\boldsymbol{x}_A}{dt^2}$$

Equivalently, the *linear momentum* $\boldsymbol{p}_\mathrm{A} = m_\mathrm{A}\boldsymbol{v}_\mathrm{A}$ relates to force as $\boldsymbol{F}_\mathrm{A} = d\boldsymbol{p}_\mathrm{A}/dt$.

In forward dynamics, the force $\boldsymbol{F}_\mathrm{A}$ is given and the motion $\boldsymbol{x}_\mathrm{A}(t)$ is found by integration, supplemented by initial position and velocity. In inverse dynamics, the motion $\boldsymbol{x}_A(t)$ is given and the force on the body is found. In both cases, the mass must be known.

Euler's Equations for Rotational Dynamics

Rotational motion requires a *pivot*, the fixed center of rotation, and the *angular velocity* vector $\boldsymbol{\omega}$ with respect to that pivot. If $\boldsymbol{r}$ is the position, with respect to the pivot, of any point in a body, the velocity $\boldsymbol{v}$ of that point is $\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{r}$.

The equivalent of the mass of a body in rotational dynamics is the inertia tensor I, a 3-by-3 matrix.

$$I_{ij} = \int_V dV \left[\delta_{ij} |\boldsymbol{r}|^2 - r_i r_j \right] \rho(\boldsymbol{r})$$

The body's mass density $\rho(\boldsymbol{r})$ is a function of $\boldsymbol{r}$ within the body's volume V. The indices i, j range over 1, 2, 3, or x, y, z. Thus

$$I_{xx} = \int_V dV \left[y^2 + z^2 \right] \rho(\boldsymbol{r}),\ I_{xy} = \int_V dV\,[-xy]\,\rho(\boldsymbol{r}),\ \text{etc.}$$

The *angular momentum* of a body is $\boldsymbol{L} = \boldsymbol{I} \cdot \boldsymbol{\omega}$. The equivalent of the force on a body in rotational dynamics is the torque τ, which is produced by a force $\boldsymbol{F}$ acting on the body at a point $\boldsymbol{r}$ as $\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F}$.

The analog to Newton's second law for rotational motion, as measured by an inertial observer, just equates the torque $\boldsymbol{\tau}_A$ applied to a body A, defined with respect to a given pivot, to the time rate of change of $\boldsymbol{L}_A$. That is, $\boldsymbol{\tau}_A = d\boldsymbol{L}_A/dt$. It is easiest to take the pivot as the origin of an inertial coordinate system such as World. Unlike the case of translational motion, however, where the mass m_A remains constant as the body moves, the inertia tensor $\boldsymbol{I}_A$ changes as the body rotates, if it is measured in an inertial frame. There is no simple way to relate $d\boldsymbol{L}_A/dt$ to the angular acceleration $d\boldsymbol{\omega}/dt$.

The common solution to this difficulty is to work in the body's own rotating frame, where the inertia tensor is constant, and take the body's CG as the pivot. Diagonalize the inertia tensor. Since I is real and symmetric, its eigenvalues (I_1, I_2, I_3) (the *principal moments of inertia*) are real. Its eigenvectors form a new orthogonal triad, the *principal axes* of the body. But this frame fixed in the body is not inertial, and the torque-angular acceleration relationship is modified from its inertial form into the *Euler equations:*

$$I_1\omega_1 - \omega_2\omega_3(I_2 - I_3) = \tau_1$$
$$I_2\omega_2 - \omega_3\omega_1(I_3 - I_1) = \tau_2$$
$$I_3\omega_3 - \omega_1\omega_2(I_1 - I_2) = \tau_3$$

The components of the rotational vectors here are projected along the principal axes that move with the body's rotation.

Linearizing the Dynamical Equations

To study a system's response to and stability against external changes, you can apply small perturbations in the motion or the forces/torques to a known trajectory and force/torque set. SimMechanics and Simulink provide analysis modes and functions for analyzing the results of perturbing mechanical motion. The later sections of this chapter, demonstrate their use:

- "Trimming Mechanical Models" on page 8-18
- "Linearizing Mechanical Models" on page 8-32

You can perturb Newton's and Euler's laws with a small additional force $\Delta\boldsymbol{F}$ and torque $\Delta\boldsymbol{\tau}$ and determine the associated perturbations in motion, $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{\omega}$. You can also perturb the system inversely, making small changes to the motion and determining the forces and torques necessary to create those changes.

The perturbed Newton's and Euler's equations are

$$\boldsymbol{F} = m \cdot d^2(\Delta\boldsymbol{x})/dt^2$$

and

$$I_1\Delta\omega_1 - (\Delta\omega_2 \cdot \omega_3 + \omega_2 \cdot \Delta\omega_3)(I_2 - I_3) = \Delta\tau_1$$
$$I_2\Delta\omega_2 - (\Delta\omega_3 \cdot \omega_1 + \omega_3 \cdot \Delta\omega_1)(I_3 - I_1) = \Delta\tau_2$$
$$I_3\Delta\omega_3 - (\Delta\omega_1 \cdot \omega_2 + \omega_1 \cdot \Delta\omega_2)(I_1 - I_2) = \Delta\tau_3$$

The vector components of the Euler's equations are projected along the body's moving principal axes.

Linearizing the Constraints

If your model has constraints, you must perturb them as well:

$$g(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0 \quad , \quad \frac{\partial g}{\partial \boldsymbol{x}} \cdot \Delta\boldsymbol{x} + \frac{\partial g}{\partial \dot{\boldsymbol{x}}} \cdot \Delta\dot{\boldsymbol{x}} = 0$$

Finding Forces from Motions

The SimMechanics Kinematics and Inverse Dynamics modes (see “Analyzing the Motion” on page 5-7) enable you to find all the forces on a closed-loop system or an open system, respectively, given a model that completely specifies the system’s motions. Unlike Forward Dynamics mode, these modes do not need to compute the positions, velocities, and accelerations of the model’s components, because the model specifies them. Consequently, Kinematics and Inverse Dynamics modes take less time to compute the forces on a system. The time saving depends on the size and complexity of the system being simulated.

To use these modes, you must first build a model of the system that specifies completely the positions, velocities, and accelerations of the system’s bodies. Such a model is called a *kinematic* model. You create a kinematic model by interconnecting blocks representing the bodies and joints of the system and then connecting actuators to the joints to specify the motions of the bodies.

A model does not have to actuate every joint to specify completely the motions of a system. In fact, the model need actuate only as many joints as there are independent degrees of freedom in the system. (See “Counting Degrees of Freedom” on page 4-77.) For example, a model of a four bar mechanism need actuate only one of the mechanism’s joints, because a four bar mechanism has only one degree of freedom. To avoid overconstraining the model’s solution, the number of actuated joints should not exceed the number of degrees of freedom.

Caution Attempting to simulate an overconstrained model causes Simulink to stop the simulation with an error.

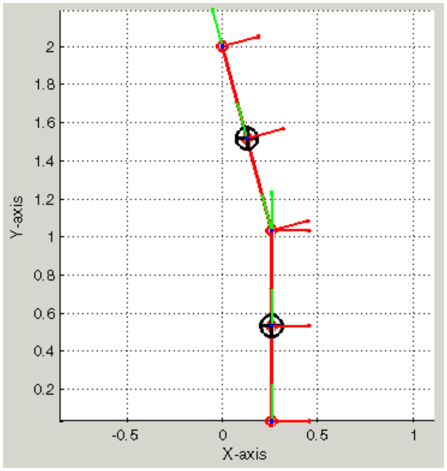
The following sections show how to use the Inverse Dynamics and Kinematics modes to find the forces on the joints of a closed- and an open-topology system, respectively.

- “Inverse Dynamics Mode with a Double Pendulum” on page 8-8
- “Kinematics Mode with a Four Bar System” on page 8-13

Inverse Dynamics Mode with a Double Pendulum

Note The Inverse Dynamics mode works only on open topologies and requires motion-actuating every independent DoF (see "Counting Degrees of Freedom" on page 4-77).

Consider a double pendulum consisting of two thin rods each 1 meter long and weighing 1 kilogram. The upper rod is initially rotated 15 degrees from the perpendicular.



Suppose that you want the pendulum to follow a certain trajectory. How much torque is required to make the pendulum follow this prescribed motion? Solving this problem entails building a kinematic model of the moving pendulum.

- The model must represent the geometry of the double pendulum and specify its motion trajectory throughout the simulation.
- The model must also measure the *computed torque* on each joint, the torque necessary to reproduce the specified motion.

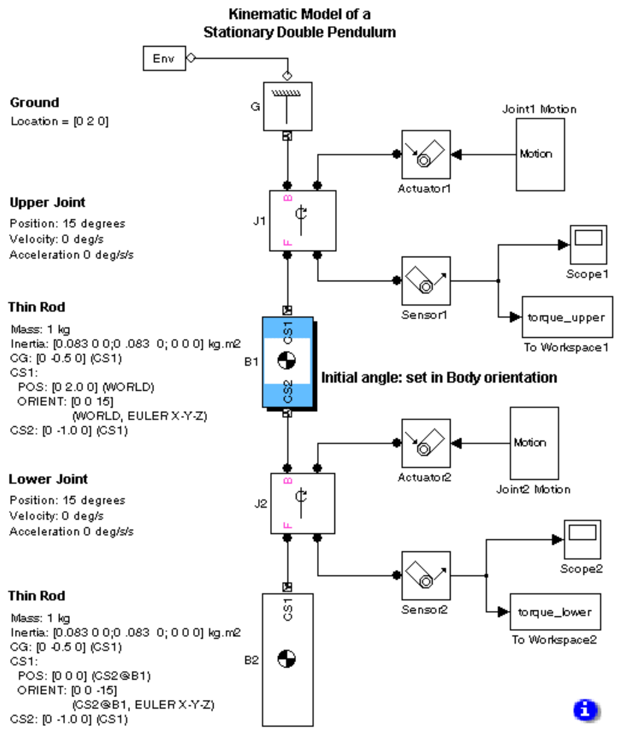
  Except in simple cases, you can find these computed torques only as approximate functions of time.

The kinematic model can take different approaches to specifying the initial state of the pendulum.

- One approach uses Body block parameters to specify the initial states.
- Another approach uses Actuator block signals.

Using Body Blocks to Specify Initial Conditions

Open the model `mech_dpend_invdyn1`. It illustrates the Body block approach to modeling initial states.

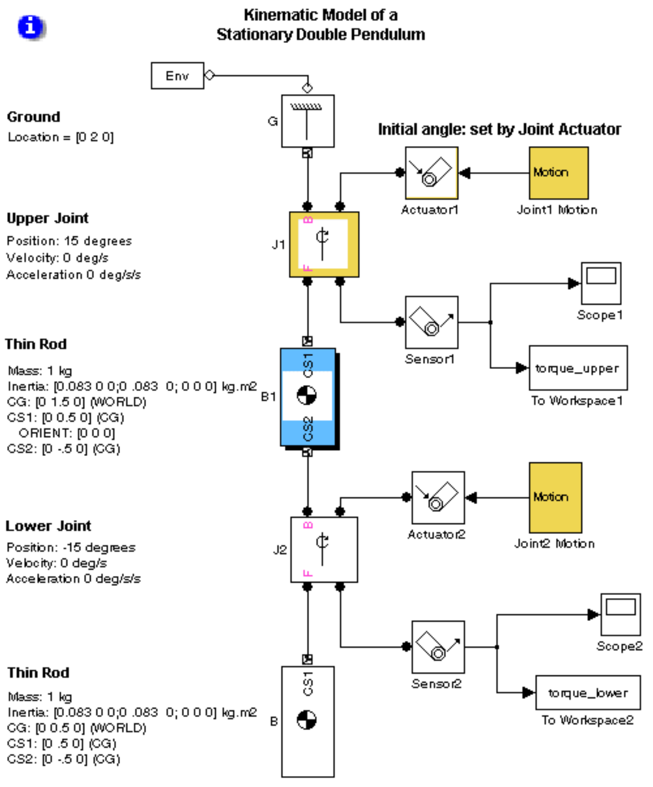


This model represents the pendulum by two Body blocks and two Revolute Joint blocks.

- The CS1 axis of the upper body (B1) of the pendulum is rotated 15 degrees from the perpendicular (see annotation for block B1).
- The coordinate systems for the lower block (B2) are aligned with CS1 of the upper block. The CS1 of B2 is rotated -15 degrees relative to CS1 of B1, i.e., it is perpendicular to the World coordinate system.

Using Actuator Blocks to Specify the Initial States

Open the model mech_dpend_invdyn2. It shows the use of Joint Actuator blocks to specify the initial kinematic state. Using actuators to specify the displacement slightly simplifies the configuration of the Body blocks.



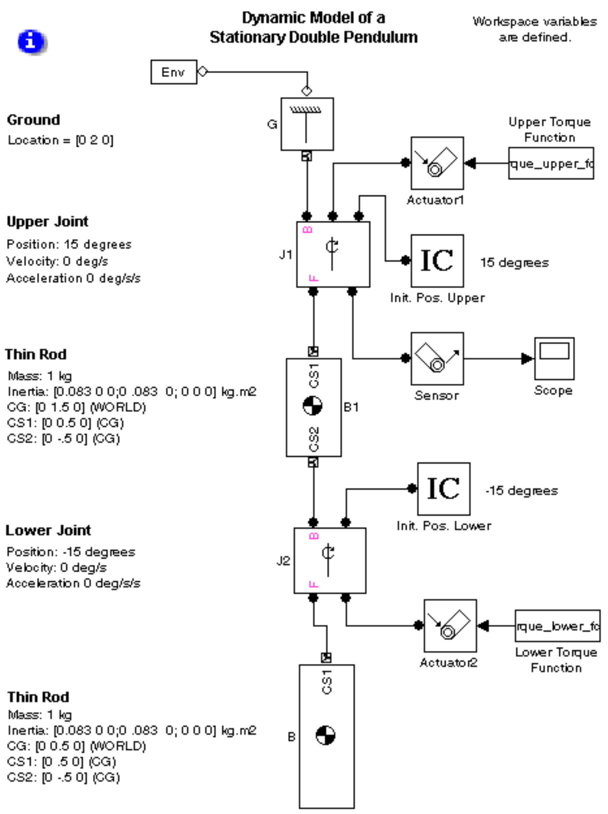
Specifying the Motion and Measuring the Computed Torques

In either model, the Joint Actuator blocks connected to the Joint blocks specify that the upper and lower joints accelerate at two distinct rates, π/2 and -π/4 radians/second^2, respectively. Sensor blocks connected to To Workspace blocks measure the computed torques on the upper and lower joints as MATLAB workspace variables `torque_upper` and `torque_lower`, respectively. These vectors capture the upper and lower computed torques at each major time step. You must simulate either model in Inverse Dynamics mode to compute the joint torques required to maintain the pendulum in its motion.

Using the Computed Torques in Forward Dynamics

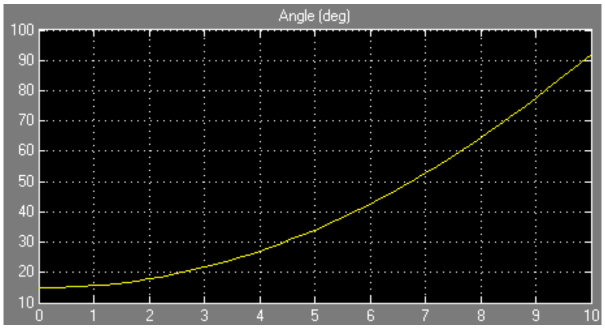
Once you know the computed torques as functions of time, you can verify that these are the correct answers by creating a version of the model that applies the computed torques to the joints and simulating that model in Forward Dynamics mode.

Open the model `mech_dpend_act`. It illustrates a forward dynamics version of the kinematic model that uses the joint actuators to specify the initial angular displacement of the pendulum bodies.



This model uses Initial Condition blocks to specify the initial 15 degree displacement of the upper body from the vertical in the world coordinate system and the corresponding initial -15 degree displacement of the lower body from the vertical in the coordinate system of the upper body. The negative displacement of the lower body is equivalent to positioning it as vertical in the world coordinate system.

From a MAT-file, the model loads the upper and lower torques, `torque_lower_fcn` and `torque_upper_fcn`, as two matrices representing discrete functions of time. Simulating this model in Forward Dynamics mode results in the following display on the upper joint scope.



If the computed torques were known exactly as continuous functions of time in the two inverse dynamics models, this plot would exactly match the upper joint motion in the original models. But the torques are measured only in a discrete approximation, and `mech_dpend_act` does not exactly reproduce the original motion.

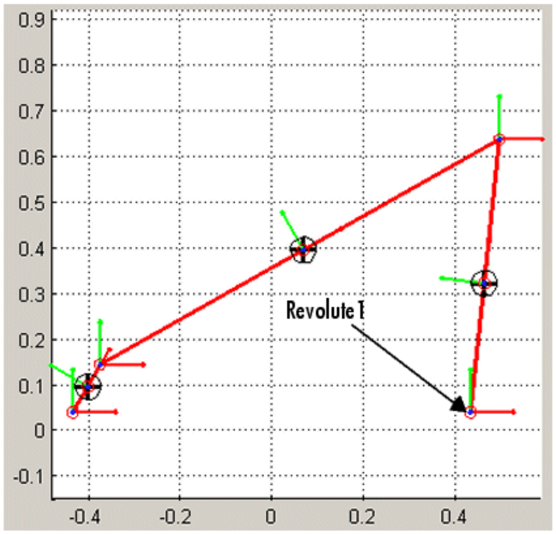
Making More Accurate Torque Measurements

You can achieve better approximations by adjusting Simulink to report sensor outputs in the original models with finer time steps. Refer to the Simulink documentation for more about exporting simulation data and refining output.

Kinematics Mode with a Four Bar System

Note The Kinematics mode works only on closed topologies and requires motion-actuating every independent DoF (see “Counting Degrees of Freedom” on page 4-77). There must also be no Joint Stiction Actuators and no nonholonomic constraints.

Consider the four bar system illustrated by the tutorial titled “Four Bar Mechanism” on page 2-36. The model is `mech_four_bar`.



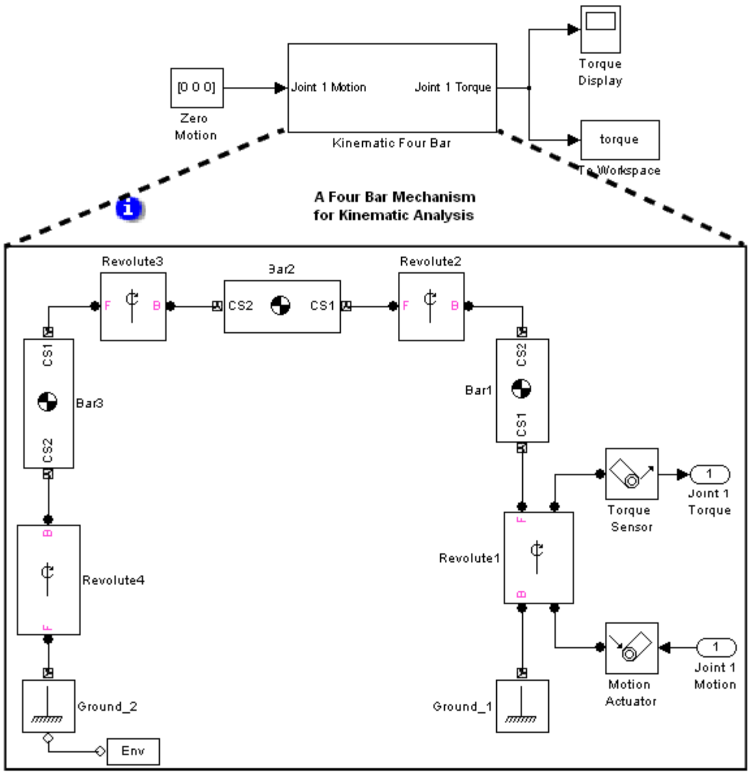
Suppose that you want to keep this system from collapsing under its own weight. Because the four bar has only one degree of freedom, applying a counterclockwise torque to the joint labeled Revolute1 would accomplish this objective. But how much torque is sufficient?

To answer this question, you must build a kinematic model of the stationary four bar system, starting with the tutorial model. The kinematic model must specify how the system moves over time. In this case, the four bar remains stationary. You can use a Joint Actuator to implement this requirement.

Transforming Forward into Inverse Dynamics

Open the model `mech_four_bar_kin`, derived from `mech_four_bar`.

- The model uses a Joint Actuator block driven by a Constant block to specify the motion on the Revolute1 joint. The Constant block outputs a three-element vector that specifies the angular position, velocity, and acceleration, respectively, of the joint as 0.
- The model uses a Joint Sensor block connected to a Scope block to display the resulting torque on the joint and a To Workspace block to save the torque signal to the MATLAB workspace.

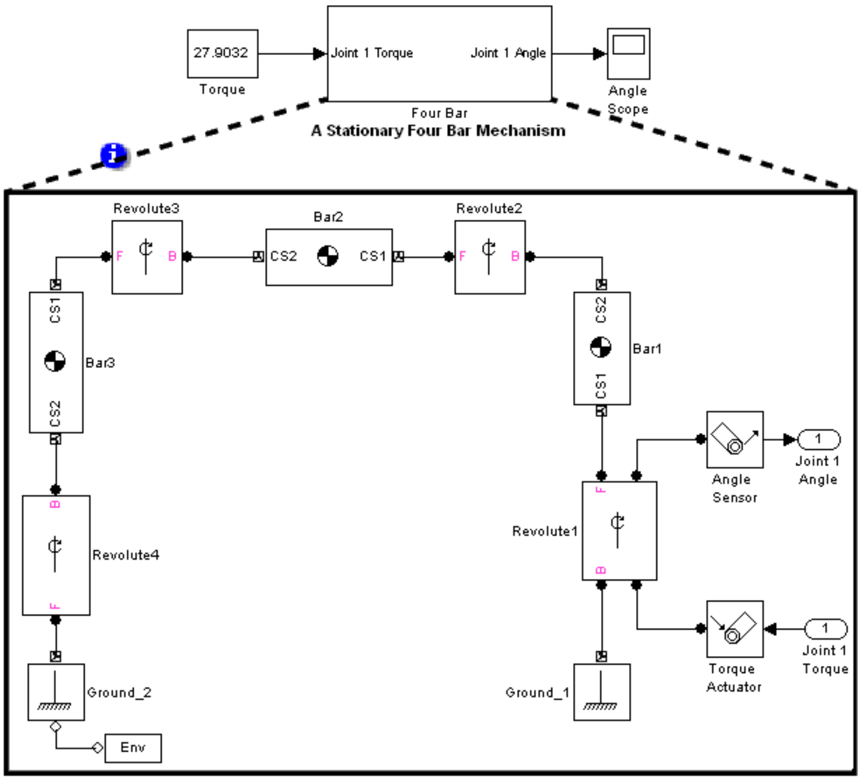


Finding and Checking the Needed Torque

Now obtain and verify the inverse dynamics solution to the question.

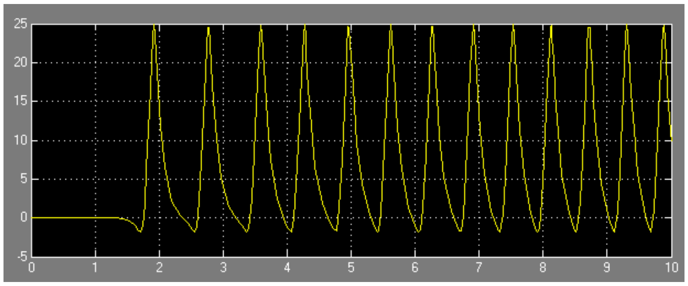
1 Run this model in Kinematics mode. The output reveals that the torque on the Revolute1 joint is 27.9032 newton-meters.

2 To verify that the computed torque is, indeed, the torque required to keep the system stationary, create a forward-dynamics model that applies the computed torque to the Revolute1 joint. Open such a model contained in `mech_four_bar_stat`.



3 Run the model in Forward Dynamics mode, with the Revolute1 Angle Scope open.

The Scope display reveals that the machine does, indeed, remain stationary, although only for about 1.5 seconds. The model is nonlinear and unstable, and the computed force value is not copied exactly in the new model.


25
20
15
10
5
0
-5
0
1
2
3
4
5
6
7
8
9
10

Trimming Mechanical Models

Trimming a mechanical system refers to the finding of solutions for inputs, outputs, states, and state derivatives satisfying conditions that you specify beforehand. For example, you can seek steady-state solutions where some or all of the derivatives of a system's states are zero. To use the Simulink `trim` command on a system represented by a SimMechanics model, you must select the SimMechanics Trimming mode (see "Analyzing the Motion" on page 5-7). You must also specify the conditions that the solution must satisfy. These examples show you how to trim mechanical models.

- "Unconstrained Trimming of a Spring-Loaded Double Pendulum" on page 8-20
- "Constrained Trimming of a Four Bar Machine" on page 8-26

Consult the Simulink documentation for more on trimming models. You can also enter `help trim` at the MATLAB command line.

Restrictions on Trimming Mechanical Models

You should avoid using certain SimMechanics or Simulink features when trimming a model.

- A trimmed SimMechanics mechanism must be assembled. Do not use disassembled joints while trimming.

  For more information, see "Modeling with Disassembled Joints" on page 4-33.
- You cannot use Driver blocks while trimming a model.
- SimMechanics ignores Joint Initial Condition Actuator blocks in a trimmed model.
- Do not incorporate events or motion discontinuities in your trimmed model. In particular, do not use SimMechanics Joint Stiction Actuator blocks. Trimming mechanical models with stiction causes an error.

Trimming in the Presence of Motion Actuation

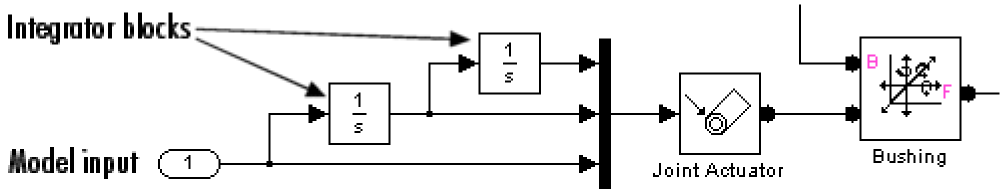
If you want to trim a SimMechanics model containing motion actuators, you must

1. Make the velocity and position/angle parts of the motion actuation signal dependent only on the acceleration signal
2. Make the velocity and position/angle consistent with the acceleration part by use of Integrator blocks. A motion actuation signal is a vector with components ordered as position/angle, velocity, and acceleration, respectively.

   This technique is recommended in “Stabilizing Numerical Derivatives in Actuator Signals” on page 4-45. It is required here.

During trimming, SimMechanics uses only the acceleration as an independent motion actuation input because it is equivalent to a force or torque. As a consequence, only the acceleration signal can be used as an independent motion actuation input.

A similar restriction holds for model linearization; see “Linearizing in the Presence of Motion Actuation” on page 8-33.



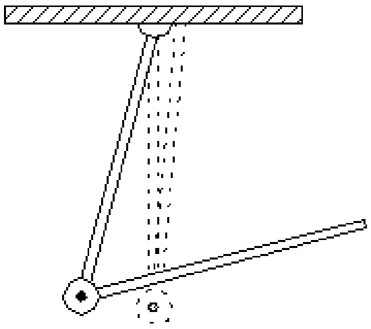
Motion Actuation as a Model Input for Trimming

Motion Actuation as an Indirect Input

You can put your model input port in another part of your model, then feed that input as an acceleration into a motion actuator with a Simulink signal line. You must still derive the velocity and position/angle motion actuation signals in the same way: by integrating whatever signal you use for acceleration once and twice, respectively.

Unconstrained Trimming of a Spring-Loaded Double Pendulum

Consider the following spring-loaded double pendulum.

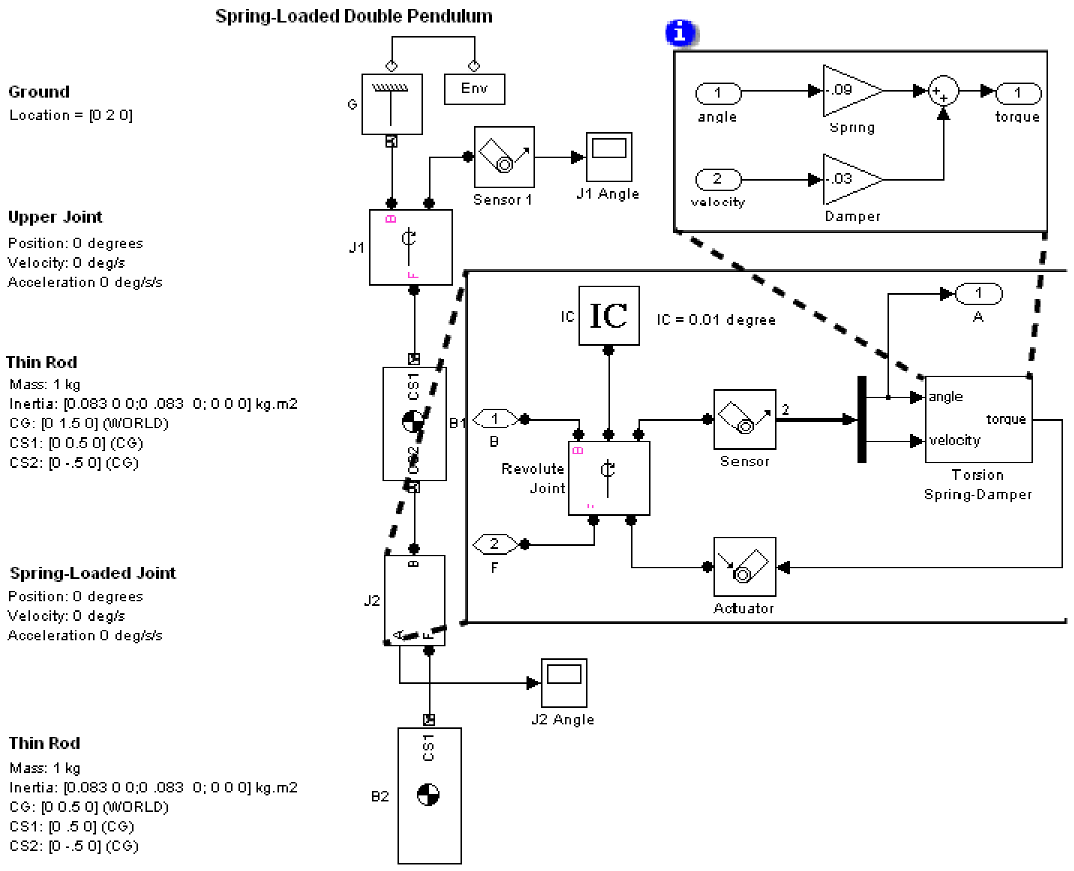


The joint connecting the upper and lower arms of this pendulum contains a torsional spring and damper system that exerts a counterclockwise torque linearly dependent on the angular displacement and velocity of the joint. Suppose that the lower arm is folded upward almost vertically and then allowed to fall under the force of gravity. At what point does the spring-damper system reach equilibrium. That is, at what point does it cease to unfold?

Making an Initial Equilibrium Guess

To find an equilibrium point for the spring-loaded double pendulum,

1 Build a SimMechanics model of the system. This diagram shows an example of such a model, `mech_dpend_trim`.



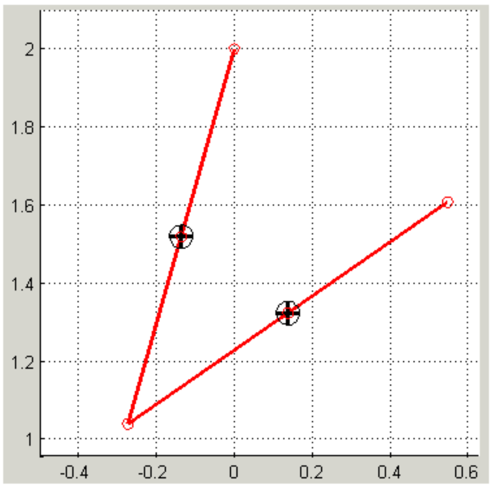
- This model uses Body blocks to model the upper and lower arms of the pendulum and a Revolute Joint block (J1) to model the connection between the pendulum and ground.
- The model uses a Subsystem block (J2) to model the spring-loaded revolute joint between the arms. This subsystem uses a negative feedback loop to model a joint subject to a damped torsional spring by multiplying the angular displacement and velocity of the joint, respectively, by spring and damper constants. The loop sums the

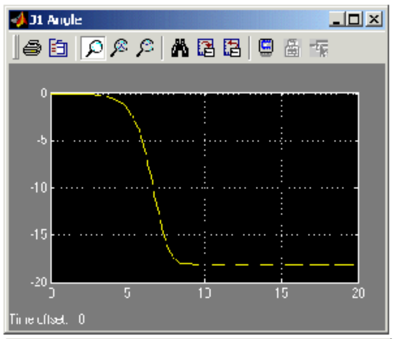
resulting torques and feeds them back into the joint with a Joint Actuator block.

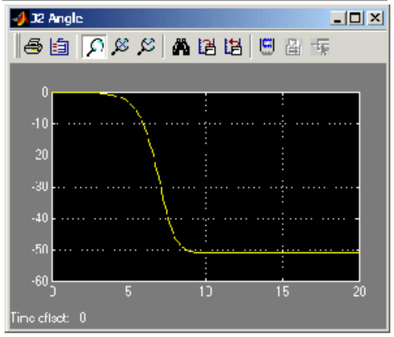
The result is that the joint experiences a torque opposing its motion and proportional to its angular displacement and velocity. You could also model this damped torsional spring with a Joint Spring & Damper block.

The spring and damper constants used here were chosen by running the model with various candidate values and choosing the ones that resulted in a moderate deflection of the pendulum.

2 Run the model in Forward Dynamics mode to estimate an initial guess for the nontrivial equilibrium point of the pendulum.







The simulation reveals that the spring stops unfolding after about 9 seconds; that is, it reaches a steady-state point. At this point the angles of the upper and lower joints are about -18 and -51 degrees, respectively, and the velocities are zero. The `trim` command can find the values of these states precisely.

Analyzing and Initializing the State Vector

Examine the model's state vector and prepare it for use in trimming.

1 Determine the layout of the model's state vector, in order to tell the `trim` command where in the model's state space to start its search for the pendulum's equilibrium point (the point where it stops unfolding). Use the SimMechanics `mech_stateVectorMgr` command to perform this task. Refer to the Ground block, G.

```
v = mech_stateVectorMgr('mech_dpend_trim/G');
v.StateNames

ans =
    'mech_dpend_trim/J2/RevoluteJoint:R1:Position'
    'mech_dpend_trim/J1:R1:Position'
    'mech_dpend_trim/J2/RevoluteJoint:R1:Velocity'
    'mech_dpend_trim/J1:R1:Velocity'
```

The `StateNames` field of the state vector object returned by `mech_stateVectorMgr` lists the names of the model's states in the order in which they appear in the model's state vector. Thus the field reveals that the model's state vector has the following structure:

```
x(1) = position of lower joint (J2)
x(2) = position of upper joint (J1)
x(3) = velocity of lower joint (J2)
x(4) = velocity of upper joint (J1)
```

2 Determine an initial state vector.

The initial state vector specifies the point in a system's state space where the `trim` command starts its search for an equilibrium point. The `trim` command searches the state space outward from the starting point, returning the first equilibrium point that it encounters. Thus, the starting point should not be at or near any of a system's trivial equilibrium points. For the double pendulum, the point [0; 0; 0; 0] (i.e., the pendulum initially folded up and stationary) is a trivial equilibrium point and therefore should be avoided. The initial state vector must be a column vector and must specify angular states in radians.

Often, the choice of a good starting point can be found only by experiment, that is, by running the `trim` command repeatedly from different starting points to find a nontrivial equilibrium point. This is true of the double pendulum of this example. Experiment reveals that this starting point,

```
ix(1) = J2 (lower joint) angle = -35 degrees = -0.6109 radians
ix(2) = J1 (upper joint) angle = -10 degrees = -0.1745 radians
ix(3) = J2 angular velocity = 0 radians/second
ix(4) = J1 angular velocity = 0 radians/second
```

yields a nontrivial equilibrium point. You can save time by creating an initial state vector set to these values.

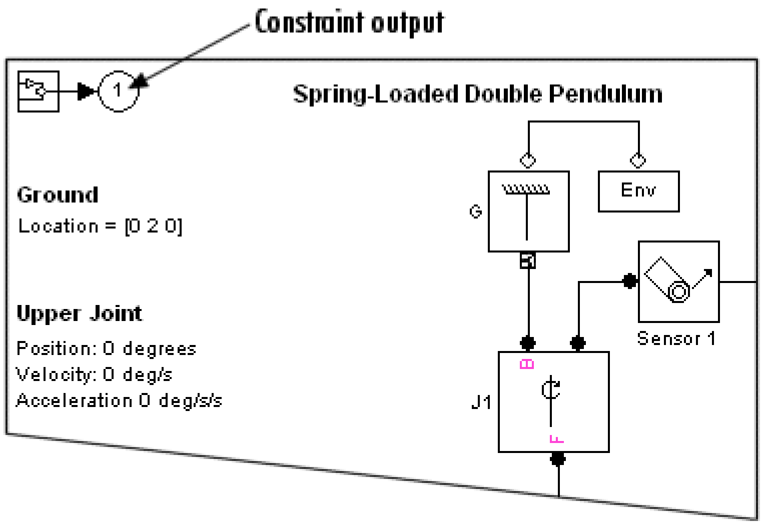
```
ix = [-35*pi/180; -10*pi/180; 0; 0];
```

Note The `trim` command ignores initial states specified by Joint Initial Condition Actuator blocks. Thus, you cannot use these blocks to specify the starting point for trimming a model. If your model contains IC blocks, create the initial state vector as if the IC blocks did not exist.

Trimming the System to Equilibrium

1 Reset the analysis type to `Trimming` on the **Parameters** pane of the Machine Environment dialog.

   This option inserts a constraint subsystem and associated output at the top level of the model. SimMechanics inserts the constraint output to make the constraints available to the `trim` command. The spring-loaded double pendulum has no constraints. Hence the constraint outport does not output nontrivial constraint data and is not needed to trim the pendulum.



2 Enter the following commands to find the equilibrium point nearest to the starting point.

```
ix = [-35*pi/180; -10*pi/180; 0; 0];
iu = [];
[x,u,y,dx] = trim('mech_dpend_trim',ix,iu);
```

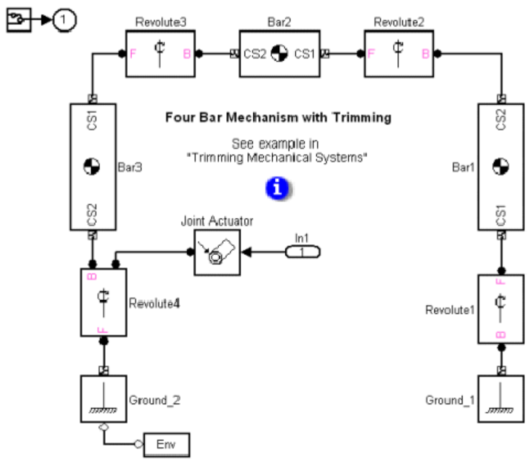
The array `ix` specifies the starting point determined in "Analyzing and Initializing the State Vector" on page 8-23. The array `iu` specifies the initial inputs of the system. Its value is null because the system has no inputs. (Thus the `u` and `y` outputs are null.) In this form, the `trim` command finds a system's steady-state (equilibrium) points, i.e., the points where the system's state derivatives are zero. The array `x` contains the state vector corresponding to the first equilibrium point located by `trim`:

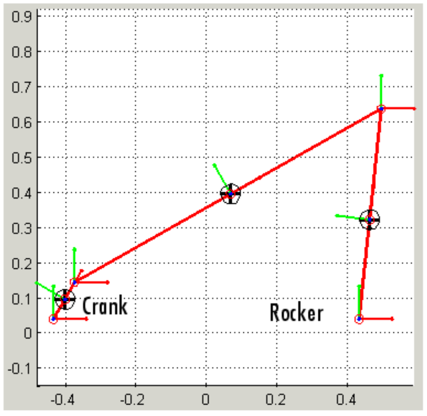
```
x =
    -0.8882
    -0.3165
    -0.0000
     0.0000
```

The resulting states are angular positions and velocities expressed in radians. Based on the layout of the model's state vector (determined previously in “Analyzing and Using the State Vector” on page 8-27) the pendulum reaches equilibrium when its upper joint has deflected to an angle of -18.1341 degrees and its lower joint to an angle of -50.8901 degrees. The system state derivatives `dx` are zero, within tolerances.

Constrained Trimming of a Four Bar Machine

Consider a planar four bar system consisting of a crank, a coupler, and a rocker. The following figure shows a block diagram and a convex hull diagram of the four bar system. The model is `mech_four_bar_trim`.





This system is constrained by virtue of being a closed loop. Not all the degrees of freedom are independent. (In fact, only one is.) Suppose you want to find the torque required to turn the crank at an angular velocity of 1 radian/second over a range of crank angles. This section outlines the procedure with the `trim` command and the SimMechanics Trimming mode to determine the torque.

Setting Up the Four Bar for Trimming

Reconfigure the model before performing the trim.

1 Cut the closed loop that represents the four bar system at the joint (Revolute1) connecting the rocker to ground (see “Modeling Bodies and Grounds” on page 4-10).

Manually cutting the rocker joint ensures that SimMechanics does not cut the four bar loop at the crank joint and thereby eliminate the crank's position and velocity from the system's state vector.

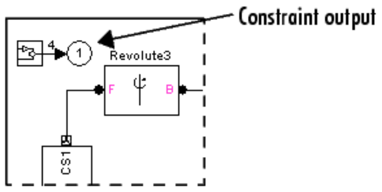
For instructions and additional information on cutting joints, see "Cutting Closed Loops" on page 4-36 and "Configuring SimMechanics Simulation Diagnostics" on page 5-12.

2 Select **Signal Dimensions** from the **Format > Port/Signal Displays** menu.

Simulink then displays the width of signals on the model diagram and hence enables you to read the number of constraints on the four bar system from the diagram in the next step.

3 Set the analysis mode to `Trimming` in the Machine Environment block.

SimMechanics then inserts a subsystem and an output block that outputs a signal representing the mechanical constraints on the four bar system. These constraints arise from the closure of the loop.



The width of the constraint signal (4) reflects the fact that the four bar system is constrained to move in a plane and thus has only four constraints: two position constraints and two velocity constraints.

Analyzing and Using the State Vector

Examine the state vector and prepare it for use in trimming.

1 Reveal the layout of the system's state vector with `mech_stateVectorMgr`:

```
Handle = get_param('mech_four_bar_trim/Revolute2','handle');
StateManager = mech_stateVectorMgr(Handle);
StateManager.StateNames
```

```
ans =
    'mech_four_bar_trim/Revolute2:R1:Position'
    'mech_four_bar_trim/Revolute3:R1:Position'
    'mech_four_bar_trim/Revolute4:R1:Position'
    'mech_four_bar_trim/Revolute2:R1:Velocity'
    'mech_four_bar_trim/Revolute3:R1:Velocity'
    'mech_four_bar_trim/Revolute4:R1:Velocity'
```

2 Specify the initial state vector `x0` and the index array `ix`:

```
x0  = [0;0;0;0;0;1];
ix  = [3;6];
```

The array `x0` specifies that the `trim` command should start its search for a solution with the four bar system in its initial position and with the crank moving at an angular velocity (state 6) of 1 radian/second. The array `ix` specifies that the angular position (state 3) and velocity (state 6) of the crank must equal their initial values, 0 radians and 1 radian/second, respectively, at the equilibrium point. It is not necessary to constrain the other states because the four bar system has only one independent position DoF and only one independent velocity DoF.

3 Specify zero as the initial estimate for the crank torque:

```
u0  = 0;
```

4 Require the constraint outputs to be 0:

```
y0  = [0;0;0;0];
iy  = [1;2;3;4];
```

The `y0` array specifies the initial values of the constraint outputs as zero. The `iy` array specifies that the constraint outputs at the solution point must equal their initial values (0). This ensures that the solution satisfies the mechanical constraints on the system.

Note The four bar system has only constraint outputs. If you were trimming a system with nonconstraint outputs, you would have to include the nonconstraint outputs in the initial output vector.

5 Specify the state derivatives to be trimmed:

```
dx0 = [0;0;1;0;0;0];
idx = [6];
```

The `dx0` array specifies the initial derivatives of the four bar system's states. In particular, it specifies that the initial derivative of the crank angle (i.e., the crank angle velocity) is 1 radian/second and all the other derivatives (i.e., velocities and accelerations) are 0. The `idx` array specifies that the acceleration of the crank at the solution point must be 0; i.e., the crank must be moving at a constant velocity. It is not necessary to constrain the accelerations of the other states because the system has only one velocity DoF.

Note The four bar system has only mechanical states. If you were trimming a system that has nonmechanical Simulink states, you would have to include these nonmechanical states in the initial state vector.

Trimming the Four Bar

Carry out the trimming and study the output.

1 Trim the system at the initial crank angle to verify that you have correctly set up the trim operation:

```
[x,u,y,dx] = ...
   trim('mech_four_bar_trim',x0,u0,y0,ix,[],iy,dx0,idx);
```

Trim the system over a range of angles.

```
Angle = [];
Input = [];
State = [];
```

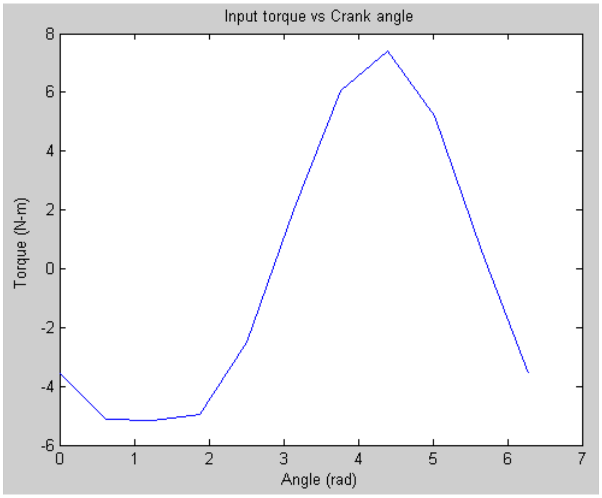
```
dAngle = 2*pi/10;
Constraint = [];

for i=1:11;
    x0(3) = (i-1)*dAngle;
    x0(6) = 1;
    [x,u,y,dx] = ...
trim('mech_four_bar_trim',x0,u0,y0,ix,[],iy,dx0,idx);
   disp(['Iteration: ', num2str(i), ' completed.']);
   Angle(i) = x0(3);
    Input(:,i) = u;
    State(:,i) = x;
    Constraint(:,i) = y;
     if (i>3),
         u0 = spline(Angle,Input,Angle(end) + dAngle);
         x0 = spline(Angle,State,Angle(end) + dAngle);
     else
         x0 = x;
         u0 = u;
      end; end;
```

2 Plot the results.

```
figure(1);
plot(Angle,Input);
grid;
xlabel('Angle (rad)');
ylabel('Torque (N-m)');
title('Input torque vs crank angle');
```

The following figure shows the resulting plot.



For More Information About Trimming Closed-Loop Systems

The following section, “Linearizing Mechanical Models” on page 8-32 contains an example, “Closed-Loop Linearization: Four Bar Machine” on page 8-40, of trimming the system in a different way, searching for the stable natural equilibrium of the four bar mechanism.

Linearizing Mechanical Models

The Simulink `linmod` command creates linear time-invariant (LTI) state-space models from Simulink models. It linearizes each block separately. You can use this command to generate an LTI state-space model from a SimMechanics model, for example, to serve as input to Control System Toolbox commands that generate controller models. The `linmod` command allows you to specify the point in state space about which it linearizes the model (the *operating point*). You should choose a point where your model is in equilibrium, i.e., where the net force on the model is zero. You can use the Simulink `trim` command to find a suitable operating point (see "Trimming Mechanical Models" on page 8-18). By default, `linmod` uses an adaptive perturbation method to linearize model. The Machine Environment dialog allows you to require that `linmod` use a fixed perturbation method instead (see "Analyzing the Motion" on page 5-7). The following examples illustrate the use of `linmod` to linearize SimMechanics models.

- "Open-Topology Linearization: Double Pendulum" on page 8-34
- "Closed-Loop Linearization: Four Bar Machine" on page 8-40

Consult the Simulink documentation for more on "Linearizing Models". You can also enter `help linmod` at the MATLAB command line.

Restrictions on Linearizing Mechanical Models

There are restrictions on how you linearize mechanical models.

- If you specify any joint primitive initial conditions with Joint Initial Condition Actuator blocks, these initial condition values always override any state vector initial values specified via the `linmod` command.

  Joint primitives with JICA blocks are preferentially chosen for the set of independent states in linearization.

- Avoid incorporating discrete events or motion discontinuities in a linearized model. If you include event- or discontinuity-triggering blocks, ensure that the machine does not induce discontinuities as it moves through the linearized regime you are modeling.

  Use of Joint Stiction Actuator blocks in a linearized model causes an error.

- Because closed loops impose constraints on states, you cannot linearize a closed-loop SimMechanics model with the `linmod2` command.

Linearizing in the Presence of Motion Actuation

During linearization, SimMechanics uses only the acceleration as an independent motion actuation input because it is equivalent to a force or torque. A similar restriction holds for model trimming; see "Trimming in the Presence of Motion Actuation" on page 8-19. As a consequence, the only motion actuation signal that can be set as a model input is the acceleration signal.

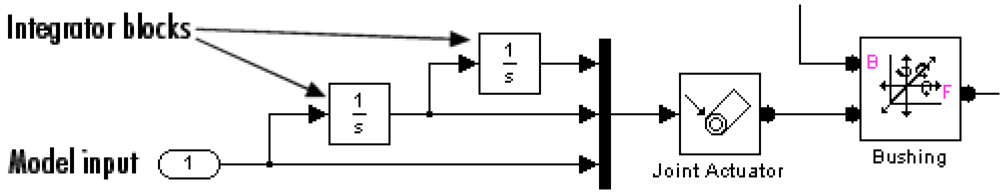
If you want to linearize a SimMechanics model containing motion actuators, you must

1. Make the velocity and position/angle parts of the motion actuation signal dependent only on the acceleration signal
2. Make the velocity and position/angle consistent with the acceleration part by use of Integrator blocks. A motion actuation signal is a vector with components ordered as position/angle, velocity, and acceleration, respectively.

   This technique is recommended in "Stabilizing Numerical Derivatives in Actuator Signals" on page 4-45. It is required here.

During linearization, SimMechanics uses only the acceleration as an independent motion actuation input because it is equivalent to a force or torque. As a consequence, only the acceleration signal can be used as an independent motion actuation input.

A similar restriction holds for model trimming; see "Trimming in the Presence of Motion Actuation" on page 8-19.



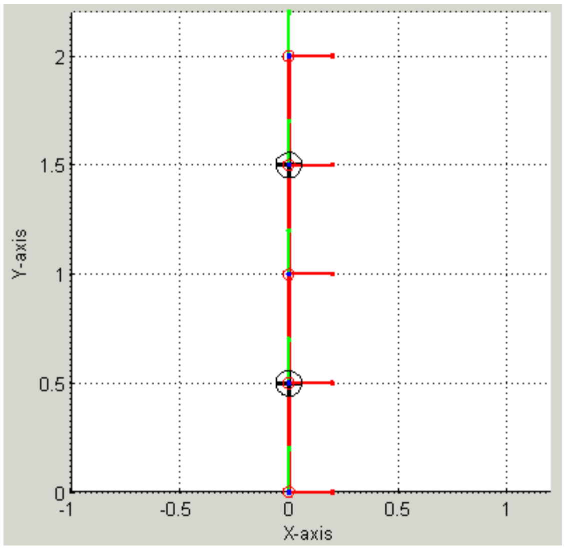
Motion Actuation as a Model Input for Linearization

Motion Actuation as an Indirect Input

You can put your model input port in another part of your model, then feed that input as an acceleration into a motion actuator with a Simulink signal line. You must still derive the velocity and position/angle motion actuation signals in the same way: by integrating whatever signal you use for acceleration once and twice, respectively.

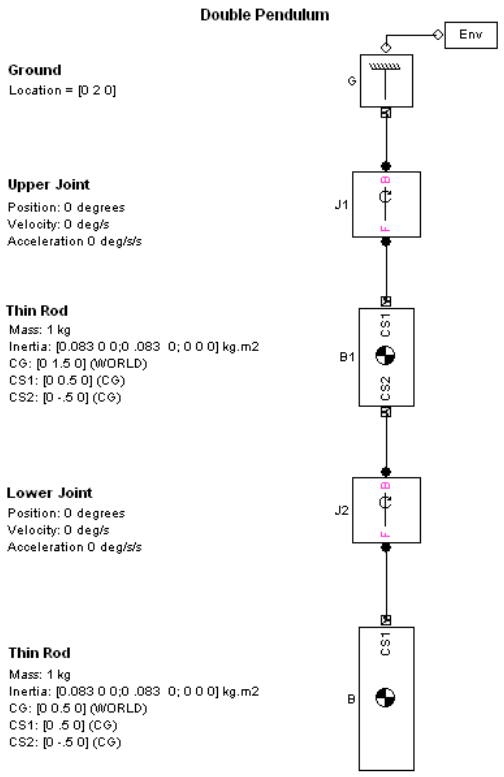
Open-Topology Linearization: Double Pendulum

Consider a double pendulum initially hanging straight up and down.



The net force on the pendulum is zero in this configuration. The pendulum is thus in equilibrium.

Open the `mech_dpend_forw` model.


Double Pendulum

Ground
Location = [0 2 0]

Upper Joint
Position: 0 degrees
Velocity: 0 deg/s
Acceleration 0 deg/s/s

Thin Rod
Mass: 1 kg
Inertia: [0.083 0 0;0 .083 0; 0 0 0] kg.m2
CG: [0 1.5 0] (WORLD)
CS1: [0 0.5 0] (CG)
CS2: [0 -.5 0] (CG)

Lower Joint
Position: 0 degrees
Velocity: 0 deg/s
Acceleration 0 deg/s/s

Thin Rod
Mass: 1 kg
Inertia: [0.083 0 0;0 .083 0; 0 0 0] kg.m2
CG: [0 0.5 0] (WORLD)
CS1: [0 .5 0] (CG)
CS2: [0 -.5 0] (CG)

Linearizing the Model

To linearize this model, enter

```
[A B C D] = linmod('mech_dpend_forw');
```

at the MATLAB command line. This form of the `linmod` command linearizes the model about the model's initial state.

Note Joint initial conditions specified with IC blocks always override any state vector initial values passed to the `linmod` command.

The double pendulum model in this example contains no IC blocks. The initial conditions specified with the `linmod` command are therefore implemented without modification.

Deriving the Linearized State Space Model

The matrices *A, B, C, D* returned by the `linmod` command correspond to the standard mathematical representation of an LTI state-space model:

$$\begin{aligned} \mathrm{d}\boldsymbol{x}/\mathrm{dt} &= A\cdot\boldsymbol{x}+B\cdot\boldsymbol{u} \\ \boldsymbol{y} &= C\cdot\boldsymbol{x}+D\cdot\boldsymbol{u} \end{aligned}$$

where x is the model's state vector, y is its outputs, and u is its inputs. The double pendulum model has no inputs or outputs. Consequently, only A is not null. This reduces the state-space model for the double pendulum to

$$\mathrm{d}\boldsymbol{x}/\mathrm{dt} = A\cdot\boldsymbol{x}$$

where

```
A =
          0          0     1.0000          0
          0          0          0     1.0000
  -137.3400    39.2400          0          0
    39.2400   -19.6200          0          0
```

This model specifies the relationship between the state derivatives and the states of the double pendulum. The state vector of the LTI model has the same format as the state vector of the SimMechanics model. The SimMechanics `mech_stateVectorMgr` command gives the format of the state vector as follows:

```
vm = mech_stateVectorMgr('mech_dpend_forw/G');
vm.StateNames
```

```
ans =
    'mech_dpend_forw/J2:R1:Position'
    'mech_dpend_forw/J1:R1:Position'
    'mech_dpend_forw/J2:R1:Velocity'
    'mech_dpend_forw/J1:R1:Velocity'
```

Right-multiplying A by the state vector $\boldsymbol{x}$ yields the differential state equations corresponding to the LTI model of the double pendulum,

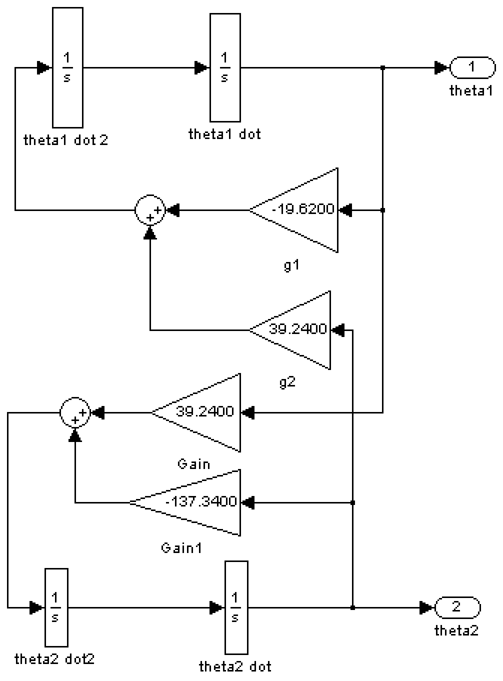
$$\ddot{\theta}_1 = -19.62 \cdot \theta_1 + \ 39.24 \cdot \theta_2$$
$$\ddot{\theta}_2 = +39.24 \cdot \theta_1 - 137.34 \cdot \theta_2$$

where

θ_1 = position of top joint (J1)
θ_2 = position of bottom joint (J2)

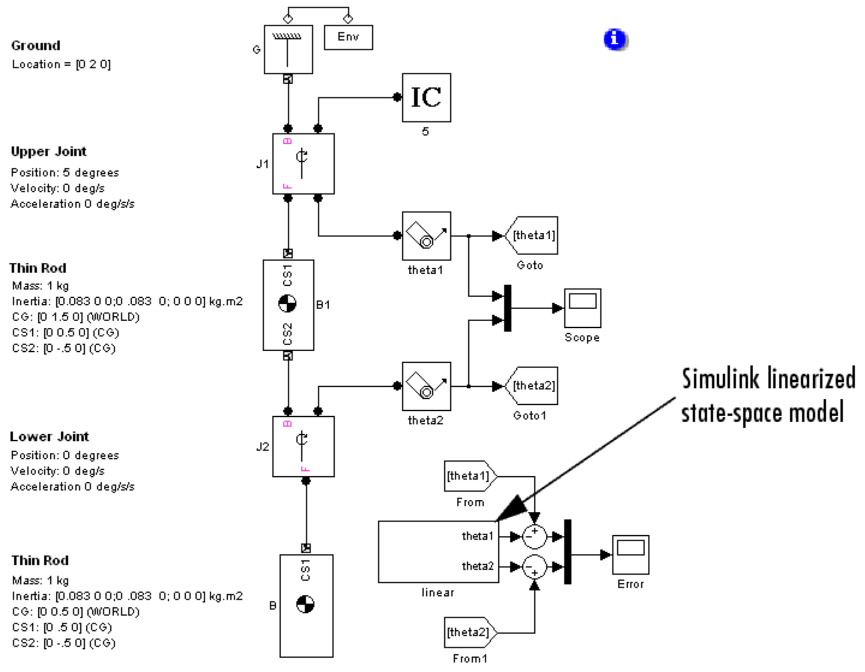
The array of coefficients on the right-hand side of the differential equations represents a matrix of squared frequencies. The eigenvalues of this matrix are the squared frequencies of the system's response modes. These modes characterize how the double pendulum responds to small perturbations in the vicinity of the operating point, which here is the force-free equilibrium.

The following Simulink model implements the state-space model represented by these equations.

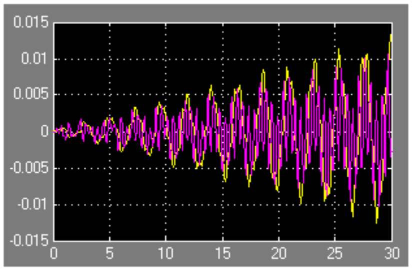


Modeling the Linearization Error

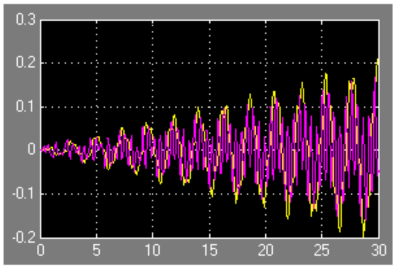
This model in turn allows creation of a model located in `mech_dpend_lin` that computes the LTI approximation error.



Running the model twice with the upper joint deflected 2 degrees and 5 degrees, respectively, shows an increase in error as the initial state of the system strays from the pendulum's equilibrium position and as time elapses. This is the expected behavior of a linear state-space approximation.



2 degrees



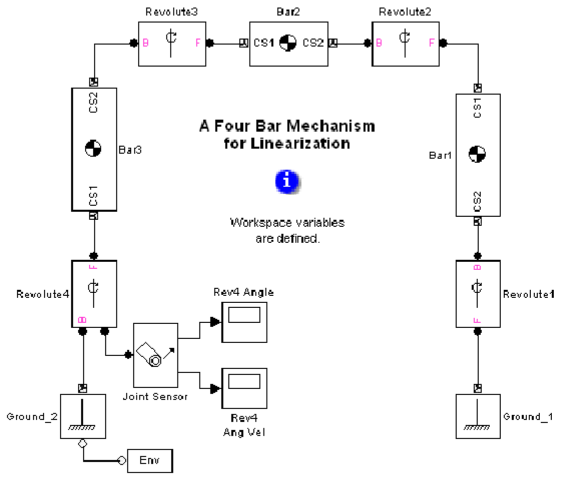
5 degrees

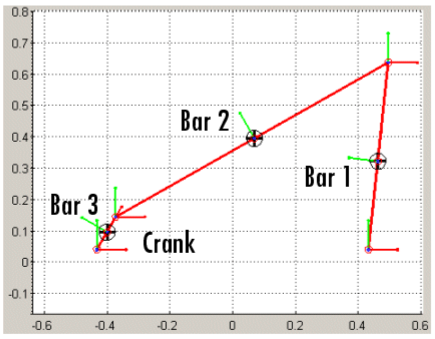
Closed-Loop Linearization: Four Bar Machine

Control System Toolbox Function This section uses the Control System Toolbox function `minreal` and assumes that this toolbox is installed on your system. Refer to the Control System Toolbox user's guide for more about this function and state-space analysis.

Linearizing a closed-loop system is more complex than open-topology analysis. Each closed loop in the machine imposes implicit constraints that render some of the degrees of freedom (DoFs) dependent. Linearization of such a system must recognize that not all the DoFs are independent. A straightforward implementation of the `linmod` command results in redundant system states. You can eliminate these with the `minreal` function, which finds the minimal state space needed to represent your linearized model. To ensure that `minreal` produces a nonnull state space, you must linearize a closed-loop system with at least one input u and one output y.

`mech_four_bar_lin` illustrates this reduction of independent DoFs: of the four revolute joints, only one is an independent DoF, which can be taken as any one of the revolutes. This model defines workspace variables in order to configure the initial geometry of lengths and angles (expressed in the model in meters and radians, respectively). Run the model in Forward Dynamics mode.



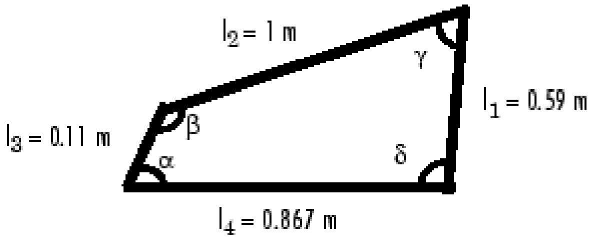


Consider a strategy to linearize the model about the four bar's (stable) natural equilibrium. You first find the natural equilibrium configuration, which is best accomplished by analyzing the loop constraints, making a guess, and then using the `trim` command to determine the equilibrium exactly. After choosing a system input and output, you then linearize the system.

Chapter 2, "Building and Visualizing Simple Machines" presents this system in detail, in the section "Four Bar Mechanism" on page 2-36. The preceding sections of this chapter, "Inverse Dynamics Mode with a Double Pendulum" on page 8-8 and "Constrained Trimming of a Four Bar Machine" on page 8-26, discuss the inverse dynamics and trimming of the four bar system.

Analyzing the Four Bar Geometry and Closed-Loop Constraint

You can determine the constraints and independent DoFs of the four bar with geometric and trigonometric identities applied to its quadrilateral shape. The lengths of the bars are l_1, l_2, and l_3, with the fixed base having length l_4.



The four joint angles satisfy α + β + γ + δ = 2π. Imagine cutting the quadrilateral diagonally from the α to the γ vertices, then from the β to the δ vertices. The law of cosines applied to these diagonals and the triangles so formed results in two constraints:

$$l_1^2 + l_2^2 - 2l_1l_2\cos\gamma = l_3^2 + l_4^2 - 2l_3l_4\cos\alpha$$

$$l_2^2 + l_3^2 - 2l_2l_3\cos\beta = l_1^2 + l_4^2 - 2l_1l_4\cos\delta$$

The four angles are thus subject to three constraints. Choose α (the crank angle) as the independent DoF. You can determine β, γ, and δ from α by inverting the constraints.

Making an Equilibrium Guess

First guess the natural equilibrium. An obvious guess for the natural equilibrium is for the crank (Bar 3) to point straight down, $\alpha = -90^o$.

1 Use the quadrilateral constraints to find

$\beta = 310.1^o$, $\gamma = 60.3^o$, and $\delta = 79.6^o$

2 Redefine the workspace angles to these values (converted to radians).

```
alpha = -90*pi/180; beta = 313.2*pi/180; gamma = 60.3*pi/180;
delta = 76.5*pi/180;
beta2 = pi - gamma - delta; delta2 = pi - delta;
```

3 Update the diagram and run the model again. This configuration is not the natural equilibrium, but it is close.

Determining the Natural Equilibrium with trim

Now find the natural equilibrium exactly by trimming the four bar in a manner similar to "Constrained Trimming of a Four Bar Machine" on page 8-26, but without external torque actuation. Revolute1 is already manually configured to be the cut joint in the closed loop, ensuring the DoF represented by Revolute4 is not eliminated from state space when the loop is cut.

1 Set the analysis mode to `Trimming`. SimMechanics inserts a subsystem and an output block that outputs a four-component signal representing the mechanical constraints resulting from the closed loop.

2 Use `mech_stateVectorMgr` to obtain the system's state vector:

```
StateManager = ...
   mech_stateVectorMgr('mech_four_bar_lin/Ground_2');
StateManager.StateNames
ans =
    'mech_four_bar_lin/Revolute2:R1:Position'
    'mech_four_bar_lin/Revolute3:R1:Position'
    'mech_four_bar_lin/Revolute4:R1:Position'
    'mech_four_bar_lin/Revolute2:R1:Velocity'
    'mech_four_bar_lin/Revolute3:R1:Velocity'
    'mech_four_bar_lin/Revolute4:R1:Velocity'
```

Revolute1 is the cut joint and is missing from the list. States 1, 2, and 3 are the revolute 2, 3, and 4 angles, respectively; while states 4, 5, and 6 are the revolute 2, 3, and 4 angular velocities, respectively.

3 Set up the necessary trimming vectors:

```
x0 = [0;0;0;0;0;0]; ix = [];
u0 = []; iu = [];

y0 = [0;0;0;0];
iy = [1;2;3;4];
dx0 = [0;0;0;0;0;0];
idx = [3;6];
```

The `x0` vector tells the `trim` command to start its search for the equilibrium with the four bar in its initial configuration (the equilibrium guess you entered into the workspace previously) and with zero angular velocities.

The index vector ix sets the states that, in the actual equilibrium, should keep the values specified in x0. Here there are none.

The u0 and iu vectors specify system inputs, but there are none.

The y0 vector sets the initial values of the constraint outputs to zero. The index vector iy requires that the constraint outputs at equilibrium be equal to their initial values (0). This ensures that the solution satisfies the mechanical constraints.

The dx0 vector specifies the initial state derivatives. The initial derivatives of the angles (i.e., the angular velocities) and of the angular velocities (i.e., the angular accelerations) are set to zero. The index vector idx specifies that the velocity and acceleration of Revolute4 in the natural equilibrium must vanish. It is not necessary to constrain the derivatives of the other states because the system has only one independent DoF.

4 Now trim the system:

```
[x,u,y,dx] = ...
    trim('mech_four_bar_lin',x0,u0,y0,ix,iu,iy,dx0,idx);
```

The u vector is empty. The components of y and dx vanish, within tolerances, indicating that in equilibrium, respectively, the mechanical constraints are satisfied and the state derivatives vanish. The last three components of x vanish, indicating zero angular velocities at equilibrium. The first three components of x represent the natural equilibrium angles (in radians), measured as deviations from the initial configuration. The Revolute4 angle is -0.2395 rad = -13.7° from the starting point.

From x, you can calculate all the angle values. The natural equilibrium is α_{eq} = -90° - 13.7° = -103.7°, β_{eq} = 310.1° + 13.0° = 323.1°, γ_{eq} = 60.3° + 2.5° = 62.8°, and δ_{eq} = 360° - α_{eq} - β_{eq} - γ_{eq} = 74.7°.

Linearizing the Model at the Natural Equilibrium

You can now linearize the system at this trim point.

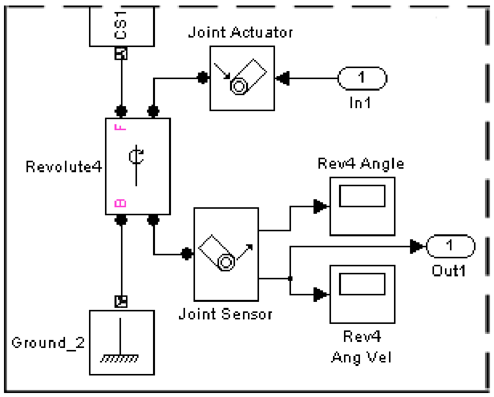
1 Reset the angles in your workspace to the natural equilibrium point:

```
alpha = alpha + x(3); beta = beta + x(2); gamma = gamma + x(1);
delta = 2*pi - alpha - beta - gamma; beta2 = pi - gamma - delta;
```

```
delta2 = pi - delta;
```

2 Change the analysis mode back to `Forward Dynamics` and update the diagram. Run the model to check that the mechanism indeed does not move.

3 To obtain a nontrivial linearized model, you need at least one input and one output. Connect a Joint Actuator to Revolute4 to actuate it with a torque. Then insert Simulink Inport and Outport blocks to input the torque and measure the angular velocity.



4 Set the input torque to zero and the initial state to the model's initial configuration, the natural equilibrium:

```
u = 0; x = [0;0;0;0;0;0];
```

5 Linearize the model and use `minreal` to eliminate the redundant states:

```
[A,B,C,D] = linmod('mech_four_bar_lin',x,u);
[a,b,c,d] = minreal(A,B,C,D);
```

leaving two states, α and $d\alpha/dt$. The component $a(2,1) = -80.0873 < 0$, indicating that this natural equilibrium is stable. The linearized motion is governed by $d^2\alpha/dt^2 = a(2,1)*\alpha$.

For More Information About State Space and Linearization

See "Open-Topology Linearization: Double Pendulum" on page 8-34 for more about the linearized state space representation.

Case Studies

SimMechanics and Simulink form a powerful basis for advanced controls applications: trimming and linearizing motion, designing controllers, converting plant and controller models to code, and simulating controller and plant on dedicated hardware. This chapter is a connected set of case studies illustrating these methods. For its example system, the studies use the Stewart platform, a moderately complex, six degree-of-freedom positioning system.

"Modeling a Stewart Platform in CAD" on page 7-46 presents a related example, converting a Stewart platform computer-aided design assembly into a SimMechanics model.

Overview of Case Studies

This section explains

- "Understanding the Stewart Platform" on page 9-3
- "About the Case Studies" on page 9-3
- "Products Needed for the Case Studies" on page 9-4

The "References" on page 9-5 explain plant analysis and control design for mechanical systems and how to generate and deploy code for machines and their controllers.

Understanding the Stewart Platform

The chapter starts with a summary of the Stewart platform and the models.

- "Introducing the Stewart Platform" on page 9-7
- "Modeling the Stewart Platform in SimMechanics" on page 9-13

About the Case Studies

The studies use Stewart platform models and a suite of products to help you carry out advanced mechanical design and simulation tasks. The tasks are grouped into the following case studies. In them, you make use of powerful techniques such as M-file scripts, linked libraries, and configurable subsystems to simplify the task of defining a complex simulation in Simulink and SimMechanics.

- "Trimming Through Inverse Dynamics" on page 9-24
- "Designing Controllers" on page 9-35
- "Simulating with Code" on page 9-69
- "Hardware in the Loop" on page 9-79

Structure and Dependencies

The studies begin with motion analysis and control design and end with code generation and hardware implementation.

The first study is important for a deeper understanding of trimming and might be useful before attempting the second. The last two studies are connected, and you should work through the third before attempting the fourth.

Caution SimMechanics code generation is intended for rapid prototyping and hardware-in-the-loop applications. It is not intended for use as production code in embedded controller applications.

Case Study Files

Each study has an associated set of demo files and is based on an appropriate variant model of the Stewart platform.

Saving Intermediate Stages of Work

It is recommended that you complete each case study in one session. If you cannot, for lack of time, you should periodically save your intermediate results from your workspace to a MAT-file.

Products Needed for the Case Studies

The case studies of this chapter require MATLAB, Simulink, and SimMechanics throughout. You should have a good working knowledge of all three.

In addition, you use several specialized products for specific tasks in each study. You should have at least a beginner's level experience with each.

Product	Required for Case Study
Control System Toolbox	"Trimming Through Inverse Dynamics" (one part) "Designing Controllers"
Robust Control Toolbox	"Designing Controllers" (last part)
Simulink Control Design	"Designing Controllers"

Product	Required for Case Study
Real-Time Workshop	“Simulating with Code” and “Hardware in the Loop”
xPC Target	“Hardware in the Loop”

References

[1] Stewart, D., “A platform with six degrees of freedom,” *Proc. Inst. Mech. Eng.,* Vol. 180, part I(15), 1965-1966, pp. 371-386.

[2] Wilkie, J., M. Johnson, and R. Katebi. *Control Engineering: An Introductory Course.* Hampshire, United Kingdom: Palgrave/St. Martin’s Press, 2002.

[3] Smith, N., and J. Wendlandt, “Creating a Stewart Platform Model Using SimMechanics,” *MATLAB Digest* 10(5) (September 2002), http://www.mathworks.com/company/newsletters/digest/sept02/stewart.html. This case study includes only the actual leg trajectory in the derivative term, not the reference trajectory. The derivative term acts in that case like damping, not as error correction.

[4] Parsons, L., and J. Glass, “Recommendations for Creating Accurate Linearized Models in Simulink,” *MATLAB Digest* 12(4) (July 2004), http://www.mathworks.com/company/newsletters/digest/july04/linmodels.html.

[5] Glover, K., and D. C. McFarlane. “Robust stabilisation of normalised coprime factor plant descriptions with H-infinity bounded uncertainty.” *IEEE Trans. on Automatic Control,* Vol. 34, 1989, pp. 821-830.

[6] Georgiou, T. T., and M. C. Smith. “Optimal robustness in the gap metric.” *IEEE Trans. on Automatic Control,* Vol. 35(6), 1990, pp. 673-687.

[7] Janka, R. S., *Specification and Design Methodology for Real-Time Embedded Systems* (New York/Berlin: Springer-Verlag, 2002).

[8] Li, Q., and C. Yao, *Real-Time Concepts for Embedded Systems* (Gilroy, California: CMP Books, 2003).

[9] Ledin, J., M. Dickens, and J. Sharp, “AIAA 2003: Single Modeling Environment for Constructing High-Fidelity Plant and Controller Models,” American Institute of Aeronautics and Astronautics, 2003, http://www.mathworks.com/products/xpctarget/technicalliterature.html.

Introducing the Stewart Platform

This section explains

Origin and Uses of the Stewart Platform

The Stewart platform is a classic design for position and motion control, originally proposed in 1965 as a flight simulator, and still commonly used for that purpose [1]. Since then, a wide range of applications have benefited from the Stewart platform. A few of the industries using this design include aerospace, automotive, nautical, and machine tool technology. The platform has been used to simulate flight, model a lunar rover, build bridges, aid in vehicle maintenance, design crane hoist mechanisms, and position satellite communication dishes and telescopes, among other tasks.

Characteristics of the Stewart Platform

The Stewart platform has an exceptional range of motion and can be accurately and easily positioned and oriented. The platform provides a large amount of rigidity, or stiffness, for a given structural mass, and thus provides significant positional certainty. The platform model is moderately complex, with a large number of mechanical constraints that require a robust simulation.

Most Stewart platform variants have six linearly actuated legs with varying combinations of leg-platform connections. The full assembly is a parallel mechanism consisting of a rigid body top or mobile plate connected to an immobile base plate and defined by at least three stationary points on the grounded base connected to the legs.

The Stewart platform used here is connected to the base plate at six points by universal joints. Each leg has two parts, an upper and a lower, connected by a cylindrical joint. Each upper leg is connected to the top plate by another universal joint. Thus the platform has 6*2 + 1 = 13 mobile parts and 6*3 = 18 joints connecting the parts.

Counting the Degrees of Freedom

The standard Stewart platform design has six independent degrees of freedom (DoFs). The mobile plate, if disconnected from the legs and thus unconstrained, also has six DoFs. The Stewart platform therefore exactly reproduces the possible motion of a free plate, but with the added benefit of stable and precise positioning control.

Here are two ways to count the Stewart platform DoFs.

- "Counting Degrees of Freedom on Bodies in Space" on page 9-8 starts with the disassembled platform parts as physical bodies in space.
- "Counting Degrees of Freedom as Joint Primitives" on page 9-9 starts with the platform represented as connected Body and Joint blocks.

Counting Degrees of Freedom on Bodies in Space

Start with the disassembled Stewart platform parts as unconstrained moving bodies. As you assemble the platform, you constrain the bodies as you connect them with joints. The base plate is immobile.

This approach is *not* the way that SimMechanics counts DoFs. See "Counting Degrees of Freedom as Joint Primitives" on page 9-9.

Bodies with DoFs. Each free body in space has six DoFs. Only after you attach them to one another with joints are they no longer able to move freely.

Joints as Constraints. Connecting bodies with joints constrains the two bodies so they can no longer move freely relative to one another.

For example, a universal joint connection allows two rotational DoFs, but imposes four constraints, three translational (positional) and one rotational.

Assembling the Stewart Platform Parts. Start assembling the Stewart platform. Each joint attachment simultaneously connects and constrains the bodies. In all, each leg imposes 12 constraints on itself and the top plate.

- The universals connecting the lower legs to the base plate impose four constraints:
  - Three positional, requiring two points to be collocated

  - One rotational, preventing the lower leg from rotating about its long axis (with respect to the immobile base)
- The cylindricals connecting the upper to the lower legs impose four constraints:
  - Two positional, allowing the two legs to slide along the long axis but not translate in the other two directions
  - Two rotational, allowing the upper leg to rotate about the long axis (with respect to the lower leg) but not rotate about the other two directions
- The universals connecting the top plate to the upper legs impose four constraints:
  - Three positional, requiring two points to be collocated
  - One rotational, preventing the upper leg from rotating about its long axis (with respect to the top plate — *not* with respect to the lower leg)

Obtaining the Independent DoFs. The Stewart platform has 13 moving bodies. With no constraints, the disassembled Stewart platform has 13*6 = 78 DoFs.

Assembling the parts imposes 12*6 = 72 constraints. Therefore, the Stewart platform has 13*6 - 12*6 = 6 independent DoFs.

Counting Degrees of Freedom as Joint Primitives

Start with the Stewart platform as an assembled SimMechanics model.

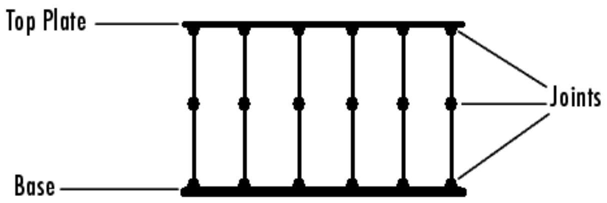
Bodies Without DoFs. In SimMechanics, a Body carries no DoFs. Instead, pairs of Bodies are connected by Joints, which express the motions of one Body relative to another.

Six Grounds represent the base plate. Thirteen Bodies represent the moving parts.

Joints Primitives as DoFs. Each Joint contains primitives. Translational and rotational primitives each express one DoF. (These are the only primitive types used here.) The Stewart platform model contains 18 Joints containing 6*6 = 36 primitives, of which 30 are rotational and 6 are translational.

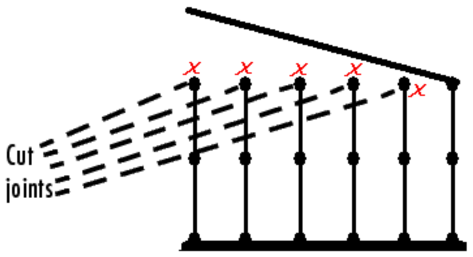
- Six Universal joints connecting the lower legs to the base. Each contains two rotational primitives.
- Six Cylindrical joints connecting the lower to the upper legs. Each contains a rotational and a translational primitive.
- Six Universal joints connecting the upper legs to the top plate. Each contains two rotational primitives.

Counting Loops. The Stewart platform legs form six loops, but only five are independent. You can obtain a topologically equivalent platform by flattening the top plate and base into lines and counting five loops that have the six legs as sides:



Cutting the Stewart Platform Joints and Deriving the Tree. To simulate a machine with closed loops (like the Stewart platform), SimMechanics replaces it internally with an equivalent machine (the ***spanning tree*** *on page Glossary-22*) obtained by cutting all the independent loops at one Joint per loop and replacing the cut Joints with (invisible) equivalent constraints.

Obtain the spanning tree by cutting five of the six upper Universals. This cutting is just enough to open all loops but not disconnect the machine into disjoint parts. The tree contains 13 (uncut) Joints constituting 6*(2+2) + 2 = 26 DoFs.



Imposing the Cutting Constraints and Deriving the Independent DoFs. To complete the conversion of the closed-loop machine into an equivalent tree, impose constraints to replace the cut Joints. There are 20 such constraints. Each constraint is equivalent to reattaching a cut Joint and analyzes into five sets of

- Three positional constraints, requiring two points to be collocated
- One rotational constraint, preventing the upper leg from rotating about its long axis relative to the top plate

Thus reattaching the cut Joints to reassemble the platform leaves 26 - 5*4 = 6 independent DoFs.

Representing the Independent Degrees of Freedom

These six independent DoFs are usually taken to be the six leg lengths. Every other DoF identified here is now dependent on these six lengths. Each time you change a length, the universals connecting the legs to the base and top plate rotate, the top plate shifts and rotates, and the upper legs rotate about their long axes.

Alternatively and equivalently, you can take the six independent DoFs to be the six DoFs of the top, mobile plate. By connecting the top plate, you replace the six independent DoFs of an unconstrained plate with six DoFs under the precise and stable control of the six-leg positioning system.

The six DoFs of the connected top plate are not in addition to the leg-length DoFs. They are just an equivalent, replacement description of the same six independent DoFs. The whole platform system, once fully connected, always has exactly six independent degrees of freedom.

For More About Bodies, Joints, Degrees of Freedom, and Topology

Chapter 4, “Modeling Mechanical Systems” shows how Bodies and Joints in SimMechanics represent bodies and DoFs. See especially these sections:

- “Modeling Joints” on page 4-20
- “Checking Model Validity” on page 4-74

Chapter 5, “Running Mechanical Models” explains the steps SimMechanics executes to analyze and simulate a machine. See especially these sections:

- “How SimMechanics Works” on page 5-15
- “Troubleshooting Simulation Errors” on page 5-17

Modeling the Stewart Platform in SimMechanics

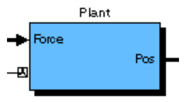
This section explains the essential details of modeling the Stewart platform with SimMechanics.

To understand these points, use any top-level model from the case studies of this chapter, except the models of "Trimming Through Inverse Dynamics" on page 9-24. These are different because they lack a controller subsystem and consist of a plant model alone.

The control design model, `mech_stewart_control`, is this section's example.

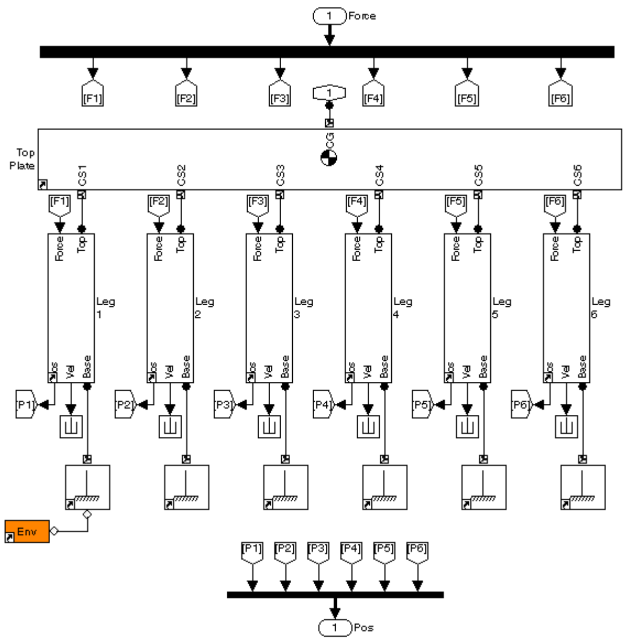
Modeling the Physical Plant

In three of the case studies, a larger control system model contains a Plant subsystem that incorporates the platform.



Viewing the Platform Model

The Plant subsystem models the Stewart platform's moving parts, the legs and top plate. Open this subsystem.



Stewart Platform Model in SimMechanics (Controls Study)

Each of the legs is an instance of a library block located in another library model, `mech_stewartplatform_leg` or `mech_stewart_control_equil_leg`.

1 Select one of the leg subsystems and right-click. Select **Link Options**, then **Go To Library Block**, to open this library.

2 Open the masked library block, Leg Subsystem, and the individual Body and Joint blocks that make up a whole leg.

3 Now close the blocks, subsystems, and linked libraries and return to the top-level model.

Modeling Controllers

Except in the “Trimming Through Inverse Dynamics” on page 9-24 study, the Stewart platform models contain controllers imposing actuating forces that guide the platform’s motion to follow as closely as possible a *nominal* or *reference* trajectory. Implementing a controller requires computing the motion errors, the difference of the reference and actual motions of the platform. All the case study models use proportional-integral-derivative (PID) control.

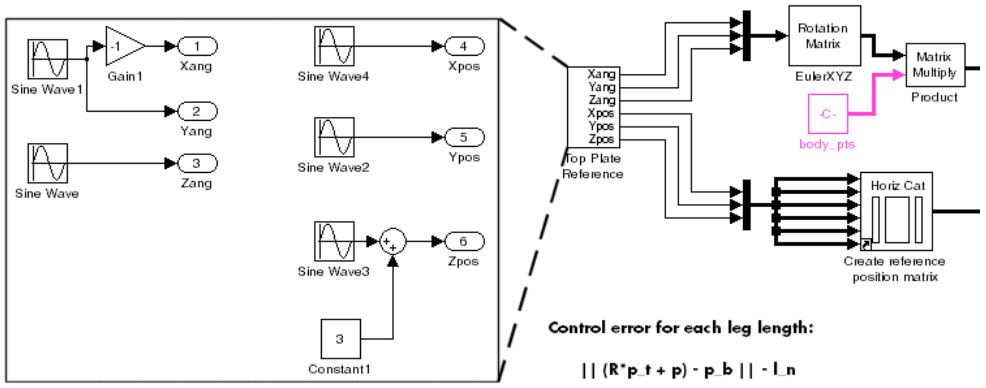
Generating the Reference Trajectory

Each model controller requires a reference trajectory.

1 Open the Leg Reference Trajectory subsystem.

This set of blocks generates the set of six leg lengths, as functions of time, corresponding to a desired trajectory for the top plate.

2 Open the subsystem called Top Plate Reference. This set of blocks generates a reference trajectory in terms of linear position and three orientation angles, as a function of time. The workspace variable `freq` sets the frequency of the reference motion.



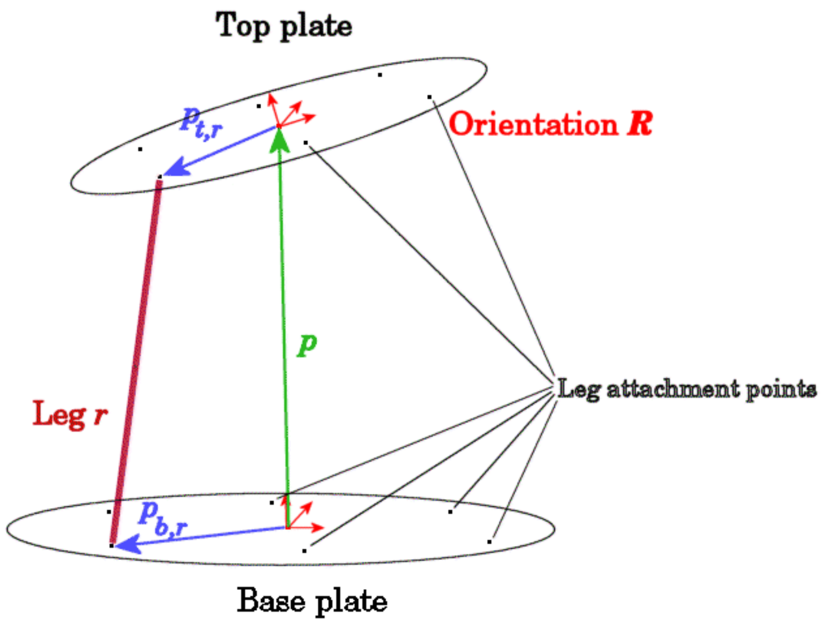
Stewart Platform Reference Trajectory Subsystem (Controls Study)

- The reference trajectory provided uses sinusoidal functions of time to define the rotational and translational degrees of freedom.
- If you want, you can design and implement another reference trajectory of your choosing and replace this sub-subsystem.

Whatever comes out of Top Plate Reference, the subsystem Leg Reference Trajectory assumes the translational position/three-angle form for the top plate. The rest of the Leg Reference Trajectory subsystem transforms these six degrees of freedom (DoFs) into the equivalent set of six DoFs expressed as the lengths of the six platform legs. The reference trajectory output of the subsystem is a six-vector of these leg lengths.

Finding the Motion Error

The actuating force on leg *r* is a function of the motion error. The error requires finding the instantaneous length of each leg from the positions of that leg's top and bottom connection points.



Defining the Length of a Stewart Platform Leg

The motion error is the difference of the desired or reference length of the leg and its instantaneous or actual length:

$$\begin{aligned}\text{Error} = E_r &= \text{reference length of leg} - \text{actual length of leg}\\ &= L_{\text{traj},r}(t) - |(R \cdot \boldsymbol{p}_{\text{t},r}) - \boldsymbol{p}_{\text{b},r}|\end{aligned}$$

The reference length $L_{\text{traj}}(t)$ is given as a function of time by the output of the Leg Reference Trajectory subsystem. The vectors $\boldsymbol{p}$, $\boldsymbol{p}_{\text{t,r}}$, and $\boldsymbol{p}_{\text{b,r}}$ are defined in the preceding figure. The orthogonal rotation matrix R specifies the orientation of the top plate with respect to the bottom.

The Standard PID Controller and Its Control Law

All the Stewart platform models use a simple PID controller and Joint Sensor blocks to measure motion. The simplest implementation of trajectory control is to apply forces to the plant proportional to the motion error. PID feedback is a common form of linear control.

A PID control law is a linear combination of a variable detected by a sensor, its time integral, and its first derivative. This Stewart platform's PID controller uses the leg position errors E_r and their integrals and velocities. The control law for each leg r has the form:

$$\mathrm{F}_{\text{act},r} = K_p \mathrm{E}_r + K_i \int_0^{\mathrm{t}} \mathrm{E}_r \, dt + K_d (d\mathrm{E}_r/dt)$$

The controller applies the actuating force $F_{\text{act,r}}$ along the leg.

- If E_r is positive, the leg is too short, and $F_{\text{act,r}}$ is positive (expansive).
- If E_r is negative, the leg is too long, and $F_{\text{act,r}}$ is negative (compressive).
- If E_r is zero, the leg has exactly the desired length, and $F_{\text{act,r}}$ is zero.

The real, nonnegative K_p, K_i, and K_d are, respectively, the proportional, integral, and derivative gains that modulate the feedback sensor signals in the control law:

- The first term is proportional to the instantaneous leg position error, or deviation from reference.
- The second term is proportional to the integral of the leg position error.
- The third term is proportional to the derivative of the leg position error.

The result is $F_{\mathrm{act,r}}$, the actuator force (input) applied by the controller to the legs. The proportional, integral, and derivative terms tend to make the legs' top attachment points $\boldsymbol{p}_{\mathrm{t,r}}$ follow the reference trajectories by suppressing the motion error.

For More About Controllers

The case study, "Designing Controllers" on page 9-35, discusses controlling platform motion in greater detail. In that study, you also use an H-infinity controller, as well as use transfer functions to take motion derivatives.

In addition, consult "References" on page 9-5.

Initializing the Stewart Platform in SimMechanics

When creating the physical components of the Stewart platform model with SimMechanics and the control blocks with Simulink, you must define the geometry of its initial state and the mass parameters of the Stewart platform bodies. Although each case study in this chapter uses a variant model, all initialize the platform and controller configuration in a common way.

Geometric, mass, dynamical, and controller information is specified in the block dialogs by referencing variables in your MATLAB workspace. An M-file script accompanies the Stewart platform models and sets these values.

Running this script configures the blocks in their starting geometric state, with the correct mass properties for the bodies. When you open it, each model uses the same initialization M-file as a pre-load function. To see this setting,

1. Go to the **File** menu and select **Model Properties**.
2. Then in the dialog, select the **Callbacks** tab and find the **Model pre-load function** field.

Stewart Platform Initialization M-File

File	Purpose
`mech_stewart_studies_setup`	M-file script to fill the workspace with geometric, dynamical, and controller data.
`inertiaCylinder`	M-file function called by `mech_stewart_studies_setup`. Computes the principal inertias of a solid cylinder.

Body and Joint Geometric Configuration

The script first defines basic angular unit conversions and axes. The World coordinate system (CS) is located at the center of the immobile base plate. The connection points on the base and top plate are defined with respect to World. These definitions include the offset angle of 60 degrees between the base and top plates, the radii of both the base and top plates, the initial position height of the top plate, and the vectors pointing along the legs. The array of top points is permuted so that the same index represents the top and bottom connection points for the same leg.

The script calculates the revolute and cylindrical axes used in the joint blocks of the leg subsystems. There are two revolute axes for each Universal joint that connects an upper leg to the top plate, one cylindrical and one revolute axis for the linear motion of the Cylindrical joint connecting upper and lower legs, and two revolute axes for each Universal block that connects a lower leg to the base plate. The script then configures all 13 moving bodies by defining coordinate systems at the center of gravity (CG) of each.

The top plate's home configuration is symmetric equilibrium: flat, with equal leg lengths specified by the workspace vector `leg_length`.

Body Mass Properties

The script defines the mass properties of all bodies. These comprise the inertia tensors and masses for the top plate, the bottom plate, and the legs. The mass properties calculation assumes that the platform is made with steel. The script calls the function `inertiaCylinder` to calculate the inertia tensors and masses of the legs and the top and base plates, given the material density, the length and inner and outer radii of the leg cylinders, and the thicknesses and radii of the top and base plates.

Motion Constants, Controller Parameters, and Initial Condition

In its final steps, the script defines motion and control constants as workspace variables: motion frequency, derivative filtering cutoff, leg actuator force saturation, and controller gains. Each case study model uses some or all of these constants, which you can change as desired.

Real force actuators are saturate at a specific force level. The Force Saturation block limits the actuating force to the value of the workspace variable `force_act_max`.

The integral (I) part of the PID controller exhibits an extended response time whose overall effect is controlled by the ratio of K_i to K_p. The Integrator for the I part has a nonzero **Initial condition** field, specified by the workspace variable `initCondI`, adjustable to compensate for initial transient behavior. The script initializes its value to

```
(upper_leg_mass+lower_leg_mass+(top_mass*1.3/6))*9.81/Ki
```

corresponding to the leg forces in symmetric equilibrium.

Motion and Filtering Constants

Dynamical Feature	**Workspace Variable**	**Associated Natural Frequency**	**Associated Time Scale**
Top plate motion	`freq` = π rad/s	`freq`/2π = 0.5 Hz	2π/`freq` = 2 s
Filtered derivative cutoff	`A` = 100*freq = 100π rad/s	`A`/2π = 50 Hz	2π/`A` = 0.02 s

PID Controller Constants

Dynamical Constant	**Workspace Variable**
Force saturation	`force_act_max` = 3e5 newtons (N)
Integral (I) gain	`Ki` = 1e4 (newtons/meter/second) (N/m-s)

PID Controller Constants (Continued)

Dynamical Constant	Workspace Variable
Proportional (P) gain	`Kp` = 2e6 newtons/meter (N/m)
Derivative (D) gain	`Kd` = 4.5e4 newtons-seconds/meter (N-s/m)

Identifying the Simulink and Mechanical States

For the purposes of SimMechanics motion analysis, you need to know the model's Simulink and mechanical states. These are distinct from the system's degrees of freedom (DoFs) and depend on the analysis mode you choose.

Pure Simulink States

If you use a controller or other subsystem made up of pure Simulink blocks with your Stewart platform, your model might contain Simulink states. For example, Integrator and Transfer Fcn blocks each have an associated state, and State-Space blocks can have many.

The default Stewart platform controller is a PID subsystem, which integrates six feedback signals and thus has six Simulink states. In the "Designing Controllers" on page 9-35 study, you can also choose to use the filtered derivative, which has 12 transfer functions and thus adds 12 Simulink states.

Mechanical States in Forward Dynamics Mode

A mechanical system modeled in SimMechanics with Joint blocks contains mechanical states distinct from Simulink states that include both joint position and velocity. In Forward Dynamics mode, the Stewart platform contains 52 tree states, of which 12 are independent.

The joints and their related DoFs are discussed in "Counting the Degrees of Freedom" on page 9-8.

Joint Primitives and States. Each Joint consists of one or more primitives. The position and velocity of a joint primitive each have a state. The Stewart platform has 36 joint primitives and thus potentially 72 states.

Cutting Joints and Obtaining the Tree States. Because the Stewart platform has closed topology, SimMechanics cuts five of the Joints to arrive at an equivalent open-topology or tree machine. These Joints are replaced internally by equivalent cutting constraints.

Five Universals and 5*2*2 = 20 joint primitives are eliminated this way. The equivalent open machine thus has 72 - 20 = 52 tree states.

Counting the Cutting Constraints. Not all these states are independent. There are 40 equivalent constraints that replace the cut Joints.

- Each cut Universal imposes one rotational and three position constraints.
- Each constraint also constrains the corresponding velocity.
- There are five cut Joints.

Thus there are 5*2*4 = 40 invisible constraints generated by the cutting.

Finding the Independent States. Thus the Stewart platform model has 52 - 40 = 12 independent mechanical states, corresponding to the six independent DoFs and their velocities.

Mechanical States in Trimming and Kinematics Modes

You can also analyze the Stewart platform's motion in inverse dynamics and locate steady-state operating points.

- Because the Stewart platform is a closed-loop system, you must simulate its inverse dynamics in the Kinematics mode.
- You can find operating points in the Trimming mode with the Simulink `trim` command.

In both the inverse dynamics and trimming cases, the Simulink states associated with the SimMechanics joint primitives are *not* the DoFs, but the *(invisible) joint-cutting constraints* that reduce the tree states to independent states. The state values measure how well the constraints are satisfied. A zero value means a constraint is satisfied perfectly.

In the mechanical part of the Stewart platform model, there are 52 tree states and 12 independent states. Thus SimMechanics counts 52 - 12 = 40 cutting constraints. In the Kinematics and Trimming modes, these 40 constraints are the mechanical states.

Open Topology and Inverse Dynamics Mode. If the Stewart platform had an open topology, you would simulate its inverse dynamics in Inverse Dynamics mode instead. However, there would be no closed loops, and SimMechanics would not cut any Joints. With no cutting constraints, an open topology machine has no states in Inverse Dynamics or Trimming mode.

For More About Mechanical States, Cutting Loops, and Analysis Modes

Learn more about SimMechanics states and loops in Chapter 4, “Modeling Mechanical Systems”:

- “Cutting Closed Loops” on page 4-36
- “Checking Model Validity” on page 4-74

Consult the `mech_stateVectorMgr` command reference as well.

For more about analysis modes, see

- “Simulating and Analyzing Mechanical Motion” on page 1-20.
- Chapter 8, “Analyzing Motion”.

Visualizing the Stewart Platform Motion

To view mechanical animation, consult “Starting SimMechanics Visualization” on page 6-2.

With the SimMechanics visualization window open, you can view the platform motion from different perspectives. View the platform in the *xy*-plane, from above. Then switch the view to the *xz*- or *yz*-plane.

The initial state of motion specified by the reference trajectory is slightly different from the home configuration and generates an initial transient.

Trimming Through Inverse Dynamics

Note This study requires Control System Toolbox at an optional step, "Finding the Minimal Realization of the Linearized Model" on page 9-32.

This case study finds a Stewart platform steady state with the Kinematics mode of SimMechanics. You specify motions and determine the forces and torques to produce those motions (the *inverse dynamics* problem). If you are not familiar with implementing inverse dynamics in SimMechanics, work through the "Finding Forces from Motions" on page 8-7 before attempting this case study.

Use the Inverse Dynamics and Kinematics modes for inverse-dynamic analysis of open- and closed-topology systems, respectively. The Stewart platform has a closed topology and thus requires the Kinematics mode to solve inverse dynamics. Once you have an operating point, you can linearize the motion.

- "What Is Trimming?" on page 9-24
- "Ways to Find an Operating Point" on page 9-25
- "Trimming in the Kinematics Mode" on page 9-25
- "Linearizing the Platform at the Operating Point" on page 9-29
- "Further Suggestions for Inverse Dynamics Trimming" on page 9-32

What Is Trimming?

Trimming a system means locating a configuration of its states with certain prior conditions imposed on the states and possibly their derivatives. In a mechanical context, it means imposing conditions on certain positions and velocities, then determining the remaining positions and velocities such that the entire state of the machine is consistent. A by-product of mechanical trimming is determination of the forces/torques necessary to produce the specified motion. These motion states constitute a *trim* or *operating point*. Trimming problems can have one solution, more than one, or none.

Pure inverse dynamics imposes prior motions on all degrees of freedom. Then all the states are determined. (The consistency of the motions is not guaranteed, but must be checked.) Only the forces/torques remain to be found.

Ways to Find an Operating Point

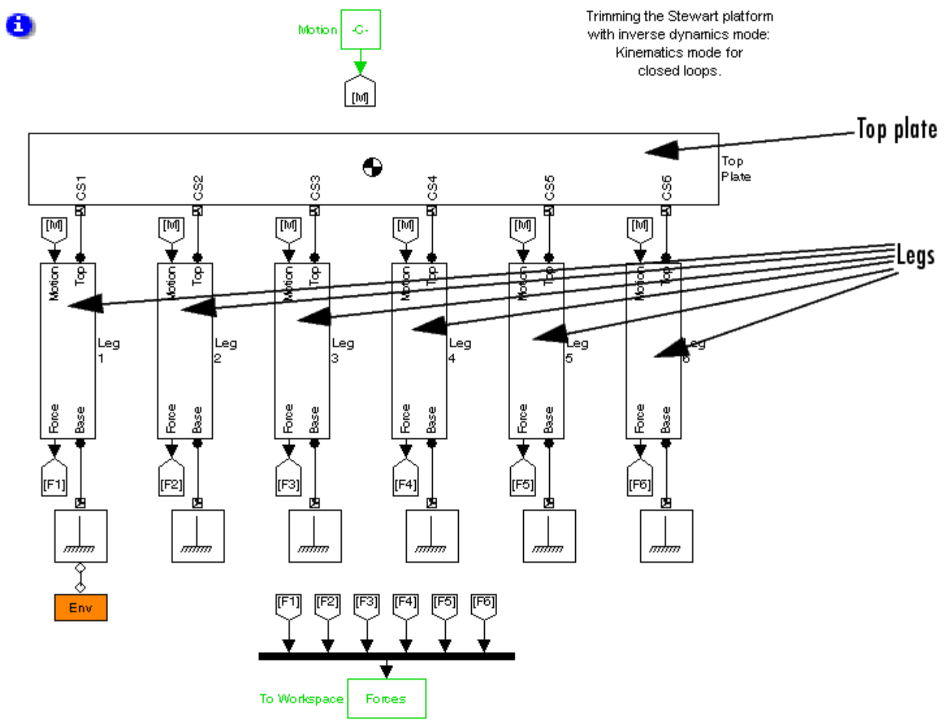
To find an operating point or steady state for a SimMechanics model,

- Use the `trim` command in Simulink. See "Trimming Mechanical Models" on page 8-18.
- Use the more powerful techniques provided by Control System Toolbox and Simulink Control Design. See "Designing Controllers" on page 9-35.
- Use the inverse dynamics modes of SimMechanics. You can manipulate the mechanical states of your model directly with motion actuation rather than manipulate them through Simulink.

Trimming in the Kinematics Mode

Here are the files needed for this case study. The models also call the initialization M-files. Open the first model.

File	Purpose
`mech_stewart_control_equil`	Kinematics model for determining Stewart platform force equilibrium
`mech_stewart_control_equil_leg`	Library model of Stewart platform leg for kinematic analysis
`mech_stewart_control_plant`	Forward dynamics model for linearizing the Stewart platform
`mech_stewartplatform_leg`	Library model of Stewart platform leg for forward dynamic analysis



Simulation Settings for Inverse Dynamics

The mech_stewart_control_equil model has some preset nondefault settings.

Configuration Parameters

Setting	Value
Solver > Simulation time > Stop time	`0.005 seconds`
Data Import/Export > Save to workspace	**Time** and **States** selected > `tout` and `xout`

Configuration Parameters (Continued)

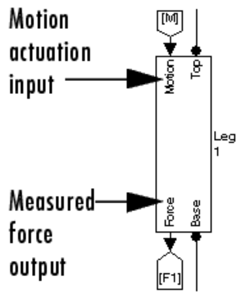
Setting	Value
SimMechanics > Diagnostics	**Mark automatically cut joints** selected
SimMechanics > Visualization	**Display machines after updating diagram** and **Show animation during simulation** selected

Machine Environment

Setting	Value
Parameters > Analysis mode	`Kinematics`
Parameters > Machine Dimensionality	`3D Only`
Constraints > Constraint solver type	`Machine Precision`
Constraints > Use robust singularity handling	Selected

Specifying the Motion

The six Stewart platform legs are instances of a basic leg saved in the `mech_stewart_control_equil_leg` library. It takes as inputs the motion actuation signals that specify position and velocity as a function of time. The position signals specify the platform's motion relative to the initial geometric configuration.



In mech_stewart_control_equil, the Motion subsystem specifies motion as trivial: zeroes for all six leg positions and velocities. That is, the model holds the platform still in its initial state.

Measuring the Steady-State Forces

Each Stewart platform leg outputs the computed leg force needed to maintain the motion specified by the motion actuation. These six measured forces are directed to your MATLAB workspace by the To Workspace block.

1 Open the To Workspace dialog.

   The output forces are stored in the vector variable `Forces`. The block retains the force vector only from the last time step.

2 Close the To Workspace dialog.

Running the Model and Obtaining the Outputs

Now run mech_stewart_control_equil.

1 Click **Start** and wait for the simulation to finish.

2 In your workspace, locate `tout` and `xout`. These are the time steps and the corresponding state values, respectively.

   In the Inverse Dynamics mode, there are 40 mechanical states counted by Simulink, associated with the mechanical constraints. Consult “Identifying the Simulink and Mechanical States” on page 9-21.

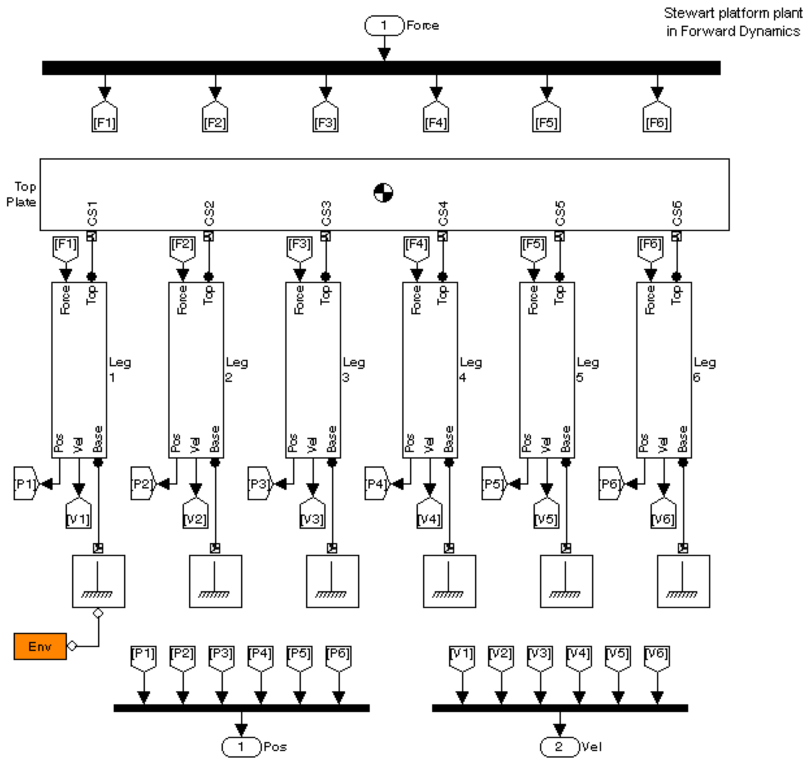
3 Locate `Forces` in the workspace. These are the six force values along each leg to maintain the platform still against falling in gravity. The values are positive (expansive) along the legs.

Linearizing the Platform at the Operating Point

Knowing the steady-state forces needed to keep the platform still, you now linearize another version of the model, mech_stewart_control_plant. It has settings similar to mech_stewart_control_equil, except that:

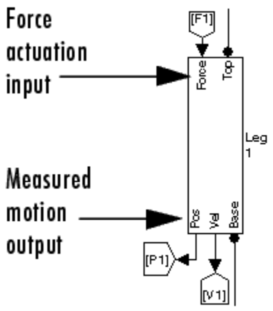
- The **Analysis mode** is set to `Forward Dynamics`.
- The simulation time is 10 seconds.
- **Time** and **Output**, `tout` and `yout`, respectively, are saved to workspace.

Open the `mech_stewart_control_plant` model.



- The six legs are instances of the mech_stewartplatform_leg library. This leg takes force as an input and outputs position and velocity, as appropriate for forward dynamics.
- The standard model input variable is `u`. The force vector signal is a model input.
- The position and velocity vector signals are model outputs. The **Data Import/Export** output variable is `yout` and will appear in your workspace assigned with data after you simulate.

Close the model.



Linearizing the Forward Dynamics Model

You can simulate the mech_stewart_control_plant model without opening it.

1 At the command line, enter

```
nomForces = Forces'; % Transpose the force vector
```

2 Linearize the model by entering

```
[A,B,C,D] = ...
   linmod('mech_stewart_control_plant',[],nomForces);
```

The arguments are, in order,

- Model name
- Model state vector (not used)
- Model input vector `u = nomForces`

These (unreduced) output matrices are the standard state-space representation of a linearized model. The space is defined by x, u, and y, the state, input, and output vectors, respectively.

$$\begin{aligned} \mathrm{d}\boldsymbol{x}/\mathrm{dt} &= A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u} \\ \boldsymbol{y} &= C \cdot \boldsymbol{x} + D \cdot \boldsymbol{u} \end{aligned}$$

There are 52 states, 6 inputs, and 12 outputs. Thus A, B, C, D have dimensions 52-by-52, 52-by-6, 12-by-52, and 12-by-6, respectively. Not all these matrix entries are independent.

Finding the Minimal Realization of the Linearized Model

Note This step requires Control System Toolbox.

Of the 52 mechanical states, the Stewart platform has only 12 independent states, corresponding to six degrees of freedom (DoFs). Each DoF corresponds to one position and one velocity.

To eliminate the redundant states, enter

```
[a,b,c,d] = ...
   minreal(A,B,C,D);
40 states removed.
```

at the command line. The a, b, c, d matrices are reduced in size to 12-by-12, 12-by-6, 12-by-12, 12-by-6, respectively.

For More About Linearization and State Space

See "Open-Topology Linearization: Double Pendulum" on page 8-34 and the Simulink documentation.

Further Suggestions for Inverse Dynamics Trimming

"Trimming in the Kinematics Mode" on page 9-25 and "Linearizing the Platform at the Operating Point" on page 9-29 present the simplest possible trimming scenario:

- All six degrees of freedom (DoFs) are determined by prior specification of positions and velocities. These are the inputs to the problem. The outputs are the forces necessary to maintain the specified motion. The simulation solves a pure inverse dynamics problem.
- The actual motion actuation signals require the platform to hold still relative to its initial geometric configuration.

General Trimming Conditions: Mixed Dynamics

In a more typical trimming problem, you specify some of the DoFs by motion actuation and leave the others free to respond to forces/torques. Such a scenario is a *mixed* dynamics problem. In SimMechanics, you can solve such problems in

- Forward Dynamics mode, where the tree states (DoFs corresponding to uncut Joints) are the mechanical states
- Kinematics mode (closed topology), where the cutting constraints that replace the cut Joints constitute the mechanical states
- Inverse Dynamics (open topology), where there are no mechanical states

Complementarity of Inverse and Forward Dynamics

Actuate DoF with	**Sense on DoF**
Forces/torques	Motions
Motions	Forces/torques

If you want to solve such a problem for the Stewart platform, you need to

- Use a library leg with
  - Force input
  - Motion output

  for each leg simulated in forward dynamics. You actuate it with a force and measure its motion. Use the mech_stewartplatform_leg block library.
- Use a library leg with
  - Motion input
  - Force output

  for each leg simulated in inverse dynamics. You actuate it with a motion and measure the corresponding force. Use the mech_stewart_control_equil_leg block library.

Using the Operating Point to Linearize a Model

The steady-state outputs are in turn the inputs for linearization.

Complementarity of Trimming and Linearization

Trimming Output Becomes...	...Linearization Input
Measured motions become...	...Motion actuation signals
Measured forces/torques become...	...Force/torque actuation signals

To carry out a linearization of your system,

1. Create a variant model in Forward Dynamics mode that takes
   - The steady-state forces as linearization input force actuation
   - The steady-state motions as linearization input motion actuation
2. Linearize with `linmod`.

   ```
   linmod('forward_dynamics_model_to_linearize', state, input)
   ```

   This command can feed model inputs into the linearized simulation as a command argument. See the command reference for more details.

Designing Controllers

Note This case study assumes some knowledge of control systems. In addition to Simulink and SimMechanics, parts of the study use these products:

- Control System Toolbox
- Simulink Control Design
- Robust Control Toolbox

You should have some experience with these tools before proceeding.

To understand trimming better, work through "Trimming Through Inverse Dynamics" on page 9-24.

A classic engineering problem is the design of *controllers* for a physical system, the *plant* [2]. SimMechanics can model a complex mechanical system and helps you design and implement a control system for the plant, in conjunction with Simulink and related control design products.

In this study, you use SimMechanics to model the plant and analyze and synthesize controllers. You explore a basic challenge of control design, the tradeoff between responsiveness and stability, by implementing first a simple controller, then a more complex and robust one [3].

Case Study Tasks

The first set of tasks implements the Stewart platform control system with the standard preoptimized PID controller.

- "A First Look at the Stewart Platform Control Model" on page 9-38 gives you an overview of the Stewart platform and controller model used in this study.
- "Improper and Biproper PID Controllers" on page 9-40 shows how the simple proportional-integral-derivative (PID) controller works and how to make it more realistic with a filtered derivative.

- “Analyzing the PID Controller Response” on page 9-44 shows you how to exploit classical control techniques to analyze the PID controller.

In the next group of tasks, you create and optimize a new PID controller.

- “Designing a New PID Controller” on page 9-47 starts the creation of a new PID controller *ab initio*.
- “Trimming and Linearizing the Platform Motion” on page 9-50 locates a steady state and linearizes the platform’s motion about this equilibrium.
- “Improving the New PID Controller” on page 9-56 uses the linearized platform dynamics to optimize the new PID controller.

The final task goes beyond PID control by introducing multivariable synthesis.

- “A Robust, Multichannel Controller” on page 9-63 designs a more complex and realistic multivariable controller and compares its performance with the new PID controller.

The final section, “For More About Designing Controllers” on page 9-67, summarizes additional control design tasks and goals.

Case Study Files

This case study uses these files, in addition to the initialization M-files.

File	Purpose
`mech_stewart_control`	Main model
`mech_stewart_control_deriv`	Configurable subsystem: Derivative block or transfer function (filtered)
`mech_stewart_controllers`	Configurable subsystem: Null, PID, or H-infinity controller
`mech_stewartplatform_leg`	Library model of Stewart platform leg; used six times in the Plant subsystem of the main model

Nature of the Control Problem

The motion of an uncontrolled physical system is represented by its position and velocity variables arranged into a *state vector* $\boldsymbol{X}$. The dynamics of the system is described by a force law:

$$\mathrm{d}\boldsymbol{X}/\mathrm{dt} = \boldsymbol{f}(\boldsymbol{X})$$

Introducing control means introducing sensors and actuators that modify the system's otherwise natural motion. The actuators impose artificial forces — collectively, the *inputs* $\boldsymbol{U}$ — on the system, while the sensors detect motions and report *outputs* $\boldsymbol{Y}$. The dynamics of the controlled system are modified:

$$\mathrm{d}\boldsymbol{X}/\mathrm{dt} = \boldsymbol{f}(\boldsymbol{X},\boldsymbol{U})$$
$$\boldsymbol{Y} = g(\boldsymbol{X},\boldsymbol{U})$$

The $\boldsymbol{U}$ and $\boldsymbol{Y}$ are the *control variables* of the system.

By selecting the proper set of $\boldsymbol{U}$ and $\boldsymbol{Y}$ and a *feedback control* or *compensator law* $\boldsymbol{U} = \boldsymbol{c}(\boldsymbol{Y})$ that modifies the system's motion $\boldsymbol{X}$ in a desired way, you impose control actuator forces for the relevant range of $\boldsymbol{X}$, $\boldsymbol{U}$, and $\boldsymbol{Y}$.

$$\begin{tikzcd}\boldsymbol{U} \arrow[r] \arrow[rr, bend right, "\boldsymbol{g}"'] & \boldsymbol{X} \arrow[r] & \boldsymbol{Y} \arrow[ll, bend right, "\boldsymbol{c}"']\end{tikzcd}$$

Selecting $\boldsymbol{c}$ is the fundamental problem of control design. The desired trajectory of $\boldsymbol{X}$ is the reference or nominal trajectory. The difference of the actual and reference trajectories is the motion error. Finding the actuator forces needed to produce a desired motion is closely related to the problem of inverse dynamics. See the case study, "Trimming Through Inverse Dynamics" on page 9-24.

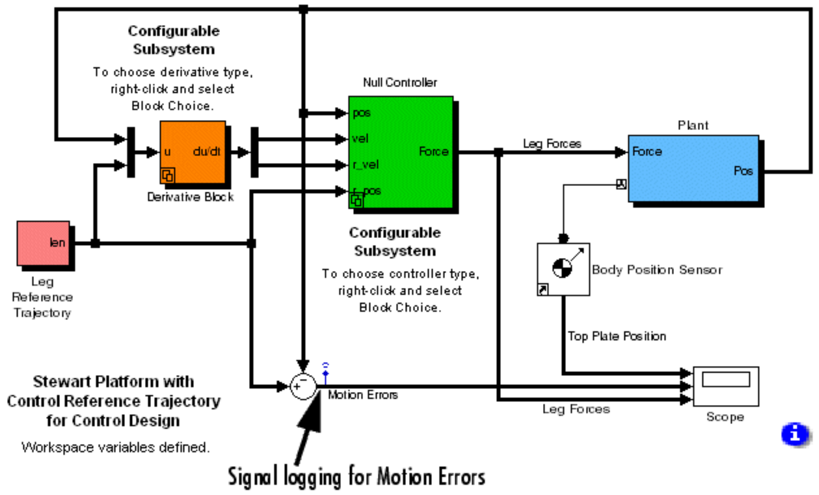
Control Transfer Function Forms and Units

The controller and plant transfer functions are often called C and G, respectively. The combined controller-plant transfer function forms are the *open-loop* CG and the *closed-loop* $CG/(1+CG)$.

Controller and plant response magnitudes are measured in decibels (dB).

A First Look at the Stewart Platform Control Model

Open the mech_stewart_control model.


Configurable Subsystem
To choose derivative type, right-click and select Block Choice.
u
du/dt
Derivative Block
Null Controller
pos
vel
r_vel
r_pos
Force
Leg Forces
Plant
Force
Pos
len
Leg Reference Trajectory
Configurable Subsystem
To choose controller type, right-click and select Block Choice.
Body Position Sensor
Top Plate Position
Stewart Platform with Control Reference Trajectory for Control Design
Workspace variables defined.
Motion Errors
Leg Forces
Scope
Signal logging for Motion Errors

Stewart Platform Control Design Model

The green controller subsystem is linked to an enabled subsystem in a related library model, mech_stewart_controllers. The initial configuration is to the Null Controller, which imposes no forces at all on the platform, the blue subsystem labeled Plant. Open the Null Controller subsystem. This controller accepts trajectory information, but outputs zero for the imposed force.

Signal logging captures the motion errors. You use this feature later to analyze controller performance.

Viewing the Controller

To see the controller subsystem library:

1 Right-click the Null Controller block and select **Link Options**, then **Go To Library Block**. The mech_stewart_controllers library opens with the Template block highlighted.

You can set this enabled subsystem in three ways, and you use all three in this case study: **Null Controller**, **PID Controller**, and **H_inf Controller**.

2 Open the controller subsystem in each setting to examine its block diagram.

3 Double-click the Template block to see the controller subsystem design. The three possible subsystem settings are listed in the Template dialog.

Close the library model.

4 You select the subsystem configuration actually used for simulation back in the original model, mech_stewart_control.

Right-click the Null Controller block and select **Block Choice**. The three possible subsystems from the mech_stewart_controllers library are listed, with **Null Controller** selected.

Configuring the Dynamics

To see the dynamical settings for the controls study model:

1 Open the Plant subsystem and the orange Machine Environment block. In the block dialog, locate the **Parameters** pane. The gravity vector points in the negative *z* direction.

Then locate the **Constraints** pane. The **Constraint solver type** is `Machine precision`, and the **Use robust singularity handling** check box is selected. For this model, such a combination is the most robust.

Close the dialog and subsystem.

2 From the **Simulation** menu, open **Configuration Parameters**. Locate the **SimMechanics** node, **Diagnostics** panel. In this simulation, automatically cut joints are marked.

Because the Stewart platform is a closed-loop system, SimMechanics cuts one joint in each closed loop formed by the two plates and a pair of legs during the simulation and marked with a red `X`. See "Counting the Degrees of Freedom" on page 9-8 and "Identifying the Simulink and Mechanical States" on page 9-21.

Close the dialog.

Simulating the Stewart Platform Without Controls

First simulate the Stewart platform without any control forces. The platform moves under the influence of gravity and initial conditions only. The reference trajectory is irrelevant because it is not used to generate any control forces.

To watch the natural or uncontrolled motion of the Stewart platform:

1. Open the Scope block. The Scope window displays three measurements:
   - Position of the top plate CG
   - Control errors
   - Control forces applied to move the legs
2. Start the model. Track the falling platform by watching the Top Plate Position graph in the Scope window. Because the controller does nothing in this version of the model, the control errors and forces are not important.

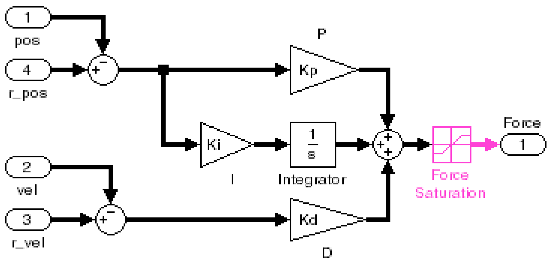
Improper and Biproper PID Controllers

Now change the model to control the Stewart platform's motion with the linear proportional-integral-derivative (PID) feedback system.

The initial controller settings are discussed in "Modeling Controllers" on page 9-15 and "Initializing the Stewart Platform in SimMechanics" on page 9-18. Here you implement two versions of this controller, improper and biproper. See "Analyzing the PID Controller Response" on page 9-44 for more.

Switching to the PID Controller Subsystem

Switch the model's controller subsystem by right-clicking on the (green) control subsystem block, selecting **Block Choice**, then **PID Controller**. The block name changes from Null Controller to PID Controller. Open it.



Stewart Platform PID Controller Subsystem

This is the PID linear feedback control system, a copy of the original subsystem contained in the mech_stewart_controllers model library. The control transfer function has the form $K_i/s + K_d s + K_p$. The control gains K_i, K_p, and K_d in their respective blocks reference the variables `Ki`, `Kp`, `Kd` defined in your workspace. Check their initialized values:

```
Ki, Kp, Kd
     Ki = 10000
     Kp = 2000000
     Kd = 45000
```

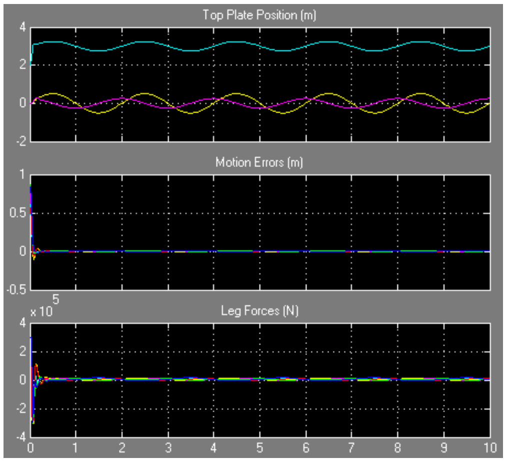
Simulating the Controlled Motion

Simulate the Stewart platform with the PID controller.

1. Open the Scope and start the simulation.

2. Observe the controlled Stewart platform motion. The Scope shows how the platform initially does not follow the reference trajectory, which starts in a different position from the platform's home configuration. The motion errors and forces on the legs are significant. Observe also that the leg forces saturate during the initial transient.

The platform moves quickly to synchronize with the reference trajectory, and the leg forces and motion errors become much smaller.



Stewart Platform Motion and Forces with the PID Controller

Finding the Numerical Derivative of the True and Reference Trajectories

The PID control law requires the time derivative of both actual and reference motion. For greater realism, the Stewart platform plant uses a Body Sensor block to detect only the actual position of the platform, leaving the velocity to be computed by the controller. Finding the reference and actual velocities requires taking numerical derivatives of the reference and actual trajectories, which each consists of the six leg lengths as functions of time.

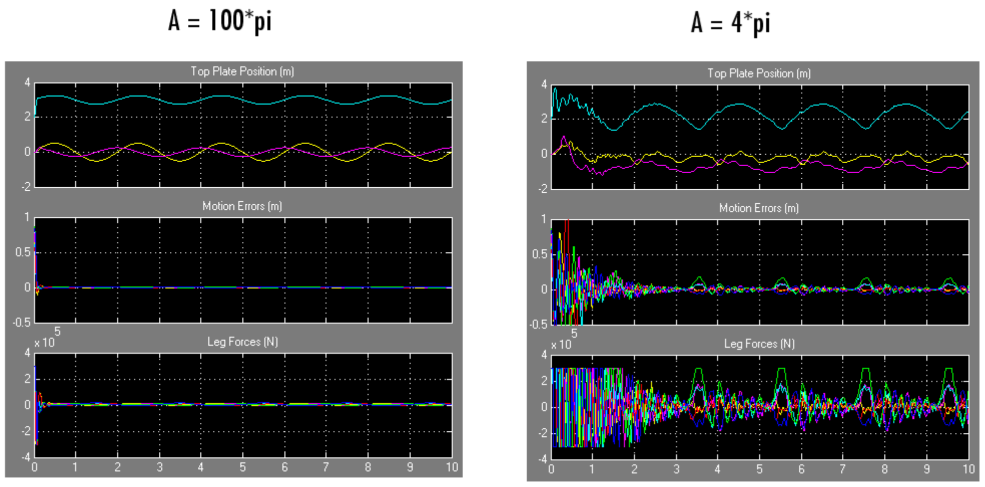
The model gives you two ways to do this. You can switch the numerical derivative configurable subsystem to implement either. This block is linked to the library mech_stewart_control_deriv, which contains the two subsystem implementations. Right-click the numerical derivative (orange) block and select **Block Choice**, then **Derivative Block** or **Filtered Derivative**.

- The first choice (improper) uses the Derivative block of Simulink. This block gives accurate but idealized results. This choice is the default.

- The second choice (biproper) applies a filter of Transfer Fcn blocks in the Laplace domain before transforming the signals back to the time domain. This choice is closer to a realistic implementation.

The transfer function has canonical form $As/(s+A)$. The transfer function acts as a low-frequency bandpass filter to damp out details of the derivative on time scales shorter than $2\pi/A$. The Transfer Fcn blocks use the workspace variable `A` representing A. Its value should be set to about 50 to 100 times the motion frequency variable `freq`. Keep the Transfer Fcn numerators and denominators in their canonical form in terms of `A`. The initialized value is $A = 100*\pi$.

The transfer function filtered derivative is more realistic, at the cost of some inaccuracy due to transients. Vary the filtered derivative behavior by adjusting `A` in your workspace. The unwanted transient behavior is worse for smaller A.



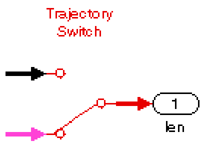
Stewart Platform Motion and Forces with PID Controller (Filtered Derivative)

Simulating at Symmetric Equilibrium

The Stewart platform's home configuration is the symmetric equilibrium of the top plate. Later in this study, you need to simulate the platform at rest. If you start the model in this state, the control forces are zero and the top plate does not move.

Keep the Filtered Derivative option and simulate this static trajectory.

1 Open the Leg Reference Trajectory subsystem. Locate the Trajectory Switch to the right. Double-click the Switch to the down position.



The reference trajectory now specifies a static reference trajectory: a platform remaining still with all legs at the same constant length.

2 Close the subsystem and start the simulation. Observe the static platform in either the Scope, the SimMechanics visualization window, or both.

3 After rerunning the model, reset the Trajectory Switch back to up.

Analyzing the PID Controller Response

Note This section requires Control System Toolbox.

You can learn more about the effect of the PID controller on the Stewart platform's motion with two control theory techniques, the *s*-plane and the frequency response, both based on the Laplace transform. See "References" on page 9-5 and the documentation for Control System Toolbox for more information.

Improper PID Controller: Theory

The PID control law is an output-input relation whose transfer function is

$$C(s) = K_p + K_i/s + K_d s = (K_p s + K_i + K_d s^2)/s = K(s - s_+)(s - s_-)/s$$

where the gains *K* are real and nonnegative. The third version is the zero-pole-gain form.

C(s) is *improper*, rising without limit for large s and having more zeros (two) than poles (one, at $s = 0$). The poles determine controller response for longer times. The zeros modify how fast the controller approaches the steady state, especially if a zero approaches and nearly cancels a pole. Obtain the steady-state by multiplying the transfer function by s, then letting s vanish.

In the PID control law, the K_i gain is the steady-state response. The transient behavior is most strongly influenced by the highest power of s (the K_d term), then by the next power of s (the K_p term), and so on. As you vary the gains, different behaviors emerge.

- If K_i vanishes, the response is all transient, with a null steady state. One zero coincides with and cancels the pole. The other zero is $-K_p/K_d$.
- If K_d vanishes, only one zero remains, at $s = -K_i/K_p$.
- If $4K_dK_i > K_p^2$, the zeros become complex and move off the real s-axis.
- If the gain is more in higher powers of s, the transient response is stronger.
- If the gain is more in the lower powers of s, the transient response is suppressed and the steady-state response emerges more quickly.

Filtered Derivative and Proper PID Controller: Theory

The simple PID control law, with an ordinary derivative, gives rise to an improper transfer function *C(s)*. Changing the ordinary derivative to a filtered derivative softens the behavior of the modified controller *c(s)* at large s.

$$\begin{aligned} c(s) &= K_p + K_i/s + K_d As/(s+A) = [K_p s(s+A) + K_i(s+A) + K_d As^2]\big/s(s+A) \\ &= [(K_p + K_d A)s^2 + (K_i + K_p A)s + K_i A]\big/(s^2 + As) \end{aligned}$$

This function is *biproper*, having two zeros and two poles, respectively, at

$$s_0 = 0,\ -A$$

$$s_\pm = -\left(\frac{K_p + K_i/A}{2\left(K_d + K_p/A\right)}\right)\left[1 \pm \sqrt{1 - 4\left(\frac{K_d + K_p/A}{K_p + K_i/A}\right)\left(\frac{K_i}{K_p + K_i/A}\right)}\right]$$

Recover the improper control law *C(s)* by letting $A \to \infty$.

PID Controller: Alternative Forms

Because *C(s)* is improper, Control System Toolbox cannot fully analyze the simple PID controller response. However, the filtered derivative alternative *c(s)* yields results similar to the ordinary derivative. A complete analysis of *c(s)* is possible.

With the `tf` command, define linear, time-invariant (LTI) transfer function objects for *C(s)* and *c(s)*, then analyze them with the LTI Viewer.

```
numC = [Kd Kp Ki]; % Improper numerator
denomC = [1 0]; % Improper denominator
cImproper = tf(numC,denomC) % Improper transfer function

numc = [Kd*A+Kp Kp*A+Ki Ki*A]; % Biproper numerator
denomc = [1 A 0]; % Biproper denominator
cBiproper = tf(numc,denomc) % Biproper transfer function
```

You can also convert *C(s)* and *c(s)* to state-space and zero-pole-gain (ZPK) forms. The latter is especially useful. Enter `help zpk` for more details.

```
zpk(cImproper) % Convert cImproper to zero-pole-gain
zpk(cBiproper) % Convert cBiproper to zero-pole-gain
```

The helpful `zpkdata` function extracts the zeros, poles, and gain from a ZPK-form controller.

PID Controller: LTI Analysis

Now open the LTI Viewer interface by entering `ltiview`.

1 Select the **File** menu, then **Import**. The **Import System Data** dialog opens.

   In the **Import from area**, select the **Workspace** option and, under **Systems in Workspace**, both entries, `cImproper` and `cBiproper`. Click **OK**.

2 Right-click within the LTI Viewer plot window to view the analysis options under **Plot Types** and **Characteristics**.

With *c(s)*, you can use all the LTI Viewer features. Your valid options for analyzing *C(s)* are limited.

- The Bode and Bode Magnitude plots show the frequency response *C(s)* for imaginary $s = j\omega$.
- The Pole/Zero plot shows the location of the poles and zeros of *C(s)*.

3 Display both the *C* and *c* systems simultaneously and compare the Bode and Pole/Zero plots.

The Bode plots are similar for small *s* (long times). For large *s* (short times), *C(s)* rises without limit, while *c(s)* levels off and results in better controller behavior.

The Pole/Zero plots show that *C(s)* has one pole and *c(s)* two poles, the common one being 0. Both transfer functions have two zeros. You can locate all of these with the `pole` and `zero` functions. Note that one zero is almost identical between *C(s)* and *c(s)*, while the other is shifted dramatically. This shift changes and softens the transient behavior of *c(s)* compared to *C(s)* for larger *s* (short times).

4 Examine the Step and Impulse plots for *c(s)* as well. These plots indicate the time behavior of the *c(s)* controller for stepped and impulsive inputs.

Designing a New PID Controller

Note This section requires Control System Toolbox. Saving intermediate model versions and workspace values is recommended.

The PID controller gains set by the initialization M-file are preoptimized. The preceding sections use these gain values as examples.

In the rest of this study, you follow a more realistic scenario where the gains are not initially known and you use control design tools in the MATLAB environment to create and optimize a filtered PID controller.

Making a First Guess for the Controller Gain

Make an initial guess for the integrator (I) gain K_i with dimensional analysis. K_i has dimensions force/length/time.

- An initial guess for the force is one-sixth the weight of the platform and legs.
- An initial guess for the length is range of vertical motion in the reference trajectory.
- An initial guess for 1/time is the natural frequency, π/2π = 0.5 Hz.

Thus an initial guess for the integrator gain is

```
Ki = 0.5*9.8*(top_mass/6+(upper_leg_mass+lower_leg_mass))/0.3

Ki = 7.1680e+003
```

Making a First Guess for the Controller Force

The initialization M-script sets the workspace variable `initCondI` to the value needed to put the platform in a symmetric equilibrium in the initial state. With a new K_i value, you need to recalibrate this initial condition.

```
initCondI = ...
   (upper_leg_mass+lower_leg_mass+(top_mass*1.3/6))*9.81/Ki

initCondI = 0.6839
```

Modifying the Null Controller with a Constant Force

Start by turning off the PID controller and applying a constant force to the platform.

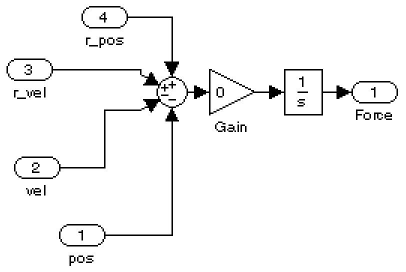
1 Right-click the controller subsystem. Select **Block Choice > Null Controller**.

2 Right-click **Null Controller** again. Select **Link Options > Go To Library Block**.

   The configurable subsystem library mech_stewart_controller opens.

3 Under **Edit**, select **Unlock Library**. Open the Null Controller template subsystem.

4 In the subsystem, between the Gain and Force (Output) blocks, insert an Integrator block.



5 Open the Integrator dialog. For **Initial condition**, enter `Ki*initCondI`. Click **OK**.

6 Close Null Controller. Save and close the mech_stewart_controller library.

7 Back in mech_stewart_control, update the diagram (**Ctrl+D**).

8 At the command line, enter `Ki*initCondI`.

This is your first guess for the controller force in one leg: the product of your PID integrator (I) gain guess and your controller initial state guess.

Simulating the Platform with the Constant Force

Now observe the effect of this constant force on the platform.

1 In the Leg Reference Trajectory subsystem, set the Trajectory Switch position to down.

2 Open the Scope and start the simulation. The control force is less than the platform weight. The platform accelerates downward.

Trimming and Linearizing the Platform Motion

Note This section requires Control System Toolbox and Simulink Control Design. Saving intermediate model versions and workspace values is recommended.

A critical step in control design is to understand the response of a plant being controlled to small disturbances in its motion [4]. This step requires

- Trimming the platform, or finding an *operating point*. This is a time trajectory satisfying certain prior conditions that you specify.

  Here you search for the simple, useful operating point of symmetric equilibrium, where the platform does not move.

- Linearizing the platform motion about the operating point.

  You save the results of the linearization to use in the next section, "Improving the New PID Controller" on page 9-56.

For More About Trimming

As described in "Trimming Through Inverse Dynamics" on page 9-24, you can trim SimMechanics models in many ways. Control System Toolbox and Simulink Control Design provide linear analysis tools more complete and powerful than what Simulink and SimMechanics alone offer.

Setting Up the Model for Trimming

Now set up the model for trimming. In Trimming mode, the model's mechanical states are the 40 constraints that reduce the 52 free (forward dynamics) states to the 12 independent states.

1 Make sure the model observes these settings.

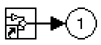
   a Keep the controller subsystem **Block Choice** set to **Null Controller** and the derivative type to **Filtered Derivative**.

   b Keep the Trajectory Switch down (static trajectory) in the Leg Reference Trajectory subsystem.

2 Reset the SimMechanics analysis mode to trimming.

- **a** Open the Plant subsystem. Double-click the orange Machine Environment block. Locate the **Parameters** tab.
- **b** For **Analysis mode**, change the pull-down menu to `Trimming`. Click **OK** and close the subsystem.

3 Observe the trimming output blocks that have appeared in the upper left of the main model.



Locating an Operating Point by Trimming

Next, locate an operating point for the Stewart platform plant.

1 Select linearization points in your model as follows. Right-click, in turn, on each of the Simulink signal lines defining the input and output of the Plant subsystem:

- Leg Forces (input)
- Pos (output)

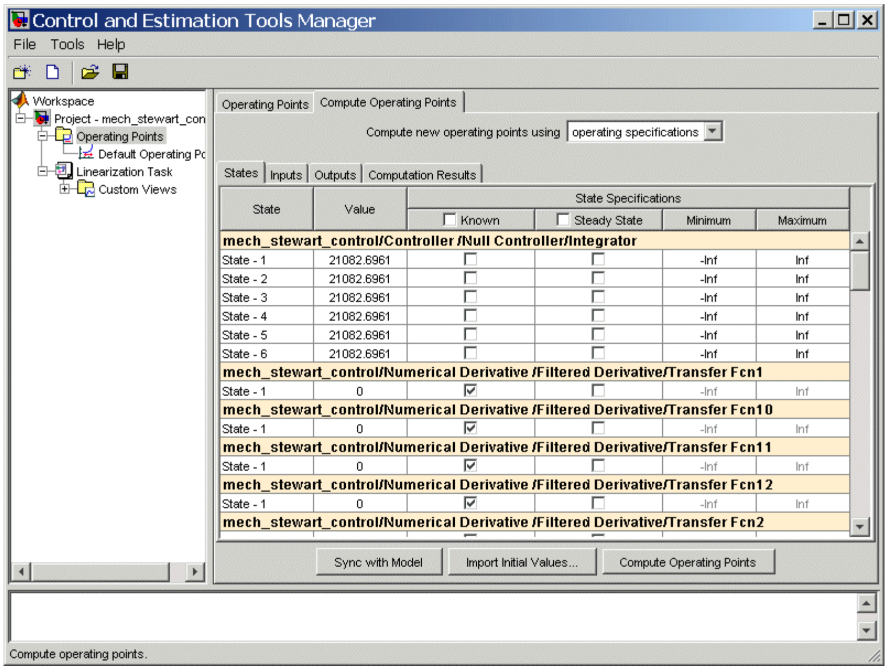
On each signal line's right-click menu, under **Linearization Points**, select

- Both **Input Point** and **Open Loop** for the input line
- Both **Output Point** and **Open Loop** for the output line

Choosing the open-loop property for these signals breaks the feedback loop from controller to plant back to controller. The plant instead takes a given set of externally imposed controller forces.

2 Then, from the model menu bar, select **Tools > Control Design > Linear Analysis**. The **Control and Estimation Tools Manager** window opens.

3 To the left of the Manager window, select the **Operating Points** node. Then, to the right, select the **Compute Operating Points** panel. Click the **Sync with Model** button at the bottom of the panel.



Compute new operating points using: operating specifications

State	Value	State Specifications			
		☐ Known	☐ Steady State	Minimum	Maximum
mech_stewart_control/Controller /Null Controller/Integrator					
State - 1	21082.6961	☐	☐	-Inf	Inf
State - 2	21082.6961	☐	☐	-Inf	Inf
State - 3	21082.6961	☐	☐	-Inf	Inf
State - 4	21082.6961	☐	☐	-Inf	Inf
State - 5	21082.6961	☐	☐	-Inf	Inf
State - 6	21082.6961	☐	☐	-Inf	Inf
mech_stewart_control/Numerical Derivative /Filtered Derivative/Transfer Fcn1					
State - 1	0	☑	☐	-Inf	Inf
mech_stewart_control/Numerical Derivative /Filtered Derivative/Transfer Fcn10					
State - 1	0	☑	☐	-Inf	Inf
mech_stewart_control/Numerical Derivative /Filtered Derivative/Transfer Fcn11					
State - 1	0	☑	☐	-Inf	Inf
mech_stewart_control/Numerical Derivative /Filtered Derivative/Transfer Fcn12					
State - 1	0	☑	☐	-Inf	Inf
mech_stewart_control/Numerical Derivative /Filtered Derivative/Transfer Fcn2					

Compute operating points.

The default subpanel is **States**. The **Steady State** check boxes are selected by default. This choice searches for a plant operating point where the platform is at rest relative to its initial configuration.

4 Examine the states by scrolling down in the **States** window.

- There are six states associated with the null controller Integrator block.

  Clear the **Steady State** check boxes for these states. The trimming will not hold the controller signal as fixed.

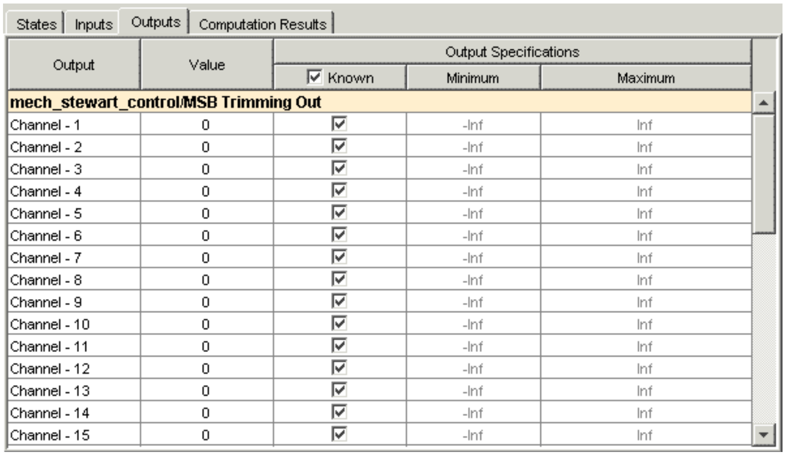
- Below these six are twelve states associated with the Transfer Fcn blocks in the Filtered Derivative subsystem.

  Free them from being fixed by clearing their **Steady State** check boxes. Make their values (0) known by selecting their **Known** check boxes.

The rest of the states are associated with the positions and velocities of the Stewart platform leg joints. Only six of these states are independent. The others are constrained. Leave their settings as the defaults.

5 Move to the **Outputs** subpanel. Under **Output Specifications**, select the **Known** check box (the topmost check box in that column). This action specifies all outputs, the state deviations from the desired operating point. There are 40 states (constraints) in Trimming mode.

The output values are specified in the **Value** column. The values are all zero, indicating that all constraints on states (the specifications of the operating point) must be satisfied within tolerance.



Output	Value	Output Specifications		
		✓ Known	Minimum	Maximum
mech_stewart_control/MSB Trimming Out				
Channel - 1	0	✓	-Inf	Inf
Channel - 2	0	✓	-Inf	Inf
Channel - 3	0	✓	-Inf	Inf
Channel - 4	0	✓	-Inf	Inf
Channel - 5	0	✓	-Inf	Inf
Channel - 6	0	✓	-Inf	Inf
Channel - 7	0	✓	-Inf	Inf
Channel - 8	0	✓	-Inf	Inf
Channel - 9	0	✓	-Inf	Inf
Channel - 10	0	✓	-Inf	Inf
Channel - 11	0	✓	-Inf	Inf
Channel - 12	0	✓	-Inf	Inf
Channel - 13	0	✓	-Inf	Inf
Channel - 14	0	✓	-Inf	Inf
Channel - 15	0	✓	-Inf	Inf

6 From the Manager window menu bar, select **Tools > Options**. The **Options** window opens. Select the **Operating Point Search** panel.

In the **Optimization Method** area, select `Nonlinear least squares` in the **Optimization Method** menu.

Leave the other defaults. Click **OK**. The **Options** window closes.

7 Back in the **Control and Estimation Tools Manager**, click the **Compute Operating Points** button at the bottom of the **Create Operating Points** panel.

The **Computation Results** subpanel indicates the progress of the trimming. When finished, it should indicate that the operating point specifications were successfully met.

In the **Operating Points** node to the left, a new **Operating Point** subnode appears, `Operating Point`, containing the results of this trimming.

Interpreting and Saving the Operating Point

Examine and save the operating point results.

1 Click **Operating Point**. Look at the **States** and **Outputs** panels.

   Under **Outputs**, the **Desired dx** values (if not marked `N/A`) are zero. For the mechanical states (constraints), the **Actual dx** values (deviations from the requested operating point) are zero within tolerance.

   This is not true for the Controller states, which you did not require to vanish. The Filtered Derivative states are all zero.

2 Save this operating point by right-clicking `Operating Point` and selecting **Export**. Except for the name, leave the defaults.

   For **Variable Name**, enter `oppoint_PLANT`. Click **OK**.

   You now have a workspace object (`opcond.OperatingPoint` class) called `oppoint_PLANT` representing the plant holding still at the start of simulation (t=0). Retain this object for later use.

3 Examine its states by entering

```
oppoint_PLANT % List plant states at t=0
```

4 Reset the controller initial condition to the new operating point.

```
initCondI = oppoint_PLANT.States(1).x(1);
```

Linearizing the Platform Motion at the Operating Point

Now switch the model back to Forward Dynamics mode. The mechanical states are now the 52 tree states corresponding to the uncut joint primitives.

1. Open the Plant subsystem, then its orange Machine Environment block. Locate the **Parameters** tab.

2. In the **Analysis mode** pull-down menu, select `Forward Dynamics`. Click **OK** and close the subsystem.

Then linearize the plant motion about the operating point you specified in earlier. Return to the Control and Estimation Tools Manager.

1. Select **Tools > Options**. In the **Options** dialog, select the **Linearization State Ordering** tab.

   Click the **Sync with Model** button at the bottom, then click **OK**.

2. Now select the **Linearization Task** node to left, then the **Operating Points** panel. Select the **Operating Point** called `Operating Point`.

3. At the bottom of the panel, make sure the **Plot linear analysis result in a** check box is selected. Then choose a plot type in the pull-down menu. For example, pick `Bode response plot`.

4. Then click the **Linearize Model** button. The LTI Viewer opens with a large family of Bode response plots.

   For later reference, you can choose other response plot types by right-clicking on one of the plots and, under **Plot Type**, selecting a different plot, such as **Bode**, **Step**, or **Impulse**. (You do not need to go back to **Linearization** and relinearize the model.)

Interpreting and Saving the Linearization Results

This plant linearization started with six inputs (the leg forces) and 12 outputs (six leg positions and six leg velocities). The LTI Viewer displays 6 x 12 = 72 response plots. To view one plot individually,

1. Right-click any one of the 72 plots and select **I/O Selector**. The **I/O Selector** dialog opens.

2. This dialog lets you to choose any response of one output relative to one input. To see that plot in the LTI Viewer, click the corresponding black dot.

Each plot shows how one of the outputs (a position or velocity) responds to the application of a small force in one of the input channels. Different plot types (impulse, step, Bode, etc.) yield different aspects of the response.

Export the results of your linearization.

1 Select **File > Export** in the LTI Viewer.

2 Choose your model and give it a unique name (call it `sys`) under **Export As**.

3 Click **Export to Workspace**. The model is saved as an LTI object. The variable class is `ss`, the canonical state space form used by Simulink.

Retain this LTI object for the next section, where you use it to improve the PID controller.

Further Suggestions

You can apply these results to other controllers (see "A Robust, Multichannel Controller" on page 9-63), as well as choose other operating points.

Improving the New PID Controller

Note This section requires Control System Toolbox and Simulink Control Design. Saving intermediate model versions and workspace values is recommended.

To proceed with this section, you need to have completed the preceding section, "Trimming and Linearizing the Platform Motion" on page 9-50.

In this section, you use the linearization results to create a controller to better match the plant. This information allows you to convert open-loop information about the controller and plant into closed-loop behavior of the coupled system.

A PID controller acts as the same controller on each of the platform legs. You can improve the controller's response to each leg's motion by working with the diagonal components of the plant response. These components represent a leg's motion response to the force acting on that leg. This control design paradigm is single-in, single-out (SISO). By symmetry, designing the PID settings with one of the leg's control behavior optimizes them for the other five.

The SISO approach ignores coupling between the legs. The last section of the study, "A Robust, Multichannel Controller" on page 9-63, tackles multichannel coupling to achieve a more accurate controller design.

What You Need from Previous Sections

From the preceding section, "Trimming and Linearizing the Platform Motion" on page 9-50, you should have these saved in your workspace:

- Linearized plant model as an LTI object (`ss` class) called `sys`
- Controller initial condition `initCondI` reset to the operating point
- Useful intermediate model versions and workspace variable MAT-files

Throughout this section, keep the derivative block as **Filtered Derivative** and the PID controller as biproper.

Reducing the State Space with Minimal Realization

Many of the mechanical states in `sys` are constrained. Remove them with the `sminreal` command. This reduction works with the structure of the `sys`, rather than (like `minreal`) with the numerical properties of `sys`.

```
G = sminreal(sys); % Structural reduction of linearized sys
```

`G` now represents the reduced linearized plant.

Exploring PID Gains, Filtered Derivative, and Force Saturation

One way to get a feel for the effect of PID feedback control on the Stewart platform's motion is to vary the gains, frequency cutoff, and force saturation systematically, while holding fixed the reference trajectory and the platform initial conditions.

The larger K_d is relative to K_p and K_i, the more sensitive the controller is to immediate changes in the reference signal. (The same is true of K_p relative to K_i.) The derivative term emphasizes rapid change. On the other hand, if K_d is small, the controller is more sluggish in response. The K_i term emphasizes memory of motion errors past. A fundamental tradeoff of control design is

- A more responsive PID controller is also less stable against high-frequency (short time-scale) disturbances such as noise.
- A more stable controller is less responsive to feedback.

For large filtering constant A, the biproper transfer function *c(s)* behaves at small s almost exactly like the improper *C(s)*. But as you reduce A, *c(s)* behaves less like *C(s)*. In the time domain, for smaller A, the controller *c(s)* shows more transient deviation from the pure derivative behavior of *C(s)*.

The PID controller also depends on the force saturation limit, set in the workspace by `force_act_max`. Making the force saturation limit too small means that the controller cannot actuate the legs sufficiently to make them keep up with the reference trajectory signal. The platform motion moves toward instability with a lower force saturation limit. Too low a limit eventually yields motion that is unacceptably extreme or completely unstable. Up to a point, you can compensate for a lower force saturation limit by making the controller more responsive.

Analyzing the Plant Response with the SISO Design Tool

A better way to optimize the PID controller is to analyze the open- and closed-loop system response with the SISO design tool.

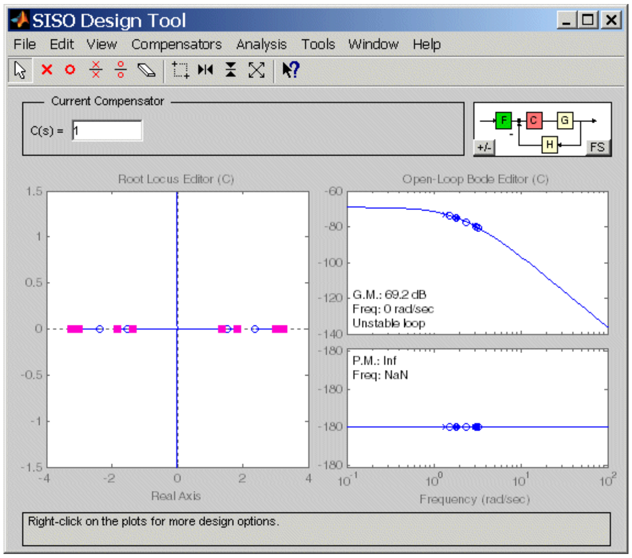
Open the **SISO Design Tool** by entering

```
sisotool(G(1,1)); % SISO design tool for first leg-leg pair
```

The design tool opens with a unity controller (compensator), `C(s) = 1`. Use the **Help** menu for complete information about the design tool, including how to interpret the plot symbols.

The **Root Locus Editor** to the left shows the closed-loop *CG/(1+CG)* response, the *s*-plane poles, zeros, and root-loci. The **Open-Loop Bode Editor** to the right shows the open-loop *CG* plant response, including poles and zeros.

The closed-loop response has eight poles, four on the left-half and four on the right-half of the *s*-plane, the latter indicating instability. The open-loop Bode plot displays the gain and phase margins.



Right-click on the plots for more design options.

SISO Design Tool with Stewart Platform Plant at Rest and Unity Controller

Designing a New Biproper PID Controller with the Plant Response

To design a biproper PID controller, add two zeros and two poles and adjust the overall gain. Observe these general rules for the poles and zeros:

- The numerator coefficients, including the overall gain, must be positive. The easiest way to ensure this is for both zeros to have negative real parts.
- One pole must occur at zero. This corresponds to the integrator (I) part.
- The other pole must have negative real part.

To implement,

1 Select **Compensators > Edit > C**. The **Edit Compensator C** dialog opens. Add poles and zeros. Click **OK**. The dialog closes.

2 In the root-locus plot, you can move controller and closed-loop poles and zeros around by dragging them with your mouse. As you move closed-loop poles, you also change the overall controller gain. Be sure to leave the initially stable closed-loop poles in the left half-plane.

   In the Bode editor, you can move open-loop (controller) poles and zeroes by dragging them. You can also change the gain and phase margins.

3 The SISO design tool controller form is $\kappa(1+\alpha s)(1+\beta s)/s(1+\gamma]s)$. The overall control gain κ is K_i in this form.

   For K_i, use the value of your first guess found previously in "Designing a New PID Controller" on page 9-47.

Optimizing the New Biproper PID Controller with the Plant Response

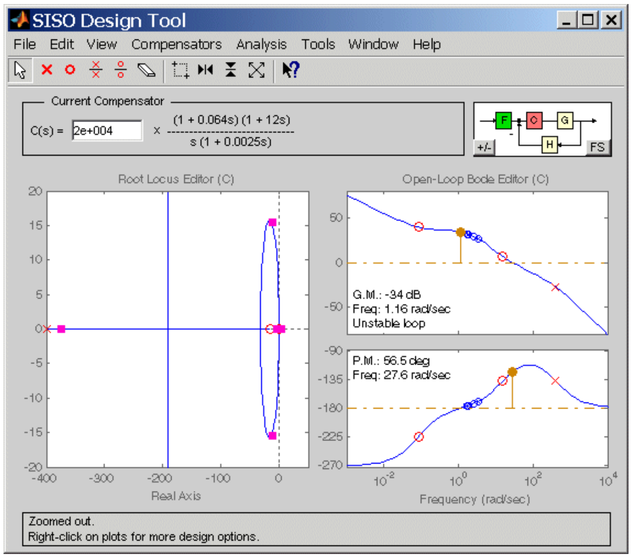
To optimize your controller, change its response to suppress undesirable and enhance desirable feedback. The objectives, typical in control problems, are a high-gain response at low frequencies to achieve tracking performance and a diminishing response at high frequencies to limit the controller's sensitivity to plant variations and noise.

The platform motions have low bandwidth, typically only a few Hertz (Hz). The system should have strong response up to a few Hz (ω = about 10 rad/s), then falling response for higher frequencies.

One controller pole must always remain at zero. Five system poles have positive (unstable) real parts, a result of the first leg coupling to the other five. You cannot eliminate these in a SISO analysis.

Improve the controller by

- Making the nonzero controller pole more negative. This increases A and increases the phase margin while decreasing the gain margin.
- Improving transient response by adjusting the controller zeros.
- Lowering the gain margin by raising the overall Bode response. This increases the overall controller gain $\kappa = K_i$.



Zoomed out.
Right-click on plots for more design options.

Saving the Optimized New Biproper Control Law

Once you have a satisfactory controller, you can export the new optimized biproper control law to the workspace and analyze it there to redefine the filtered PID controller parameters K_i, K_p, K_d, and A.

Export the modified compensator from the SISO design tool.

1. Go to **File > Export**. Select `Compensator`. Rename it `cBiproperOpt` under **Export as**.

2. Then click **Export to Workspace**.

`cBiproperOpt` is a zero-pole-gain form (LTI object of class `zpk`). For example,

```
cBiproperOpt

Zero/pole/gain:
6171074.4994 (s+15.51) (s+0.08378)
----------------------------------
          s (s+400)
```

Resetting the PID Gains and Derivative Cutoff

Extract the biproper PID controller parameters by inverting the zeros $s_\pm$, poles, and gain K. The standard zero-pole-gain form is

$$c(s) = K(s - s_+)(s - s_-) / s(s+A) = [(K_p + AK_d)*s^2 + (K_i + AK_p)*s + AK_i] / s(s + A)$$

- A = the negative of the biproper nonzero pole
- The gains are:

$$K_i = Ks_+s_- \text{ , } K_p = -[K(s_+ + s_-) + K_i]/A \text{ , } K_d = (K - K_p)/A$$

Reset your workspace variables accordingly.

```
[z,p,k] = zpkdata(cBiproperOpt) % Extract ZPK data from cBiproper
A = -p{1,1}(2) % Extract nonzero pole
Ki = k*z{1,1}(1)*z{1,1}(2)/A % Extract  Ki gain
Kp = -(k*(z{1,1}(1) + z{1,1}(2)) + Ki)/A % Extract Kp gain
Kd = (k - Kp)/A % Extract Kd gain
```

Checking the Symmetric Equilibrium

Check that the symmetric equilibrium is stable with your new controller.

1 Make sure the Trajectory Switch is set to down.

2 Update the diagram (**Ctrl+D**) and rerun the model.

   A trim point is rarely exact. There is typically a small but nonzero motion error as the platform relaxes toward equilibrium.

Simulating the Moving Platform and Capturing the Motion Errors

Now test the platform motion with the moving trajectory and your new retuned biproper control law.

1 Set the Trajectory Switch back to up.

2 Restart the model. You should see reasonable motion errors and leg forces, except perhaps for an initial transient.

3 Capture the Motion Errors from the logged signals structure `sigsOut`.

```
pid_opt_TS = sigsOut.('Motion Errors'); % Record motion errors
```

A Robust, Multichannel Controller

Note This part of the study requires Control System Toolbox and Robust Control Toolbox.

To complete this section, you need to have completed the preceding section, "Improving the New PID Controller" on page 9-56.

The controllers you have designed so far in this study are based on classical PID techniques, where each channel is subject to the same control law and the control law is tuned one channel at a time. This approach misses the *cross-coupling*, the effect that the force on one platform leg has on the motion of the other legs.

In this section, you redesign the Stewart platform controller by using modern techniques that take multichannel coupling into account and implementing a robust *H-infinity* controller [5], [6].

What You Need from Previous Sections

From preceding sections, you should have these saved in your workspace:

- Reduced state space representation `G` of the plant
- Time series structure `pid_opt_TS`

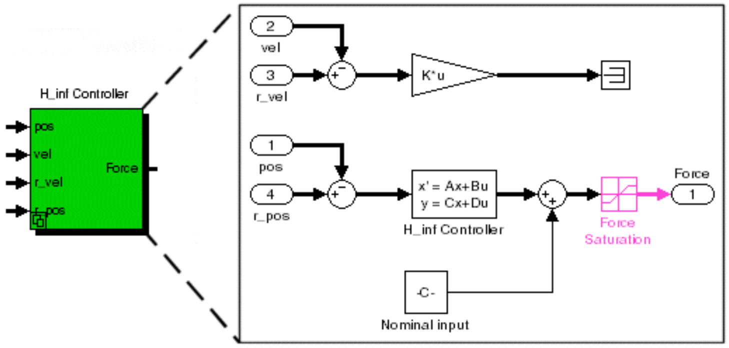
Viewing the H-Infinity Controller

Before starting,

1. From its right-click menu, under **Block choice**, switch the controller subsystem to **H_inf Controller**.

2. Make sure that the derivative subsystem remains set to **Filtered Derivative** and the Trajectory Switch in the Leg Reference Trajectory subsystem is set to up.

Examine the controller subsystem, which is implemented via state space.



Stewart Platform H-Infinity Controller Subsystem

Defining a Desired Loop Shape Response

Start by specifying a desired open-loop response $|C^*G(1,1)|$ and plot its singular values. For example,

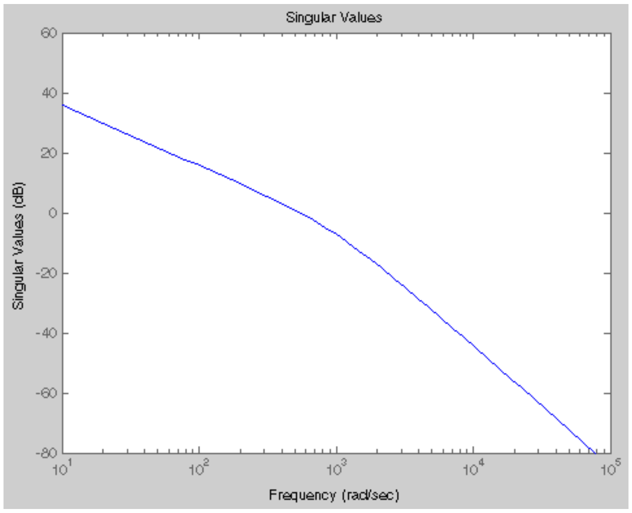
```
  Lsd = zpk([],[-1000 0],612770) % Define desired loop shape

  Zero/pole/gain:
    612770
  ----------
   s (s+1000)

   sigma(Lsd) % Plot singular values
```

View the closed-loop response generated by this loop shape by entering:

```
step(feedback(Lsd,1)) % Feedback step response
```



Desired Loop Shape: Singular Values

Synthesize and Reduce a Controller with the Desired Loop Shape

Now create a controller using the desired loop shape and plant response:

```
[K_ls,CL,GAM,INFO] = loopsyn(G,Lsd); % Synthesize controller
```

Check the size of the controller by entering

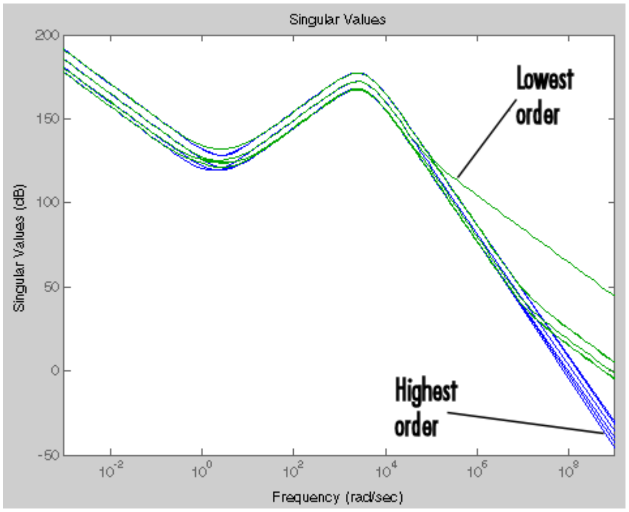
```
size(K_ls) % Check size of loopsyn controller
```

The example controller has 48 states. It is usually impractical to implement a controller of such high order and computational intensity. So try reducing the controller to 24th order:

```
Kr_ls = reduce(K_ls,24); % Reduce controller order
```

To estimate how many states you can ignore (truncate), plot both the full and reduced singular values

```
sigma(K_ls,Kr_ls) % Plot singular values
```



Full and Reduced Loop-Synthesized Controllers: Singular Values

Simulating the Robust Controller and Capturing Its Motion Errors

From the synthesized loop shape, extract the matrices needed to define the state space model used in the H_inf Controller subsystem.

```
[Ak,Bk,Ck,Dk] = ssdata(Kr_ls); % Extract state space model
```

Run the loop-synthesized controller model. Then capture the motion errors.

```
loopsyn_TS = sigsOut.('Motion Errors'); % Record motion errors
```

Plotting and Comparing the Results

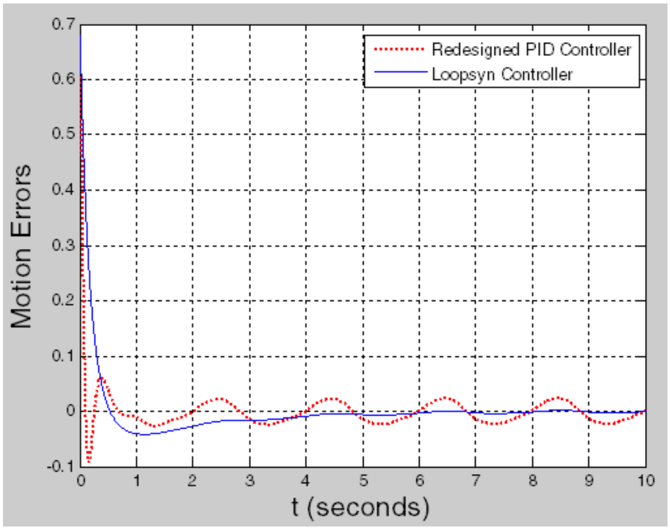
Finally, compare the motion error data from the two controllers:

- Redesigned PID
- Robust loop-synthesized

At the command line, enter:

```
figure
plot(pid_opt_TS.Time,pid_opt_TS.Data(1,:),'r', ...
   loopsyn_TS.Time,loopsyn_TS.Data(1,:),'b')
ylabel('Motion Errors','FontSize',16)
xlabel('t (seconds)','FontSize',16)
legend('Redesigned PID Controller','Loopsyn Controller')
```

Apart from the initial transient, the loop-synthesized controller performs better than the redesigned PID controller. In this example, the late-time robust controller motion errors are more than an order of magnitude smaller and exhibit no oscillatory "ringing."



Redesigned PID and Loop-Synthesized Control System Motion Errors

For More About Designing Controllers

The problems and techniques of this study only touch the basics of control design. In practice, you need to consider additional issues. Also consult "References" on page 9-5.

Finding Other Operating Points

To fully understand the plant, you need to find other plant operating points and optimize the controller in other representative states.

See the preceding case study, "Trimming Through Inverse Dynamics" on page 9-24.

Compensating for Noise and Uncertainty

To make the controller more robust, you should consider the effect of parameter uncertainty and signal noise. This step involves comparing typical plant motion frequencies, noise frequencies, and the filtered derivative cutoff.

The following toolboxes can help with such tasks:

- Robust Control Toolbox
- Simulink Response Optimization
- Simulink Parameter Estimation

Designing for Hardware Implementation

To move toward hardware implementation, you must consider discretizing the controller [7]. Among other requirements, this necessitates using a fixed-step solver, optimizing the solver step size and sample rate, and adjusting the filtered derivative cutoff.

See the following two case studies:

- "Simulating with Code" on page 9-69
- "Hardware in the Loop" on page 9-79

Simulating with Code

Note This advanced case study requires some experience with the code generation features of Simulink. To complete it, you need to have Real-Time Workshop installed, in addition to SimMechanics.

This case study leads you through a representative set of tasks related to turning a Stewart platform model into generated code. “Learning About the Model” on page 9-69 presents the model and special code generation requirements. After you read it, proceed with the code generation tasks.

Code Generation Tasks

All code generation-related files and subdirectories are created in your current MATLAB directory.

1. In “Generating an S-Function Block for the Plant” on page 9-73, you convert the plant subsystem into a new, reusable S-Function block.
2. In “Model Referencing the Plant” on page 9-75, you reference the plant as an external model from the plant subsystem in your original model and convert the referenced model to code.
3. In “Generating Stand-Alone Code for the Whole Model” on page 9-77, you convert the entire model into stand-alone code.

For More Information About Code Generation

To learn more about generating code from Simulink models, consult the documentation for Real-Time Workshop.

To learn more about code generation with SimMechanics, see “Generating Code” on page 5-28.

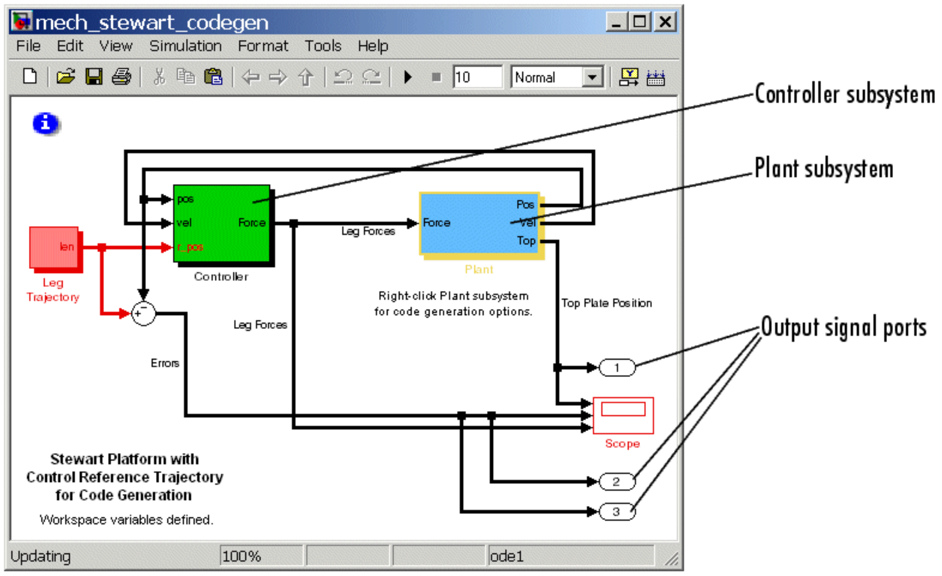
Learning About the Model

This study is based on these demo files, in addition to the initialization M-files. Copy them into an empty directory before starting each case study task.

File	Purpose
`mech_stewart_codegen`	Basic model
`mech_stewart_codegen_plant`	Plant subsystem as separate model

Open the first model. Then update the model by either entering **Ctrl+D** from the keyboard.

You use the second model file later for model reference in "Model Referencing the Plant" on page 9-75.



Solver and Sample Time Step Sizes

The model defines two time steps, `dt1` and `dt2`. The model initializes both to the same value (see Configuration Parameters for Stewart Platform Code Generation on page 9-72):

- `dt1` defines the fixed-step solver time step.
- `dt2` defines the sample rate for the generation of the trajectory signals.

Structure of the Model

The two major parts of the model are the PID controller and the plant. You can generate code from the entire model or from only part of it. In this study, you convert the plant subsystem to code in two ways, by an S-function block and by model reference. You then convert the whole model to stand-alone code.

Caution To convert a subsystem alone to code requires placing all of the SimMechanics blocks (the blocks with the distinctive Physical Modeling connector ports ● and Body Coordinate System ports ⊞) into the subsystem. mech_stewart_codegen encapsulates all SimMechanics blocks in the Plant subsystem.

Run the model before continuing with code generation. You can view the Stewart platform motion by opening the Scope. You can also enable visualization. (See Chapter 6, "Visualizing and Animating Machines".)

Simulation Settings for Code Generation

Some of the Simulink and SimMechanics settings in mech_stewart_codegen are different from the defaults.

From the model's **Format** menu, check that these entries are selected:

- **Port/Signal Displays > Sample Time Colors**
- **Port/Signal Displays > Wide Nonscalar Lines**

Other settings are optimized for code generation.

1. View the Plant subsystem parameters by right-clicking the subsystem and selecting **Subsystem Parameters**, then close the dialog.
   - **Treat as atomic unit** is selected.
   - **Minimize algebraic loop occurrences** is selected.
2. Now view the **Configuration Parameters** dialog by selecting it from the model's **Simulation** menu. View the different nodes, then close the dialog.

- **Solver** node. The model uses a fixed-step solver. While S-function Target does not require fixed-step solvers, most Real-Time Workshop targets require fixed-step solvers.
- **Data Import/Export** node. Time, states, and output are selected for data export.
- Outputs correspond to the ports connected to the outport signals.

  Top Plate Position: translation and rotation (Port 1)

  Errors: difference of reference and actual top plate positions (Port 2)

  Leg Forces: control forces parallel to each Stewart platform leg (Port 3)
- The states represent the states of the controller and the plant. The controller has Simulink states, and the plant has SimMechanics mechanical states.

  The model states are not identical to the system's independent degrees of freedom (DoFs). See "Counting the Degrees of Freedom" on page 9-8 and "Identifying the Simulink and Mechanical States" on page 9-21.

Configuration Parameters for Stewart Platform Code Generation

Node	Settings
Solver	**Solver options**: **Type**: `Fixed-step` **Solver options**: **Solver**: `ode1 (Euler)` **Fixed-step size**: `dt1` (`5e-3` seconds)
Data Import/Export	**Time**: `tout` **States**: `xout` **Output**: `yout`
Optimization	**Simulation and code generation**: **Inline parameters** selected (needed for Model Reference)
Model Referencing	**Minimize algebraic loop occurrences** selected

Configuration Parameters for Stewart Platform Code Generation (Continued)

Node	Settings
Real-Time Workshop	**Target selection**: **System target file**:`ert.tlc (no auto configuration)` (Embedded Real-Time Target) **Interface**: **Software environment**: **continuous time** selected **Interface**: **Verification**: **MAT-file logging** selected
SimMechanics	**Diagnostics**: all cleared **Visualization**: all cleared

3 Now open the Plant subsystem and the orange Machine Environment block. Check the following settings, then close the dialog.

The constraint solver is set to stabilizing, a robust choice appropriate for a fixed-step simulation of moderate computational cost. Robust singularity handling is selected.

Machine Environment Settings for Stewart Platform Code Generation

Panel	Settings
Parameters	**Linear assembly tolerance**: `1e-3 m` **Angular assembly tolerance**: `1e-2 rad`
Constraints	**Constraint solver type**: `Stabilizing` **Use robust singularity handling** selected
Visualization	**Visualize machine** selected

Generating an S-Function Block for the Plant

The S-function Target feature of Real-Time Workshop lets you generate an S-function block for a subsystem. This block points to a (non-stand-alone) auxiliary binary file that hides the original subsystem. You can then use the S-function block in multiple instances in any Simulink model, including your original one, without SimMechanics.

Caution To conform to Real-Time Workshop default settings, you must reset the simulation (stop) time to 50 times the step size, or `50*dt1`, just before generating the S-function block. Afterward, you can set the stop time back to its original value before starting the simulation with the S-function block.

1. Right-click on the Plant subsystem. Select **Real-Time Workshop**, then click **Generate S-Function** in the submenu.

   A new window opens, **Generate S-function for Subsystem: Plant**, listing the workspace variables used in the subsystem. At this point, you can make ordinary Simulink parameters tunable, but you cannot tune SimMechanics parameters. See "Generating Code" on page 5-28.

2. Proceed with generating the code files by clicking **Build** in the tunable parameter window. Follow the generation in the command window.

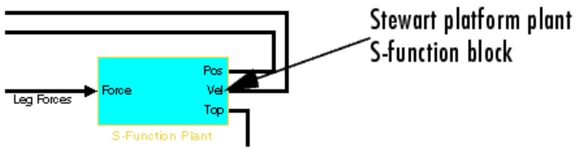
   Two auxiliary subdirectories are created, as well as C source and header files and a (non-stand-alone) linked binary. Each of these files has a name, `Plant_sf`, derived from the subsystem name.

   A new Simulink model window also appears, containing the new, reusable S-function block named Plant that points to the linked binary. Rename this block to S-Function Plant.

3. From the original mech_stewart_codegen model window, cut the Plant subsystem. Paste it into the new, untitled window.

   Save this new model, containing the S-function and subsystem blocks for future use, as mech_stewart_codegen_plant_sfunc.

4. Copy the S-Function Plant block from the untitled window into the original model window. Connect the signal lines to the S-function block.



5 Now start the model with the new S-function block. This modified model no longer requires SimMechanics. The performance is about the same as the original model with the subsystem.

Save the modified model for future use as mech_stewart_codegen_sfunc.

Model Referencing the Plant

Real-Time Workshop gives you another way to generate code for a subsystem. Using the Model Reference feature, you can put the subsystem in a separate model, then replace the subsystem block in the original model with a model reference block that points to the new model holding the subsystem. For mech_stewart_codegen, the plant subsystem is contained in the model file mech_stewart_codegen_plant.

One advantage of model referencing is that it allows you to incrementally compile parts of your model, one at a time. This feature saves significant time when you generate code from large models.

Simulation Settings for Model Reference

Some settings in mech_stewart_codegen_plant are different from the defaults. Many differ from the defaults in the same way that mech_stewart_codegen does. Here are additional settings in the **Configuration Parameters** of this model that differ from the defaults.

Node	Settings
Model Referencing	**Rebuild options for all referenced models**: **Rebuild options**: `Never` **Rebuild options for all referenced models**: **Never Rebuild targets diagnostic**: `None`
Real-Time Workshop	**Interface**: **Code interface**: **Single output/update function** cleared

Setting Up and Running the Main Model for Model Reference

To reconstruct your model for model referencing,

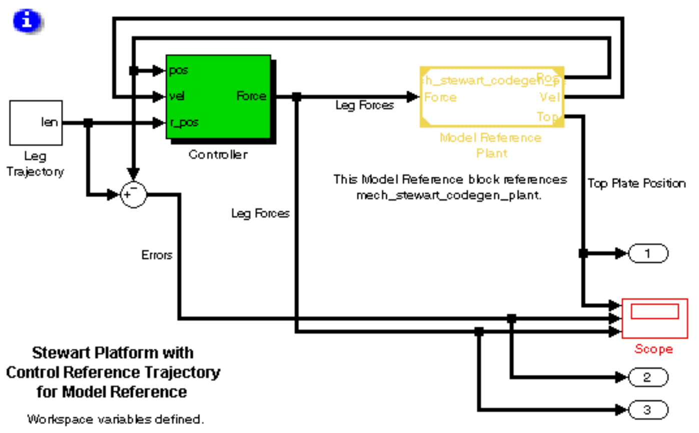
1 In mech_stewart_codegen, cut the Plant subsystem.

Keep this subsystem for future use. From the preceding part of the study, "Generating an S-Function Block for the Plant" on page 9-73, you have a model, mech_stewart_codegen_plant_sfunc, with the subsystem. If you have not already saved the subsystem here, do so now by pasting it in.

2 From the Simulink Ports & Subsystems library, drag and drop a Model block into mech_stewart_codegen. Rename the block Model Reference Plant.

3 Open the Model Reference Plant block dialog. In the **Model name** field, enter mech_stewart_codegen_plant. Click **OK**.

Save the new, modified model as mech_stewart_codegen_modelref.


pos
vel
r_pos
Force
Controller
len
Leg Trajectory
Leg Forces
h_stewart_codegen
Force
Pos
Vel
Top
Model Reference Plant
This Model Reference block references mech_stewart_codegen_plant.
Top Plate Position
Leg Forces
Errors
1
Scope
2
3
Stewart Platform with Control Reference Trajectory for Model Reference
Workspace variables defined.

Stewart Platform with Control Reference Trajectory for Model Reference

4 In the toolbar of mech_stewart_codegen_modelref, click the **Start** button.

In the command window, watch the code generation for model referencing. When it finishes, the simulation starts. Watch the simulation results by opening the Scope block.

Running or updating the main model generates a code directory and a non-stand-alone (linked) binary file called mech_stewart_codegen_plant_msf from the referenced model. The model reference block in the original model points to this binary. You can view the model reference code generation in the command window each time you update the diagram.

Generating Stand-Alone Code for the Whole Model

In this section, you generate a stand-alone executable from the original Stewart platform model, mech_stewart_codegen, using Real-Time Workshop and the Embedded Real-Time target. This executable is portable and independent of MATLAB.

1 From the **Tools** menu in the model menu bar, select **Real-Time Workshop**, then **Build Model**. The build process begins in the command window.

   Real-Time Workshop generates two auxiliary subdirectories, as well as a stand-alone executable named mech_stewart_codegen.

2 Start the executable by entering

```
!mech_stewart_codegen
```

   A MAT-file called `mech_stewart_codegen.mat` is created whenever you run the executable. This file contains the output, state, and time data exported from the model.

3 You can load this MAT-file into your workspace and examine its variables, all distinguished by the `rt_` prefix.

   From the MATLAB Desktop **Current Directory** window, right-click `mech_stewart_codegen.mat` and select **Import Data**. The **Import Wizard** appears, listing the variables that were generated, at each time step, by running the executable. These include:

   - `rt_tout`: Simulation times
   - `rt_xout`: States
   - `rt_yout`: Outputs

4 Click **Finish** on the **Import Wizard** dialog. The variables are loaded into your workspace.

   Examine the variables in workspace and double-click each of the three. The **Array Editor** displays the variable values as arrays (in the two cases of outputs and states, arrays as parts of data structures).

These variables include:

- The variable `rt_tout` containing the simulation times.
- The variable `rt_xout` containing the state signals. These states include a six-column array representing the Controller subsystem states and a 52-column array representing the mechanical states of the Plant.
  - The six controller states are the six leg positions integrated by the Controller/Integrator block for PID control.
  - The 52 mechanical states are discussed in "Identifying the Simulink and Mechanical States" on page 9-21.
- The variable `rt_yout` containing the output signals. These outputs are the three output signals designated by the model's output signal ports (Top Plate Position, Errors, and Leg Forces).

Hardware in the Loop

Note This case study requires experience with code generation and dedicated hardware deployment. To complete it, you need to have installed the following products, besides MATLAB, Simulink, and SimMechanics:

- Real-Time Workshop
- xPC Target

Working first through “Simulating with Code” on page 9-69, is strongly recommended.

A common step after generating and compiling code from a model is to download the compiled executable to a computer dedicated to running just that application. For a model with a control system, you can download the complete model as a unit or separate the controller and plant into different executables on different computers. You can also execute the controller part as embedded code on a dedicated computer that controls an actual plant. Such application deployments are known as *hardware in the loop* or *rapid prototyping* [8].

xPC Target and Real-Time Workshop allow you to generate and compile code from a SimMechanics model and download it to a computer with an IBM PC-type processor. xPC Target acts as another target within Real-Time Workshop and requires a fixed-step solver. You can use xPC Target to implement controller-plant models in many configurations [9].

This section outlines some model conversion-downloading applications based on the Stewart platform modeled in SimMechanics.

For More Information About xPC Target

Consult the xPC Target documentation for complete instructions on downloading and running executable code in different configurations.

Files Needed for This Study

This study requires mech_stewart_xpc, as well as the initialization M-files.

Adjusting Hardware for Computational Demands

Simulation with a fixed simulation time is subject to the basic tradeoff between accuracy and speed. (See “Improving Performance” on page 5-23.) You can make a simulation more accurate by reducing its step size, but at the expense of creating more time steps and slowing down the real clock time. You can speed up the simulation by increasing the time step size, but you risk losing enough accuracy that the simulation fails to converge.

Real-Time Simulation Tradeoff for SimMechanics

A typical requirement for code running on dedicated processors is that the simulation run in real time. That is, the compiled code should run with

- A finite number of steps (requiring fixed-step solvers)
- Execution time no longer than the physical time being simulated

These requirements are particularly critical for controller code.

With SimMechanics, the accuracy-speed tradeoff is acute. SimMechanics models are computationally intensive and become even more so the more closed loops and constraints you add.

- With dedicated processor execution, reducing the step size ultimately leads to processor overload. The processor needs more clock time to execute a step than the solver time step allows.
- In SimMechanics simulations, convergence failure resulting from too large a time step typically appears as a failure of your simulation to respect constraint tolerances, assembly tolerances, or both.

Simple SimMechanics models require central processor speeds in the mid-hundreds of megahertz (MHz) range. More complex models such as the Stewart platform (with 36 degrees of freedom, as well as 5 independent closed loops and 40 constraints arising from cutting those loops) demand more processor speed, starting in the low gigahertz (GHz) range.

Mitigating the Real-Time Simulation Tradeoff

You have two ways to alleviate the conflict between accuracy and speed in real-time simulation.

- Increase the processor speed. This allows you to reduce the solver step size while keeping the clock time unchanged.
- Break up a complete model into parts, each simulated by its own model downloaded to and executed on a different processor.

Both approaches are complicated by additional factors, such as memory caching and bus speed. Real-time simulation distinguishes between the sample time in signal buses and the solver step size.

Caution Sample time must be a positive integral multiple of solver step size. For SimMechanics models, avoid making sample time larger than step size to prevent simulation convergence failures.

Downloading a Complete Model to the Target

As a trial of running the Stewart platform simulation on dedicated hardware, here you convert a model to code, then download it and run it on an external PC-type computer. The model requires a processor of speed approximately 2 GHz or faster, and a separate target computer monitor.

Consult the xPC Target documentation for details on preparing the target computer, establishing the host-target connection, and interacting with the target from the host.

Setting Up the Target Computer and Host-Target Connection

The results here were obtained with host and target PC-type computers, each with a 3 GHz Pentium 4 processor and 1 gigabyte of RAM, communicating with each other by an RS-232 connection.

To set up the connection and start the target, you need an RS-232 cable and a blank, formatted floppy disk. The target requires a floppy disk drive. You can observe target simulation on a target monitor, your host monitor, or both.

1. Connect the host and target computer to one another with their respective RS-232 ports and a cable.
2. From MATLAB, prepare an xPC Target boot floppy disk.
3. Insert the prepared xPC Target boot disk into the target PC floppy drive. Start the target computer.
4. After the target has finished booting, confirm the host-target connection.

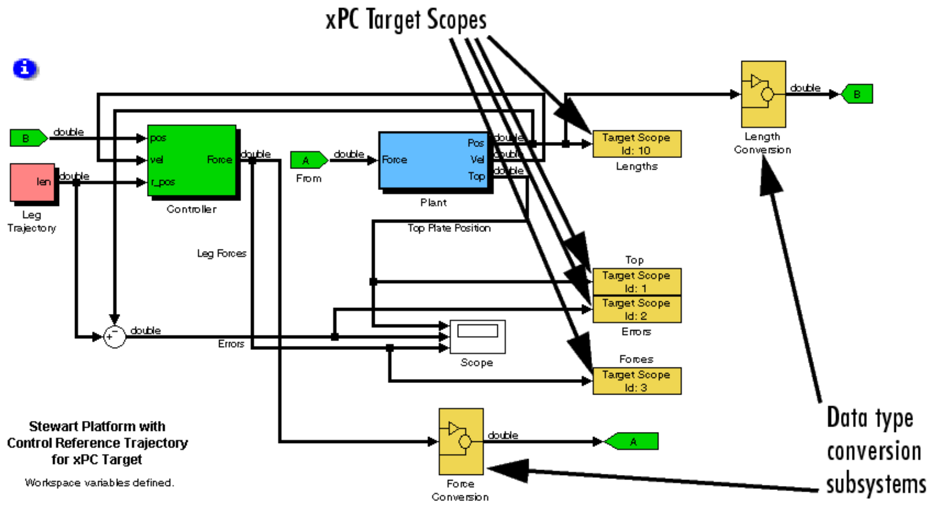
Examining and Running the xPC Model — Data Type Conversion

For this example, you use a variant of the code generation model presented in the preceding study, “Simulating with Code” on page 9-69.

- The model contains xPC Scope blocks for observing the simulation results later. The **Scope type** for each is `Target`. Thus they will appear on the target PC after you download the compiled code.
- The controller and plant work with the default Simulink 64-bit floating double data type. To test the effect of the type conversion needed for passing signals on a hardware bus, the model also contains subsystems that convert these floating doubles to fixed-point integers, then back to doubles.

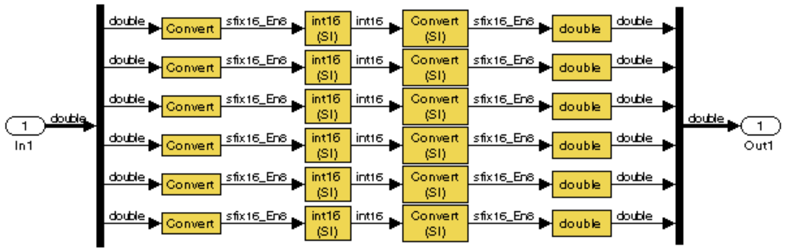
  The data conversion truncates the controller-plant data and changes the simulation behavior somewhat. It is critical to test the impact of such changes before deploying the code to hardware.

1 Open this model, mech_stewart_xpc. Update the diagram (**Ctrl+D**). The vector signals now appear as wide lines and display their data types.



Stewart Platform with Control Reference Trajectory for xPC Target

2 Open the Force Conversion and Length Conversion subsystems. Each subsystem converts a vector signal from floating doubles to 16-bit integers (typical of hardware buses) and back to doubles. These subsystems mimic the effect of hardware buses communicating between controller and plant.

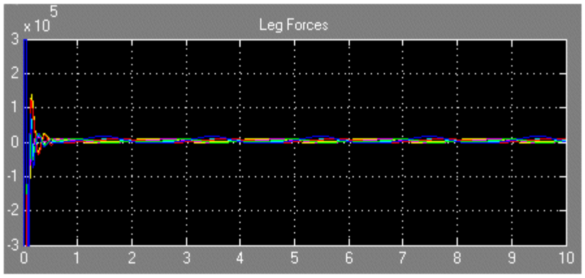


Before the data are converted to integer format, they must be converted from floating to fixed point, truncating the floating double signals. The Data Type Conversion blocks that change doubles to fixed points apply scaling to ensure that information lost to truncation is “small,” as defined by the force and leg length numbers typical of this simulation. These scalings are set in the Data Type Conversion block dialogs.

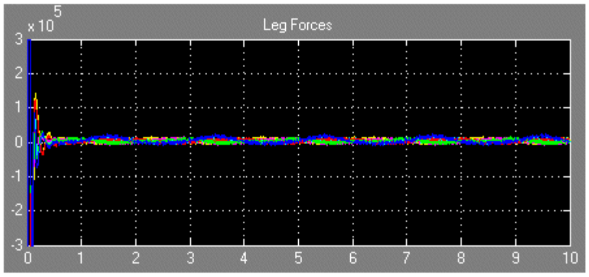
3 Close the Conversion subsystems. Open the Scope.

4 Run the model and observe the motion. Afterward, close the Scope.

The difference between this Stewart platform simulation and earlier ones is clear in the Leg Forces scope trace, which exhibits a small level of “noise” after the initial transient has passed. This “noise” is due to data truncation when the floating doubles are converted to fixed point.



Leg forces without fixed-point truncation



Leg forces with fixed-point truncation

Generating and Downloading Code from the xPC Model

In the next steps, you convert the model to code and download it to the target.

1 Confirm the solver step size (dt1) and sample time (dt2) by entering

```
dt1, dt2
```

at the command line. The values are 5.0 milliseconds (ms).

2. Check the code generation target selection in **Configuration Parameters**, under the **Real-Time Workshop** node, **Target selection > System target file**. The target selection is `xpctarget.tlc`.

   Under the **Real-Time Workshop** node, check the **xPC Target options** entry. Leave these default settings.

3. On the **Real-Time Workshop** panel, click **Build** to start code generation.

   Follow the progress on the command window, as Real-Time Workshop generates and compiles the model, then downloads it to your target computer. When the download is complete, you see the four empty xPC Target scope windows on the target monitor.

Running the xPC Stewart Platform Model on the Target

The xPC Target interface creates an object called `tg` that allows you to control the application on the target machine.

1. Using the xPC Target interface, start the target application.

   The target computer monitor displays the execution. In the Command Window, the xPC Target interface summarizes the execution results.

2. Stop the target application. The Command Window displays the execution summary. The target scopes display the simulation results.

Viewing the Target Simulation with xPC Scopes

xPC Target allows you to observe simulation in various ways. The xPC Target documentation explains the details.

- In the first run, you observed target-type xPC scopes on the target monitor.
- You can change the **Scope type** of one or more xPC scopes to `Host` and observe them on your host computer instead.
- The xPC Target interface also allows you to connect and display such scopes while the simulation is running. You can make connecting and displaying scopes during simulation easier by changing the stop time to infinity (`inf`).

Adjusting the Step and Sample Times — Testing for CPU Overload

You can make your simulation more accurate by reducing the solver step size. But by requiring more steps, you also make the simulation more intensive. If the solver step size drops below the task execution time (TET), the target processor cannot keep up with the simulation and suffers CPU overload.

The xPC Target summary in the Command Window indicates if CPU overload has occurred when you start or stop target object (`tg`) execution.

Test for CPU overload by reducing `dt1` and `dt2`.

1 Enter

```
dt1 = 0.0025; dt2 = 0.0025;
```

2 Build and download the generated code again.

3 Start the target application.

You can understand how close to, or how far into, CPU overload your model is by comparing the TET with the solver/sample time.

- If the TET value is smaller than the sample/solver time, the target processor is able to keep up with the solver.
- If the TET value is larger than the sample/solver time, the target processor cannot keep up with the solver. CPU overload halts target execution.

You can keep reducing the solver/sample time until you cause CPU overload. This point is the limit of your target processor with this model. You can work around CPU overload by

- Using a faster processor. The ratio of TET to sample time indicates roughly how much faster the processor needs to be.
- Increasing the solver/sample time. Be sure not to increase it too much, to avoid simulation convergence failures.

See “Adjusting Hardware for Computational Demands” on page 9-80.

Configuring for Realistic Hardware

Typical goals of downloading compiled code to a dedicated computer are

- Simulating controller and plant in real time
- Embedding a discretized version of the controller code on a dedicated computer that controls an actual plant

Separating Controller and Plant — Bus Communication — Discretization

Controller and plant communicate through a hardware bus configured with a specific data protocol. The xPC Target block library contains communication blocks based on a variety of data protocols matching common hardware buses. In realistic applications, the controller is often already discretized (simulated with discrete states) and requires no conversion from floating point.

The plant simulation remains continuous (not discrete) to better imitate the actual physical system.

Caution You cannot use discrete states with SimMechanics blocks in your model. Discretizing a controller requires separating controller and plant into different models.

Hardware Configuration Possibilities

Choose a model and hardware configuration depending on your needs.

- Separate controller and plant into different subsystems that communicate through a physical bus interfaced with xPC Target bus blocks, rather than normal Simulink signal lines. To run such a model on a target requires the target to have the corresponding hardware card and bus cable.
- Separate controller and plant into two different models that also communicate through a physical bus interfaced with xPC Target bus blocks. You then download the two models to two separate targets that communicate through a bus cable connected to the corresponding hardware cards.

Once you separate controller and plant into different models, you can discretize the controller.

- Embed the controller on a dedicated target that controls an actual Stewart platform. The target and platform communicate through a bus or other I/O hardware corresponding to the blocks used in the controller model.

Mitigating Real-Time Tradeoffs

Real-time simulations are restricted by the tradeoff between accuracy and speed and limited by target execution time and maintaining convergence. You need to ensure that your memory caching and bus, not just your processor(s), are fast enough to cope with the computational demands of SimMechanics. See "Adjusting Hardware for Computational Demands" on page 9-80.

Blocks — By Category

Machines, Bodies, and Grounds (p. 10-2)	Create machines and model bodies
Joints (p. 10-2)	Add degrees of freedom
Constraints and Drivers (p. 10-5)	Remove degrees of freedom
Actuators and Sensors (p. 10-5)	Initiate, impose, and measure mechanical motions
Force Elements (p. 10-6)	Interbody forces
Utilities (p. 10-6)	Miscellaneous useful blocks

Machines, Bodies, and Grounds

Body	Represent customizable rigid body
Ground	Represent immobile point at rest in World
Machine Environment	Set up mechanical environment for machine
Shared Environment	Connect two mechanical components so that they share same mechanical environment

Joints

Assembled Joints (p. 10-2)	Collocated joints
Disassembled Joints (p. 10-4)	Dislocated joints
Massless Connectors (p. 10-4)	Rigidly separated joints

Assembled Joints

Bearing	Represent composite joint with one translational and three rotational DoFs
Bushing	Represent composite joint with three translational and three rotational DoFs
Custom Joint	Represent customizable composite joint with up to three translational and up to three rotational degrees of freedom

Cylindrical	Represent composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes
Gimbal	Represent composite joint with three rotational DoFs
In-Plane	Represent composite joint with two translational DoFs
Planar	Represent composite joint with two translational DoFs and one rotational DoF, with rotational axis orthogonal to plane of translational axes
Prismatic	Represent prismatic joint with one translational degree of freedom
Revolute	Represent assembled revolute joint with one rotational degree of freedom
Screw	Represent composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes and linear pitch constraint between translational and rotational motion
Six-DoF	Represent composite joint with three translational and three rotational DoFs
Spherical	Represent assembled spherical joint with three rotational degrees of freedom
Telescoping	Represent composite joint with one translational and three rotational DoFs

Universal	Represent composite joint with two rotational DoFs
Weld	Represent joint with no DoFs

Disassembled Joints

Disassembled Cylindrical	Represent disassembled cylindrical joint, with one translational DoF and one rotational DoF along and about misaligned axes, with no constraints
Disassembled Prismatic	Represent disassembled prismatic joint with one translational degree of freedom along misaligned axes
Disassembled Revolute	Represent disassembled revolute joint with one rotational degree of freedom about misaligned axes
Disassembled Spherical	Represent disassembled spherical joint with three rotational degrees of freedom about dislocated pivots

Massless Connectors

Revolute-Revolute	Represent composite joint composed of two revolute primitives spatially separated by massless connector of constant length
Revolute-Spherical	Represent composite joint composed of revolute and spherical primitives spatially separated by massless connector of constant length
Spherical-Spherical	Represent composite joint composed of two spherical primitives spatially separated by massless connector of constant length

Constraints and Drivers

Angle Driver	Specify angle between two body axis vectors as function of time
Distance Driver	Specify distance between two Body CS origins as function of time
Gear Constraint	Constrain rotational motion of two bodies to move along tangent pitch circles
Linear Driver	Specify component of vector difference of two Body CS origins as function of time
Parallel Constraint	Constrain body axis vectors of two bodies to be parallel
Point-Curve Constraint	Constrain motion of point on one body to be along curve on another body
Velocity Driver	Specify linear combination of the linear and angular velocities of two bodies as function of time

Actuators and Sensors

Body Actuator	Apply force or torque to body
Body Sensor	Measure body motion
Constraint & Driver Sensor	Measure constraint force or torque between pair of constrained bodies
Driver Actuator	Apply relative motion between a pair of constrained bodies through driver
Joint Actuator	Apply force, torque, or motion to joint primitive

Joint Initial Condition Actuator	Apply initial positions and velocities to primitives of Joint before starting simulation
Joint Sensor	Measure motion of and force or torque on joint primitive
Joint Stiction Actuator	Apply classical friction to joint primitive
Variable Mass & Inertia Actuator	Vary mass and inertia on body at specific body coordinate system as function of time (*not* including thrust force or torque)

Force Elements

Body Spring & Damper	Model damped linear oscillator force between two bodies
Joint Spring & Damper	Model damped linear oscillator force or torque on prismatic or revolute joint between two bodies

Utilities

Connection Port	Create Physical Modeling connector port for subsystem
Continuous Angle	Convert discontinuous, bounded angular output from sensor to continuous, unbounded angular output

Mechanical Branching Bar	Map multiple sensor or actuator lines to one sensor or actuator port on Joint, Constraint, or Driver, or to one Body coordinate system port on Body
RotationMatrix2VR	Convert 3-by-3 rotation matrix to equivalent VRML form of rotation axis and angle

11

Blocks — Alphabetical List

Angle Driver

Purpose

Specify angle between two body axis vectors as function of time

Library

Constraints & Drivers

Description

The Angle Driver block drives axis vectors defined on two Bodies. You specify fixed base and fixed follower body axis vectors $\boldsymbol{a}_B$, $\boldsymbol{a}_F$ in the Body CS on either side of the Driver on each body, then drive the angle between the body axis vectors as a function of time.

The Angle Driver block specifies the angle θ defined by

$$\cos\theta = |\boldsymbol{a}_B \cdot \boldsymbol{a}_F| / (|\boldsymbol{a}_B|\,|\boldsymbol{a}_F|)$$

as a function of time: $\theta = \theta(t=0) + f(t)$. You connect the Angle Driver to a Driver Actuator block.

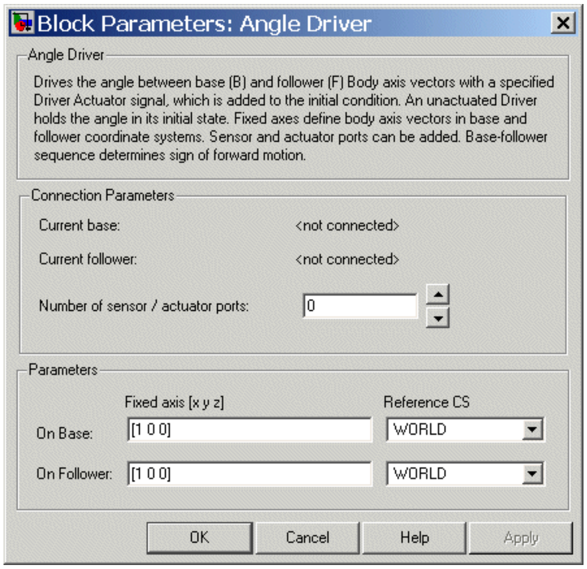
The Simulink input signal into the Driver Actuator specifies the time-dependent driving function *f(t)* and its first two derivatives, as well as their units. If you do not actuate Angle Driver, this block acts as a time-independent constraint that freezes the angle between the two body axes at its initial value $\theta(t=0)$ during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

Note If the two axes come close to aligning, that is, if θ approaches zero, the constraint between the two axes becomes singular, and the simulation slows down. See "How SimMechanics Works" on page 5-15 and "Handling Motion Singularities" on page 5-9.

You can also connect a Constraint & Driver Sensor to any Driver measure the reaction forces/torques between the driven bodies.

Dialog Box and Parameters


Block Parameters: Angle Driver

Angle Driver

Drives the angle between base (B) and follower (F) Body axis vectors with a specified Driver Actuator signal, which is added to the initial condition. An unactuated Driver holds the angle in its initial state. Fixed axes define body axis vectors in base and follower coordinate systems. Sensor and actuator ports can be added. Base-follower sequence determines sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

	Fixed axis [x y z]	Reference CS
On Base:	[1 0 0]	WORLD
On Follower:	[1 0 0]	WORLD

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.

Current base

When you connect the base (B) connector port on the Angle Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Angle Driver Base and Follower Body Connector Ports on page 11-4.

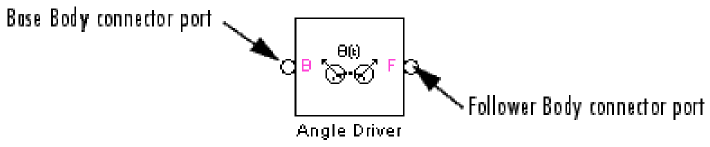
Current follower

When you connect the follower (F) connector port on the Angle Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Angle Driver Base and Follower Body Connector Ports on page 11-4.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is `0`.

To activate the Driver, connect a Driver Actuator.



Angle Driver Base and Follower Body Connector Ports

Parameters

Fixed axis [x y z]

For the **Base** and **Follower** bodies, respectively, enter the body axis vectors. The defaults are `[1 0 0]`.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the **Base** and **Follower** body axis vectors are oriented with respect to. This CS also determines the absolute meaning of reaction forces/torques at this Driver. The defaults are `WORLD`.

See Also

Constraint & Driver Sensor, Driver Actuator, Parallel Constraint, Velocity Driver

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Drivers.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on using drivers in closed loops.

See “Constraints and Drivers” on page 10-5.

Purpose

Represent composite joint with one translational and three rotational DoFs

Library

Joints

Description

The Bearing block represents a composite joint with one translational degree of freedom (DoF) as one prismatic primitive and three rotational DoFs as three revolute primitives. There are no constraints among the primitives. Unlike Telescoping, Bearing represents the rotational DoFs as three revolutes, rather than as one spherical.

Caution A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel ("gimbal lock"). The simulation stops with an error in this case.

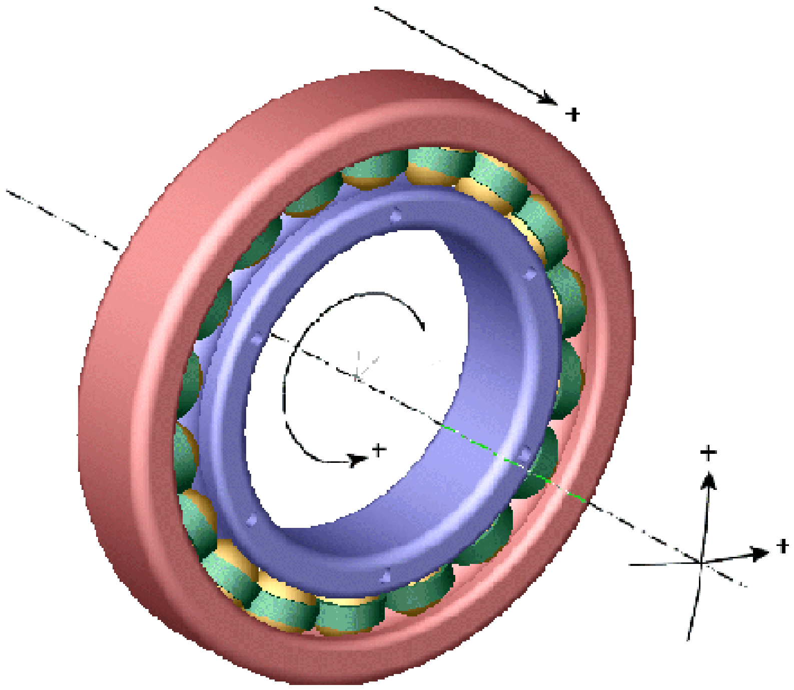
A joint with three revolute primitives must be configured in the initial state with the three revolute primitive axes mutually orthogonal. There are no restrictions on the primitive axes once the simulation starts, except to prevent any two of the primitive axes from becoming parallel.

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Bearing block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

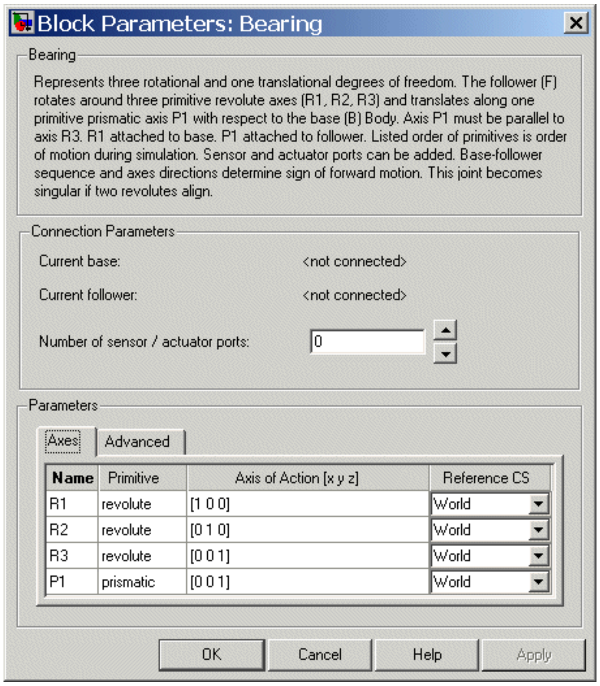
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

Note Bearings are often represented by one translational and one rotational DoF. The Bearing block has *three* rotational degrees of freedom, rather than one, in order to represent transverse "play" in the joint.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Block Parameters: Bearing

Bearing

Represents three rotational and one translational degrees of freedom. The follower (F) rotates around three primitive revolute axes (R1, R2, R3) and translates along one primitive prismatic axis P1 with respect to the base (B) Body. Axis P1 must be parallel to axis R3. R1 attached to base. P1 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if two revolutes align.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[1 0 0]	World
R2	revolute	[0 1 0]	World
R3	revolute	[0 0 1]	World
P1	prismatic	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Bearing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Bearing Base and Follower Body Connector Ports on page 11-8.

The base Body is automatically connected to the first joint primitive R1 in the primitive list in **Parameters**.

Current follower

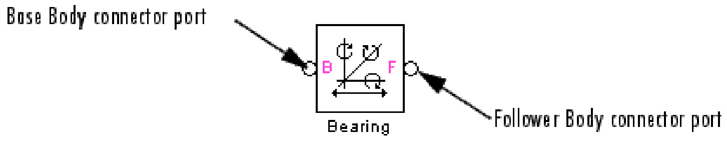
When you connect the follower (F) connector port on the Bearing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Bearing Base and Follower Body Connector Ports on page 11-8.

The follower Body is automatically connected to the last joint primitive P1 in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.



Bearing Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Bearing has an entry line. These lines specify the direction of the axes of action of the DoFs that the Bearing represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Bearing block: revolute primitives R1, R2, R3, and prismatic primitive P1.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

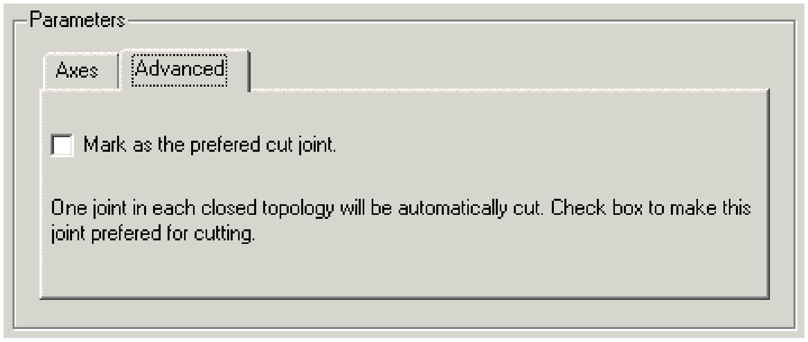
- Prismatic: axis of translation
- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes can be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Bushing, Cylindrical, Gimbal, Prismatic, Revolute

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Purpose

Represent customizable rigid body

Library

Bodies

Description

The Body block represents a rigid body whose properties you customize. The representation you specify includes:

- The body's *mass* and *moment of inertia tensor*
- The coordinates for the body's *center of gravity* (CG)
- One or more *Body coordinate systems* (CSs)

A rigid body is defined in space by the position of its CG (or center of mass) and its orientation in some CS.

Setting Body Initial Conditions The initial position and orientation of a body are set by the entries in its Body dialog that define the body's *home configuration*. These initial conditions remain unchanged, unless, with a Joint Initial Condition Actuator, you change the initial conditions of the Joint(s) connected to the Body prior to starting the simulation, or you actuate the Body with a Body Actuator. The imposition of additional initial conditions defines the *initial configuration* of the body.

Defining a Body with Geometric and Mass Properties

In SimMechanics, you enter the body's properties in two sets, the *geometric properties* and the *mass properties*.

Geometric Properties

The geometric properties are defined by the body's Body CSs.

- The CS with its origin at the CG is required. The CG point specifies both the initial position of the whole body and the origin of the CG CS. You must also orient the CG CS axes.

- You can place one or more additional Body CSs on a body. The Body dialog requires at least one. You must define each Body CS by the position of its origin and the orientation of its CS axes.
- Each connection of a Joint, Constraint/Drive, Actuator, or Sensor block to a Body requires an anchor point on the Body. This anchor point is one of the Body CS origins.
- Body CSs on the block available for connections are shown by Body CS ports ⊞ on the sides of the block. You can show or hide each Body CS on the block sides.
- The set of a body's Body CS origins (including the CG CS) defines the body's *convex hull*, one of the visualization shapes available for representing a body in space.

Mass Properties

The mass properties are defined by the body's mass and inertia tensor.

- The mass is the body's inertia and controls the translational acceleration of the CG in response to an applied force.
- The inertia tensor measures the distribution of mass density in the body and controls the rotational acceleration of the body about the CG in response to an applied torque.
- The components of the inertia tensor control the initial orientation of the body and are always interpreted as being in the CG CS axes. The orientation of the CG CS axes with respect to another CS external to the body (the World CS, a CS on a Ground, or a CS on another Body) then determines the orientation of the body with respect to other bodies or with respect to World.
- The body's inertia tensor defines its *principal axes* and *moments* and its *equivalent ellipsoid*, one of the visualization shapes available for representing a body in space.

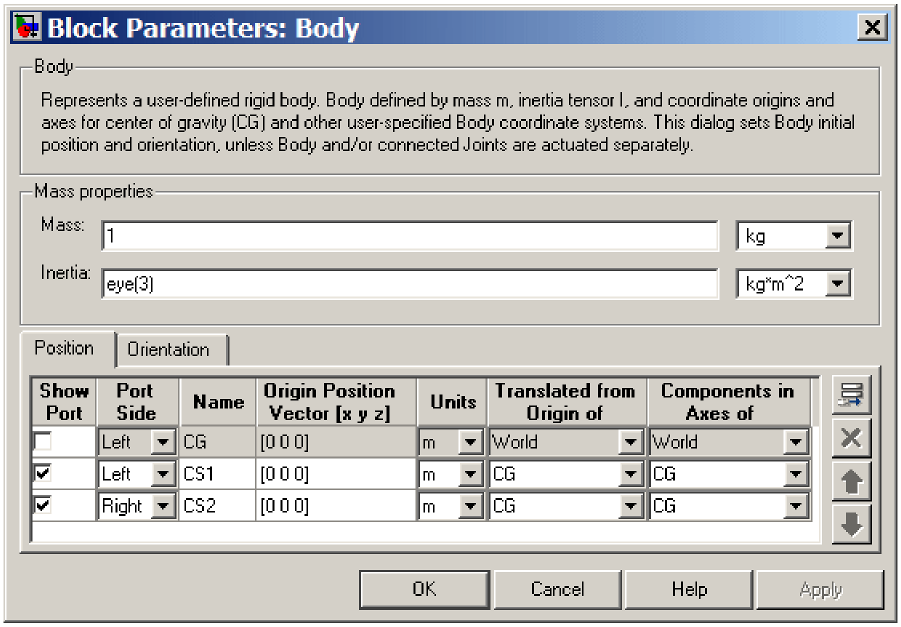
Default Initial State of a Body

These two properties determine a body's initial position and orientation:

- The position of a body's CG sets its initial position.
- The body's inertia tensor components (in the CG CS) and the orientation of the CG CS axes with respect to other CSs in the machine set its initial orientation.

The initial conditions of a machine can be changed with Joint Initial Condition Actuator blocks before you start a simulation. If you do not change the initial state of a Body before simulation, SimMechanics sets its initial position and orientation to its Body dialog entries, defining the body's home configuration. SimMechanics also sets the Body's initial linear/angular velocities to zero in this case.

Dialog Box and Parameters


Block Parameters: Body

Body

Represents a user-defined rigid body. Body defined by mass m, inertia tensor I, and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties

Mass: 1 kg

Inertia: eye(3) kg*m^2

Position | Orientation

Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	CG	CG
✓	Right	CS2	[0 0 0]	m	CG	CG

The dialog has two active areas, **Mass properties** and Body coordinate systems.

Mass Properties

Mass

Enter the mass of the body in the first field and choose units in the pull-down menu to the right. The mass must be a positive, real number or MATLAB equivalent expression. The defaults are `1` and `kg` (kilograms).

Inertia

Enter the inertia tensor (with respect to the Body CG CS axes) in the first field and choose units in the pull-down menu to the right. The tensor must be a 3-by-3 real, symmetric matrix. The default tensor is `eye(3)`, the MATLAB 3-by-3 identity matrix. A zero tensor `zeros(3,3)` defines a point mass. The units default is `kg-m`^{2} (kilograms-meters^2).

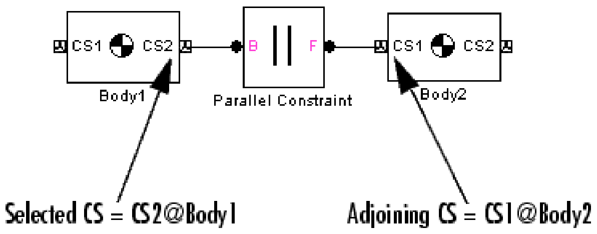
Body Coordinate Systems

Configuring a Body Coordinate System

You set up Body CSs in the Body coordinate systems area:

- The default configuration consists of three Body CSs: the required CG CS attached to the body's CG and two other optional Body CSs, called "CS1" and "CS2," for connecting Joints, Constraints, or Drivers.
- You can configure the CG CS but not delete it. You also cannot create additional CG CSs, although you can duplicate the CG CS with a different name. (See more about Body coordinate systems controls following.)
- The other CSs can be configured or deleted as you want, keeping at least one.
- Configuring a Body CSs requires two groups of steps:
  - Positioning the Body CS origin in the **Position** panel
  - Orienting the Body CS axes in the **Orientation** panel
- Defining Body CSs requires referring to other, pre-existing CSs in the model. In a given Body block, you can refer to Body and Grounded CSs in three ways. The references must be to:
  - World

- Other Body CSs on the same body
- The *Adjoining CS*, the coordinate system on a neighboring body or ground directly connected to the selected Body CS by a Joint, Constraint, or Driver



- You must directly or indirectly define all Body CSs by reference to a Ground or to World. With indirect reference, you specify a Body CS relative to another CS and so on, in a chain of references that ultimately ends in a Ground or World. The CS reference chains of the **Position** and **Orientation** panels can be different. The CS reference chains must not form a closed cycle.
- Toggle between the **Position** or **Orientation** panels with the tabs.

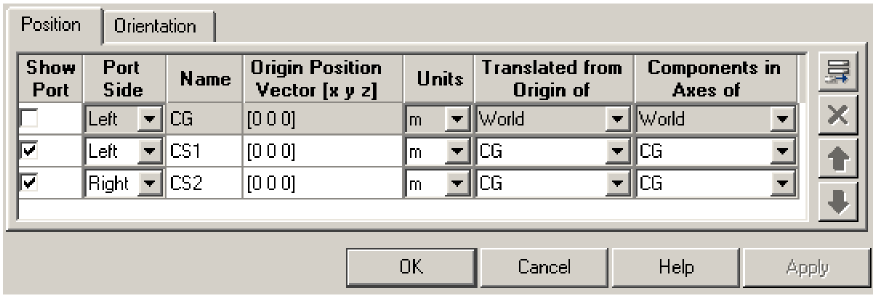
  Each Body CS is labeled with a name, `CG` for the CG CS, and `CS1`, `CS2`, etc., for additional CSs.

Configuring the Position Fields

The **Position** fields for each Body CS specify the position of that CS's origin as a translation vector:

- The numerical components of the vector carry units.
- The translation vector's components are oriented with respect to another set of CS axes.
- The origin is displaced from the origin of another, pre-existing CS in your machine by this translation vector.

Highlight each Body CS to configure it.



Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	CG	CG
✓	Right	CS2	[0 0 0]	m	CG	CG

Origin Position Vector [x y z]
Enter the translation vector that defines the position of the Body CS's origin.

The entry for the CG CS origin positions the entire body.

Units
Choose linear units for the translation vector. The default is m (meters).

Translated from Origin of
In the pull-down menu, choose the other, pre-existing CS in your machine that defines the starting point for the translation vector. The choices are `World`, `Adjoining`, and the other Body CSs on this Body. The ending point of the translation vector is this Body CS's origin.

For the CG CS, the default starting-point CS is `World`. For the additional Body CSs (CS1, CS2, etc.), the default starting point CS is this Body's `CG`.

Components in Axes of
In the pull-down menu, choose the CS whose axes define the orientation of the translation vector's components. The choices are `World`, `Adjoining`, and the other Body CSs on this Body. The

translation vector's components are measured relative to the axes of the CS chosen in this column.

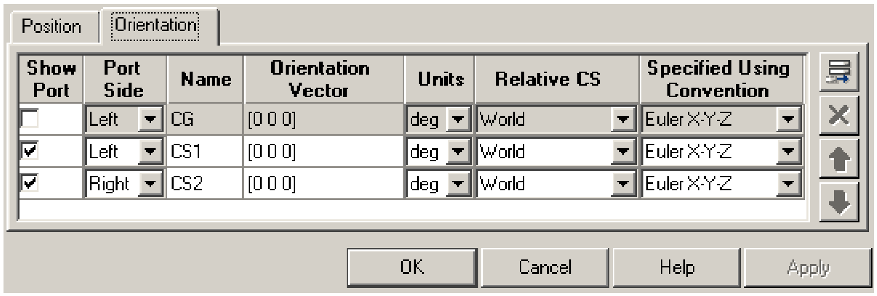
For the CG CS, the default orientation CS is `World`. For the additional Body CSs (CS1, CS2, etc.), the default orientation CS is this Body's `CG`.

Configuring the Orientation Fields

The **Orientation** fields for each Body CS specify the orientation of that CS's triad of axes as a rotation:

- The orientation vector specifying the rotation vector has three components.
- The numerical components of the vector carry units.
- The rotation is oriented with respect to some other, pre-existing set of CS coordinate axes in your machine.
- The orientation vector's components are interpreted in the convention of a rotation representation.

Highlight each Body CS to configure it.



Show Port	Port Side	Name	Orientation Vector	Units	Relative CS	Specified Using Convention
☐	Left	CG	[0 0 0]	deg	World	Euler X-Y-Z
✓	Left	CS1	[0 0 0]	deg	World	Euler X-Y-Z
✓	Right	CS2	[0 0 0]	deg	World	Euler X-Y-Z

Orientation Vector

Enter the components of the rotation that defines the orientation of the Body CS's axes. The geometric meaning of these components is determined by the **Specified Using Convention** column.

The special entry for the CG CS orients the CG CS axes. Together with the **Inertia tensor** entry in **Mass properties**, the CG CS axes orient the whole body with respect to another CS in your machine.

Units

Choose angular units for the rotation, degrees or radians. The default is `deg` (degrees).

Relative CS

In the pull-down menu, choose one of the other pre-existing CSs in your machine to define the starting orientation for the rotation. The choices are `World`, `Adjoining`, and the other Body CSs on this Body.

Specified Using Convention

In the pull-down menu, choose the representation type for the rotation:

Rotation Type	Orientation Vector Components
`Quaternion`	[n_x*sin(θ/2) n_y*sin(θ/2) n_z*sin(θ/2) cos(θ/2)]
`3x3Transform`	3-by-3 orthogonal rotation matrix R
`Euler`	Rotation angles about sequence of three axes defining Euler angle convention [*first-axis second-axis third-axis*]

Rotation Conventions

There are three conventions in a Body block for representing rotations. See "Body Motion in SimMechanics" on page 3-4 and "How SimMechanics Represents Body Orientation" on page 3-11 to learn more about rotations.

- `Euler`

  The Euler angle convention specifies the rotation of the Body CS axes by rotating about three axes in a sequence. The components of the 1-by-3 row vector are the angles of rotation about those three axes, respectively in sequence, in degrees or radians.

  For example, `Euler X-Y-Z` means rotate about the original X axis, then about the first intermediate Y axis, and then about the second intermediate Z axis. Another example: `Euler X-Z-Y` means rotate about the original X axis, then about the first intermediate Z axis, and then about the second intermediate Y axis.

- `3-by-3 Transform`

  The transform convention specifies the rotation as a dimensionless 3-by-3 orthogonal rotation matrix. The inverse of an orthogonal matrix R is equal to its transpose: $R^{-1} = R^{T}$.

  The columns of R are the *(x,y,z)* unit vectors of the Body CS axes. The units menu is inactive.

- `Quaternion`

  The quaternion convention specifies the rotation in angle-axis form as a dimensionless 1-by-4 row vector:

  $$[n_x\text{*}\sin(\theta/2)\ \ n_y\text{*}\sin(\theta/2)\ \ n_z\text{*}\sin(\theta/2)\ \ \cos(\theta/2)]$$

  $\boldsymbol{n} = (n_x,n_y,n_z)$ is a three-component vector of unit length: $\boldsymbol{n}\cdot\boldsymbol{n} = n_x^2 + n_y^2 + n_z^2 = 1$.

  The unit vector $\boldsymbol{n}$ specifies the axis of rotation. The rotation angle about that axis is θ and follows the right-hand rule.

Managing the Body Coordinate Systems List

The Body coordinate system controls (see the following figure, Body Coordinate Systems Controls on page 11-21) allow you to add, reorder, and delete Body CSs on a Body block.

To add a Body CS to the list:

1 Highlight an existing Body CS in the list.

2 Click the **Add** button (see the following figure, Body Coordinate Systems Controls on page 11-21).

   A new Body CS appears immediately below the Body CS you highlighted. New Body CSs are named in sequence after the current ones: CS3, CS4, etc.

To change the position of a Body CS in the list:

1 Highlight the Body CS whose position you want to change.

2 Click on the **Up** or **Down** button (see the following figure, Body Coordinate Systems Controls on page 11-21) until the Body CS is where you want it.

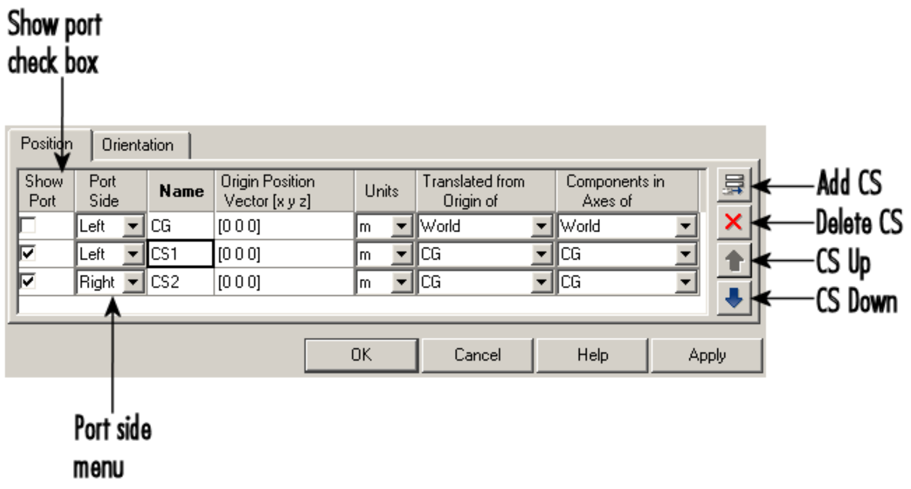
To delete a Body CS from the list:

1 Highlight the Body CS you want to delete.

   You cannot delete the Body's CG CS or the last one of the non-CG CSs.

2 Click on the **Delete** button (see the following figure, Body Coordinate Systems Controls on page 11-21).

   The Body CS you highlighted disappears.



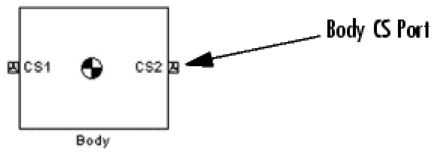
Show Port	Port Side	Name	Origin Position Vector [x y z]	Units	Translated from Origin of	Components in Axes of
☐	Left	CG	[0 0 0]	m	World	World
✓	Left	CS1	[0 0 0]	m	CG	CG
✓	Right	CS2	[0 0 0]	m	CG	CG

Body Coordinate Systems Controls

Managing Body CS Ports on a Body Block

Connecting a Joint, Constraint, Driver, Actuator, or Sensor block to a Body block requires an existing and configured Body CS on that Body:

- These other blocks define, constrain, impart, and measure the motion of bodies with respect to the origin and coordinate axes of Body CSs. Connect each of these blocks to a Body CS with a connection line.
- The actual connection line running from the other block to the Body block must be anchored to a displayed Body CS Port on the side of the Body block in the model window.



- A displayed Body CS Port on a Body block indicates a Body CS with the displayed name configured internally within the Body block.
- Not all the Body CSs configured inside a Body block need to be displayed, however.

  See the Body Coordinate Systems Controls figure preceding.

Show Port

Select this check box for any Body CS to create a corresponding Body CS Port ⊞ on the side of the Body block. The Body CS on that line in the Body CS list is now accessible for connection to other blocks.

Clear this check box to remove the Body CS Port corresponding to that Body CS on that line in the list.

The defaults are not selected for `CG`, selected for `CS1` and `CS2`.

To apply your choices to the displayed Body block, click **Apply**.

Port Side

From the pull-down menu, choose which side of the Body block you want the Body CS Port for that Body CS to be placed on, `Left` or `Right`.

The defaults are `Left` for `CG` and `CS1` and `Right` for `CS2`.

To apply your choices to the displayed Body block, click **Apply**.

See Also

Body Actuator, Body Sensor, Ground, Mechanical Branching Bar

See Chapter 3, "Representing Motion" for more details about representing body position and orientation.

See "Modeling Bodies and Grounds" on page 4-10, "Creating Body CS Ports" on page 4-19, and "Starting SimMechanics Visualization" on page 6-2 for more on setting up Bodies in machines.

See “Modeling Actuators” on page 4-45 for setting general initial conditions (positions and velocities) of DoFs in a machine.

See the relevant entries in the Glossary: **adjoining CS**, **axis-angle rotation**, **body**, **Body CS**, **center of gravity (CG)**, **convex hull**, **coordinate system (CS)**, **equivalent ellipsoid**, **Euler angles**, **inertia tensor**, **mass**, **principal axes**, **principal inertial moments**, **quaternion**, **right-hand rule**, and **rotation matrix**.

Purpose Apply force or torque to body

Library Sensors & Actuators

Description The Body Actuator block actuates a Body block with a generalized force signal, representing a force/torque applied to the body:

- Force for translational motion
- Torque for rotational motion

The generalized force is a function of time specified by a Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block.

The Body Actuator applies the actuation signal in the reference coordinate system (CS) specified in the block dialog.

The inport is the Simulink input signal. The output is the connector port you connect to the Body block you want to actuate.

Note You should carefully distinguish the Body Actuator from the Driver blocks:

- The Body Actuator block applies generalized forces to one body in a specified reference CS.
- The Driver blocks drive relative degrees of freedom between pairs of bodies.

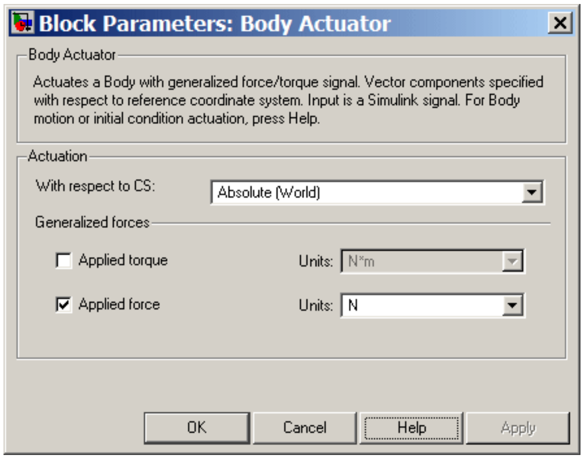
Other Ways to Actuate Bodies

The Body Actuator block actuates a Body with force/torque signals only. To actuate a Body with motion signals or initial conditions, or to drive the relative degrees of freedom between a pair of Bodies, see

"Actuating a Joint" on page 4-52 and "Joint Actuator Example: Body Driver" on page 4-54.

The `mech_body_driver` model from the Demos library shows how to drive the relative DoFs between a pair of bodies. To actuate one body alone, use this model and replace the second Body block with a Ground block. To set body initial conditions, replace the second Body block with a Ground block and the Joint Actuators with Joint Initial Condition Actuators.

Dialog Box and Parameters



Block Parameters: Body Actuator

Body Actuator

Actuates a Body with generalized force/torque signal. Vector components specified with respect to reference coordinate system. Input is a Simulink signal. For Body motion or initial condition actuation, press Help.

Actuation

With respect to CS: Absolute (World)

Generalized forces

☐ Applied torque — Units: N*m

☑ Applied force — Units: N

The dialog has one active area, **Actuation**.

Actuation

With respect to CS

In the pull-down menu, choose the coordinate system (CS) in which the actuating force/torque is interpreted: either the `Local (Body CS)` to which the Actuator is connected or the default `Absolute (World)`.

Generalized Forces

You can apply a force, a torque, or both generalized forces to a body.

If you apply both, you need to bundle the torque and force vectors into a 6-component signal, in the order shown in the dialog.

Applied torque
Select the check box if part or all of the actuating signal is a rotational torque. The default is not selected. The Simulink torque input is a 3-component bundled signal.

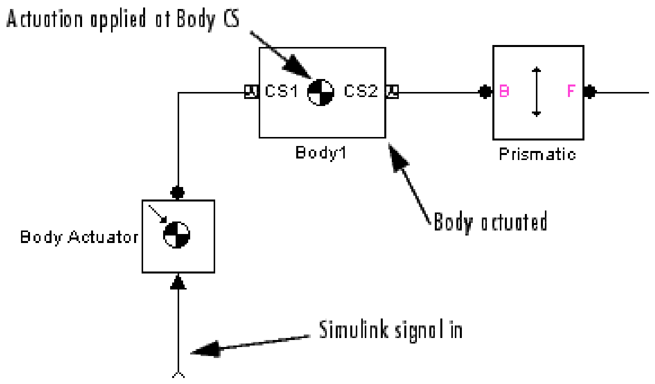
In the **Units** pull-down menu, choose units for the actuating torque. The default is `N*m` (newton-meters).

Applied force
Select the check box if part or all of the actuating signal is a translational force. The default is selected. The Simulink force input is a 3-component bundled signal.

In the **Units** pull-down menu, choose units for the actuating force. The default is `N` (newtons).

Example

Here is a Body Actuator connected to a Body:



You must connect the Body Actuator to the Body at one of that Body's attached Body CSs, at the corresponding Body CS Port. The actuation signal acts on the Body at that Body CS's origin.

See Also

Body, Body Sensor, Driver Actuator, Joint Actuator, Joint Initial Condition Actuator, Mechanical Branching Bar

See “Body Motion in SimMechanics” on page 3-4 for more details on body coordinate system rotations.

See “Actuating a Joint” on page 4-52 and “Joint Actuator Example: Body Driver” on page 4-54.

See “Machines, Bodies, and Grounds” on page 10-2 and “Constraints and Drivers” on page 10-5.

In Simulink, see the Signal Routing Library and the Sources Library.

Body Sensor

Purpose

Measure body motion

Library

Sensors & Actuators

Description

The Body Sensor block senses the motion of a body represented by a Body block. You connect the Body Sensor to a Body coordinate system (CS) on the Body whose motion you want to sense. The sensor specifically measures the motion of the origin of this Body CS.

The Body Sensor measures the components of translational and rotational motion in any combination of:

- Translational position, velocity, and acceleration vectors. The position vector has its tail at the World CS origin.
- Rotational orientation (a 3-by-3 rotation matrix R) and angular velocity and acceleration vectors

In the block dialog, you choose the reference coordinate system (CS) axes in which these components are represented.

The input is the connector port connected to the Body being sensed. The outport is a set of Simulink signals or one bundled Simulink signal of the selected matrix and/or vector components.

Body Position-Orientation and the Home Configuration

The Body Sensor block can measure the position and/or orientation of a body. It measures these relative to the home configuration of the machine, the machine state *before* the application of initial condition actuators and assembly of disassembled joints. Thus the Body Sensor includes the effect of the latter, which act before the simulation starts.

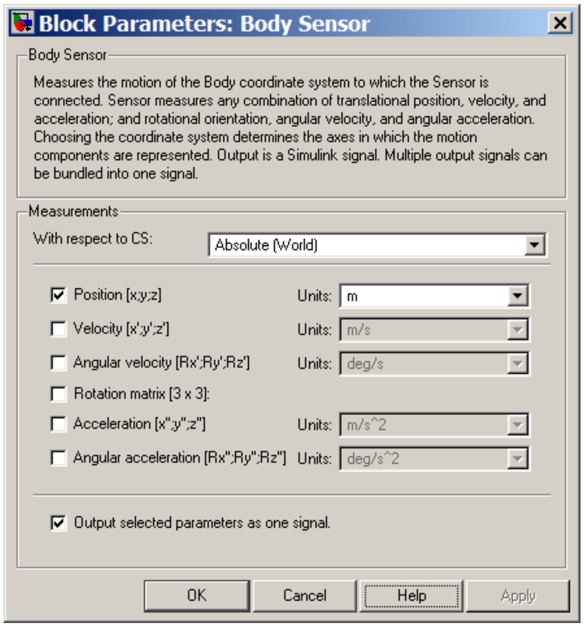
Defining Coordinate Representations and Body Orientation

A body's orientation rotation matrix R relates the components of the same vector $\boldsymbol{v}$ as measured in the inertial World CS and in the Body CS by $\boldsymbol{v}_\mathrm{b} = [R^\mathrm{T}]\cdot\boldsymbol{v}_\mathrm{W}$. The column vector $\boldsymbol{v}_\mathrm{W}$ lists the vector $\boldsymbol{v}$'s three components measured in the World CS. The column vector $\boldsymbol{v}_\mathrm{b}$ lists the vector $\boldsymbol{v}$'s three components measured in the Body CS.

The columns of the rotation matrix *R* are the components of the Body CS unit basis vectors measured with respect to the World axes.

See "Body Motion in SimMechanics" on page 3-4 and "How SimMechanics Represents Body Orientation" on page 3-11 for more details on representing body position and orientation, rotation matrices, and angular velocity.

Dialog Box and Parameters



Block Parameters: Body Sensor

Body Sensor

Measures the motion of the Body coordinate system to which the Sensor is connected. Sensor measures any combination of translational position, velocity, and acceleration; and rotational orientation, angular velocity, and angular acceleration. Choosing the coordinate system determines the axes in which the motion components are represented. Output is a Simulink signal. Multiple output signals can be bundled into one signal.

Measurements

With respect to CS: Absolute (World)

☑ Position [x;y;z]	Units: m
☐ Velocity [x';y';z']	Units: m/s
☐ Angular velocity [Rx';Ry';Rz']	Units: deg/s
☐ Rotation matrix [3 x 3]:	
☐ Acceleration [x'';y'';z'']	Units: m/s^2
☐ Angular acceleration [Rx'';Ry'';Rz'']	Units: deg/s^2

☑ Output selected parameters as one signal.

The dialog has one active area, **Measurements**.

Measurements

With respect to CS

In the pull-down menu, choose the coordinate system in which the body motion components are represented: either the `Local (Body CS)` to which the Sensor is connected or the default `Absolute (World)`.

In the `Absolute` case, the rotation matrix R and the motion vectors have components represented in the inertial World CS axes. In the `Local` case, the same body motion components are premultiplied by the body's inverse orientation rotation matrix $R^{-1} = R^T$.

Each vector measurement is a row vector in the Simulink output signal. The selected signals are ordered in the same sequence as the dialog.

Select the check box for each of the possible measurements you want to make:

- Translational motion: **Position**, **Velocity**, and **Acceleration** vectors: $\boldsymbol{r}$, $\boldsymbol{v} = \mathrm{d}\boldsymbol{r}/\mathrm{dt}$, and $\boldsymbol{a} = \mathrm{d}\boldsymbol{v}/\mathrm{dt}$, respectively.
- Rotational motion: **Angular velocity** and **Angular acceleration** vectors and **Rotation matrix**:
  - The **Rotation matrix** is the 3-by-3 orthogonal rotation matrix R:

$$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

    representing rotational orientation and satisfying $R^TR = RR^T = I$. The components are output columnwise as a 9-component row vector: $(R_{11}, R_{21}, R_{31}, R_{12}, \ldots)$.
  - If you choose the **With respect to coordinate system** as `Absolute (`World, the **Rotation matrix** measures the body's rotational orientation with respect to the World CS. Recall the relationship of vector components in the World and body coordinate axes, $\boldsymbol{v}_W = R]\cdot\boldsymbol{v}_b$.
  - If you choose the **With respect to coordinate system** as `Local (Body CS)`, the **Rotation matrix** returns the 3-by-3 identity matrix $R^TR = I$.

- The angular velocity is $\omega_j = (1/2)\Sigma_{ik}\ \varepsilon_{ijk}\Omega_{ik}$, where the matrix $\Omega = +(dR/dt)*R^T = -R*(dR^T/dt)$, and ε is the permutation symbol. The angular acceleration is $\alpha = d\omega/dt$.

In the **Units** pull-down menus, choose the units for each of the measurements you want:

- Translation: the defaults are `m` (meters), `m/s` (meters/second), and `m/s`^{2} (meters/second^2), respectively, for **Position**, **Velocity**, and **Acceleration**.
- Rotation: the defaults are `deg/s` (degrees/second) and `deg/s`^{2} (degrees/second^2), respectively, for **Angular velocity** and **Angular acceleration**. The **Rotation matrix** is dimensionless.

Output selected parameters as one signal

Select this check box to convert the output signals into a single bundled signal. The default is selected. If you clear it, the Body Sensor block will grow as many Simulink outports as there are active signals selected, one port for each selected signal.

If the check box is selected, the Simulink signal out has all the active (selected) signals ordered into a single row vector, in the same order you see in the dialog. Nonselected components are removed from the vector signal.

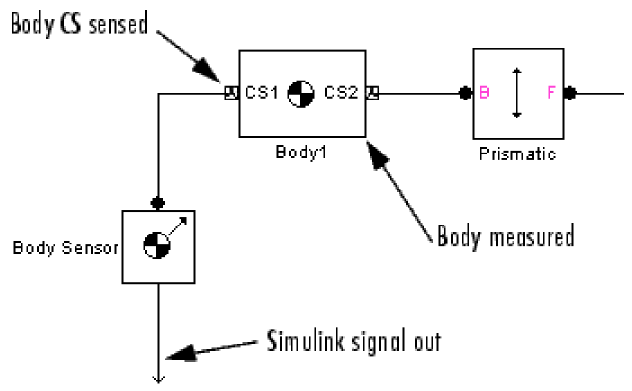
The sensor outputs are ordered and labeled as follows.

Body Sensor Output Signal	Label
Position	p
Velocity	v
Angular velocity	av
Rotation matrix	[R]

Body Sensor Output Signal	Label
Acceleration	a
Angular velocity	aa

Example

Here is a Body Sensor connected to a Body:



You must connect the Body Sensor to the Body at one of its Body CS ports. The Sensor measures the motion of that Body CS.

See Also

Body, Body Actuator, Constraint & Driver Sensor, Joint Sensor, Mechanical Branching Bar

See “Kinematics and the Machine’s State of Motion” on page 3-2, “Body Motion in SimMechanics” on page 3-4, and “How SimMechanics Represents Body Orientation” on page 3-11 for more details on representing body position and orientation.

See “Modeling Sensors” on page 4-63.

See the relevant entries in the Glossary about body orientation: **axis-angle rotation**, **Euler angles**, **right-hand rule**, and **rotation matrix**.

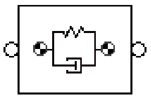
In Simulink, see the Signal Routing Library and the Sinks Library.

Body Spring & Damper

Purpose Model damped linear oscillator force between two bodies

Library Force Elements

Description



The Body Spring & Damper block models the force of a damped spring acting between two bodies. By Newton's third law, the spring applies equal and opposite forces to the two bodies. You can use this Force Element block to model any linear (Hooke's law) force with constant coefficients that acts between a pair of bodies.

You connect a Body Spring & Damper between two Body coordinate systems (CSs), each on one body. The vector between the Body CSs defines the direction and length of the spring. One of the Bodies can be a Ground.

Note The spring and the damper forces act *only* along the axis connecting the two Body CSs.

Grounding the Connected Submachines

The Body Spring & Damper block contains a Shared Environment block. The submachines connected to either side of this block constitute a single composite machine that requires exactly one Machine Environment block, but at least one Ground for *each* submachine.

Referencing Coordinate Systems on the Connected Bodies

The Body Spring & Damper block is not a Joint and cannot propagate adjoining coordinate systems from a Body on one side to a Body on the other side.

One Body is connected to one side of the Body Spring & Damper at one of that Body's CSs. If you attempt to define that CS in terms of the adjoining CS (the connected CS of the other Body connected to the other side), the first Body cannot detect the connected CS of the second body. If you need to define adjoining CSs on either side of a Body Spring & Damper, add a Joint block in parallel with the spring-damper.

Adding Joints in Parallel to the Body Spring & Damper

Caution The Body Spring & Damper has no degrees of freedom (DoFs).

To represent the DoFs of one body with respect to the other, either

- Connect one or more Joints in series with the Bodies.
- Create additional Body CSs on each body and connect them with a Joint in parallel with the Body Spring & Damper. To create parallel grounds, insert additional Ground blocks.

  You can add more Joint blocks between the Bodies to represent one, two, or three prismatic primitives. Use Prismatic blocks or a Custom Joint block to accomplish this.

Body Spring and Damper Force Law

You connect this block to each Body, A or B, at a Body coordinate system (CS). If $\boldsymbol{r}_\mathrm{A}$ and $\boldsymbol{r}_\mathrm{B}$ are the positions of these Body CSs, the relative position vector connecting them is $\boldsymbol{r} = \boldsymbol{r}_\mathrm{B} - \boldsymbol{r}_\mathrm{A}$. The distance of separation is $|\boldsymbol{r}|$. The relative velocity is $\boldsymbol{v} = d\boldsymbol{r}/dt$. Then the vector force that body A exerts on body B is

$$\boldsymbol{F} = -k(|\boldsymbol{r}| - r_0)(\boldsymbol{r}/|\boldsymbol{r}|) - b(\boldsymbol{v}\cdot\boldsymbol{r})(\boldsymbol{r}/|\boldsymbol{r}|^2)$$

The first term represents the spring or linear displacement force. The second represents the damper or velocity dissipation force, which acts only along the direction of $\boldsymbol{r}$. Thus the damper is equivalent to a dashpot, not a viscous medium.

You specify

- The spring constant k. A stable spring requires $k > 0$.

- The natural spring length (offset) r_0. The natural length is the length of the spring with no forces acting on it and physically must be nonnegative: $r_0 \geq 0$.
- The damping constant b. A damping representing dissipation and respecting the second law of thermodynamics requires $b \geq 0$. You can use a negative b to represent energy pumping.

Body Spring and Damper Force in Singular Cases

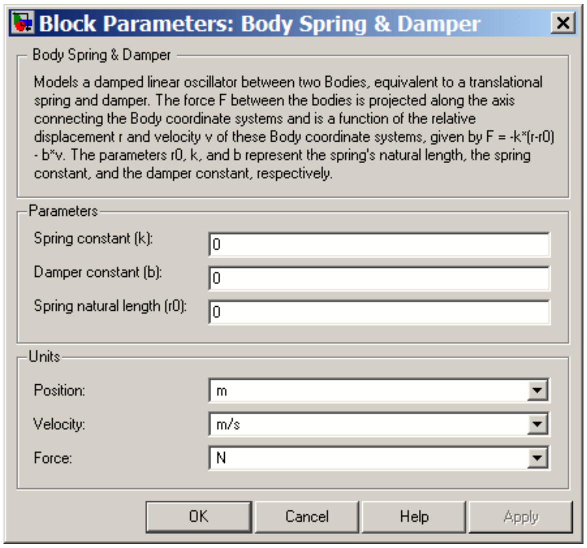
Caution In certain cases, the force formula breaks down, and SimMechanics uses special rules to determine the spring-damper force.

To avoid singularities in the initial state of motion, be sure to set the bodies' initial conditions of position and velocity to physically sensible values.

These cases include the following:

- If both r_0 and $\boldsymbol{v} \neq 0$, and $\boldsymbol{r} = 0$ at some instant, both terms in the force become singular. The spring force is reprojected along the velocity vector. That is, $\boldsymbol{v}/|\boldsymbol{v}|$ replaces $\boldsymbol{r}/|\boldsymbol{r}|$ in both terms of the force law, once in the first term and twice in the second. If the state $\boldsymbol{r} = 0$ does not persist for more than an instant, this replacement has no effect on the motion.
- If $r_0 \neq 0$, and both $\boldsymbol{r}$ and $\boldsymbol{v} = 0$ at some instant, the force direction is undefined. The simulation stops with an error.

Dialog Box and Parameters



Block Parameters: Body Spring & Damper

Body Spring & Damper

Models a damped linear oscillator between two Bodies, equivalent to a translational spring and damper. The force F between the bodies is projected along the axis connecting the Body coordinate systems and is a function of the relative displacement r and velocity v of these Body coordinate systems, given by F = -k*(r-r0) - b*v. The parameters r0, k, and b represent the spring's natural length, the spring constant, and the damper constant, respectively.

Parameters

Spring constant (k): 0

Damper constant (b): 0

Spring natural length (r0): 0

Units

Position: m

Velocity: m/s

Force: N

The dialog has two active areas, **Parameters** and **Units**.

Parameters

Spring constant (k)

Enter the linear spring force constant k. The default is 0.

The units for k are derived implicitly from your choice of position and force units.

Damper constant (b)

Enter the linear damping force constant b. The default is 0.

The units for b are derived implicitly from your choice of velocity and force units.

Spring natural length (r0)

Enter the spring's natural length (offset) r_0. The default is 0.

Units

Position

In the pull-down menu, choose units for the relative position vector $\boldsymbol{r}$. The default is m (meters).

Velocity

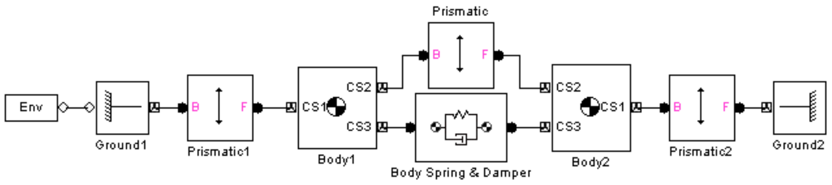
In the pull-down menu, choose units for the relative velocity vector $\boldsymbol{v}$. The default is `m/s` (meters/second).

Force

In the pull-down menu, choose units for the spring-damper force $\boldsymbol{F}$ acting between the bodies. The default is `N` (newtons).

Example

This is a simple but representative use of the Body Spring & Damper.



See Also

Body, Body Actuator, Body Sensor, Custom Joint, Ground, Joint Spring & Damper, Machine Environment, Prismatic, Shared Environment

See “Modeling Force Elements” on page 4-69.

Bushing

Purpose

Represent composite joint with three translational and three rotational DoFs

Library

Joints

Description

The Bushing block represents a composite joint with three translational degrees of freedom (DoFs) as three prismatic primitives and three rotational DoFs as three revolute primitives. There are no constraints among the primitives. Unlike Six-DoF, Bushing represents the rotational DoFs as three revolutes, rather than as one spherical.

Caution A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). A joint with two or three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with errors in these cases.

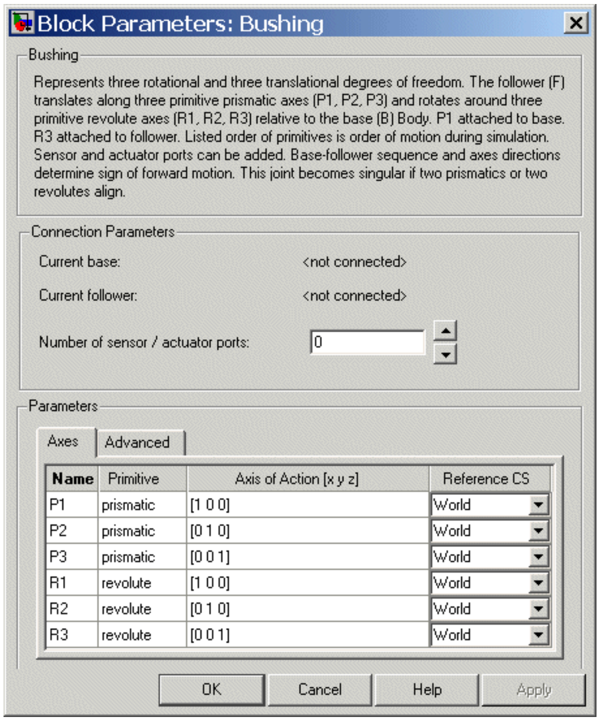
A joint with three revolute primitives must be configured in the initial state with the three revolute primitive axes mutually orthogonal. There are no restrictions on the primitive axes once the simulation starts, except to prevent any two of the primitive axes from becoming parallel.

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Bushing block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.

Dialog Box and Parameters



Block Parameters: Bushing

Bushing

Represents three rotational and three translational degrees of freedom. The follower (F) translates along three primitive prismatic axes (P1, P2, P3) and rotates around three primitive revolute axes (R1, R2, R3) relative to the base (B) Body. P1 attached to base. R3 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if two prismatics or two revolutes align.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[1 0 0]	World
P2	prismatic	[0 1 0]	World
P3	prismatic	[0 0 1]	World
R1	revolute	[1 0 0]	World
R2	revolute	[0 1 0]	World
R3	revolute	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Bushing Base and Follower Body Connector Ports on page 11-41.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

Current follower

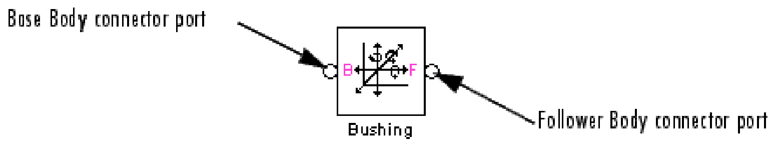
When you connect the follower (F) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Bushing Base and Follower Body Connector Ports on page 11-41.

The follower Body is automatically connected to the last joint primitive R3 in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.



Bushing Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Bushing has an entry line. These lines specify the direction of the axes of action of the DoFs that the Bushing represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Bushing block: prismatic primitives P1, P2, P3, and revolute primitives R1, R2, R3.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

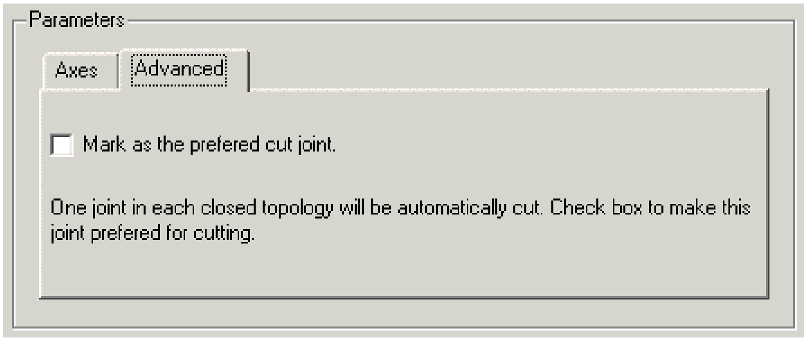
- Prismatic: axis of translation
- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes and no two of the prismatic axes can be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Bearing, Cylindrical, Gimbal, Prismatic, Revolute, Six-DoF

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Connection Port

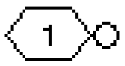
Purpose

Create Physical Modeling connector port for subsystem

Library

Utilities

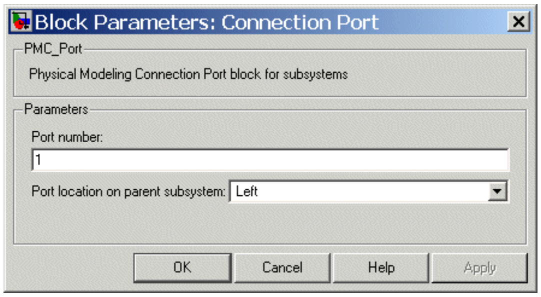
Description



The Connection Port block, placed inside a subsystem composed of SimMechanics blocks, creates a SimMechanics open round connector port ○ on the boundary of the subsystem. Once connected to a connection line, the Port becomes solid ●.

You connect individual SimMechanics blocks and subsystems made of SimMechanics blocks to one another with SimMechanics connection lines, instead of normal Simulink signal lines. These are anchored at the open, round connector ports ○. Subsystems constructed out of SimMechanics blocks automatically have such open round connector ports. You can add additional connector ports by adding Connection Port blocks to your subsystem.

Dialog Box and Parameters


Block Parameters: Connection Port

PMC_Port

Physical Modeling Connection Port block for subsystems

Parameters

Port number:

1

Port location on parent subsystem: Left

Port number
: This field labels the subsystem connector port created by this block. Multiple connector ports on the boundary of a single subsystem require different numbers as labels. The default value for the first Port is `1`.

Port location on parent subsystem
: Choose here on which side of the parent subsystem boundary the Port is placed. The choices are `Left` or `Right`. The default choice is `Left`.

See Also

In Simulink, see Creating Subsystems.

Constraint & Driver Sensor

Purpose Measure constraint force or torque between pair of constrained bodies

Library Sensors & Actuators

Description The Constraint & Driver Sensor block measures the force/torque of constraint (reaction force/torque) between a pair of bodies. You connect this block to the Constraint or Driver block connected between the two Bodies. The output signal is the reaction force/torque.

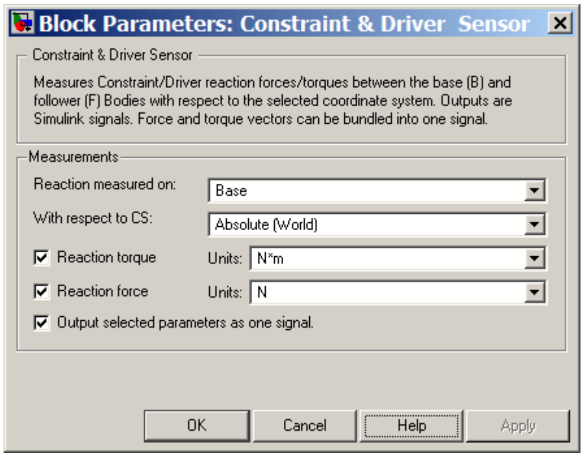
The Constraint & Driver Sensor measures the reaction force/torque in the reference coordinate system (CS) specified in the block dialog. The Constraint or Driver block connects a base and a follower Body. You choose in the dialog to measure the reaction force/torque on either the base or the follower Body.

The input is the connector port connected to the Constraint or Driver block you want to sense. The outport is a set of Simulink signals or one bundled Simulink signal of the reaction force/torque vector(s).

Physical and Unphysical Reaction Forces Not all the components of the output reaction force/torque signal are significant. Only those components projected into the subspace of the degrees of freedom constrained or driven by the connected Constraint or Driver block are physical. Components orthogonal to the constrained or driven degrees of freedom are unphysical.

A body's orientation rotation matrix R relates vector components measured in the body CS and in the inertial World CS by $[R]\cdot\boldsymbol{v}_{\mathrm{b}} = \boldsymbol{v}_{\mathrm{s}}$. The column vector $\boldsymbol{v}_{\mathrm{b}}$ lists the vector $\boldsymbol{v}$'s three components measured in the body CS. The column vector $\boldsymbol{v}_{\mathrm{s}}$ lists the vector $\boldsymbol{v}$'s three components measured in the World CS.

Dialog Box and Parameters


Block Parameters: Constraint & Driver Sensor

Constraint & Driver Sensor

Measures Constraint/Driver reaction forces/torques between the base (B) and follower (F) Bodies with respect to the selected coordinate system. Outputs are Simulink signals. Force and torque vectors can be bundled into one signal.

Measurements

Reaction measured on: Base

With respect to CS: Absolute (World)

☑ Reaction torque Units: N*m

☑ Reaction force Units: N

☑ Output selected parameters as one signal.

The dialog has one active area, **Measurements**.

Measurements

Reactions measured on

In the pull-down menu, choose to measure the reaction force/torque on the base (B) or follower (F) Body. The default is `Base`.

With respect to CS

In the pull-down menu, choose the CS in which the reaction force/torque or motion is interpreted. The default is `Absolute (World)`.

In the `Absolute` case, the force vectors have components measured relative to the inertial World CS axes. In the `Local` case, the same force vector signals are premultiplied by the inverse rotation matrix $R^{-1} = R^{T}$ for the Body selected in **Reactions measured on**.

Reaction torque

Select the check box if you want to measure the reaction torque. The default is selected. The torque is a row vector in the Simulink output signal.

In the pull-down menu, choose the units for the reaction torque. The default is `N*m` (newton-meters).

Reaction force

Select the check box if you want to measure the reaction force. The default is selected. The force is a row vector in the Simulink output signal.

In the pull-down menu, choose the units for the reaction force. The default is `N` (newtons).

Output selected parameters as one signal

Select this check box to convert the output signals into a single bundled signal. The default is selected. If you clear it, the Constraint & Driver Sensor block will grow as many Simulink outports as there are active signals selected, one port for each selected signal.

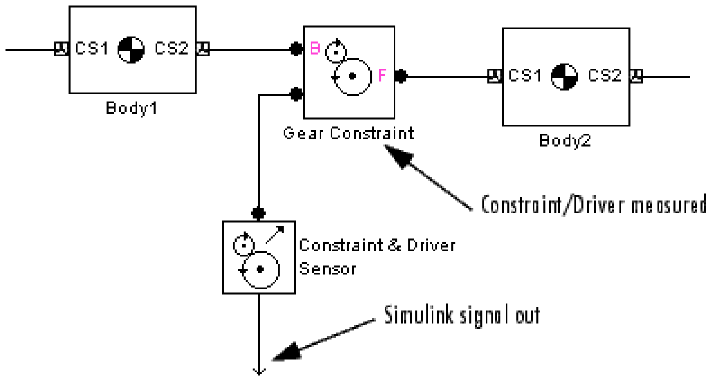
If the check box is selected, the Simulink signal out has all the active signals bundled into a single row vector, ordered in the order shown in the dialog. The type of the signal components depends on which measurements are active (selected).

The sensor outputs are ordered and labeled as follows.

Constraint & Driver Sensor Output Signal	Label
Reaction torque	Tr
Reaction force	Fr

Example

Here is a Constraint & Driver Sensor connected to a Gear Constraint, which connects and constraints two Bodies:



You must add a Sensor port (connector port) to the Constraint/Driver block to connect the Constraint & Driver Sensor to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Constraint & Driver Sensor data.

See Also

Body Sensor, Driver Actuator, Joint Sensor, Mechanical Branching Bar

See "Body Motion in SimMechanics" on page 3-4 and "Modeling Sensors" on page 4-63.

In Simulink, see the Signal Routing Library and the Sinks Library.

Continuous Angle

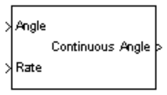
Purpose

Convert discontinuous, bounded angular output from sensor to continuous, unbounded angular output

Library

Utilities

Description



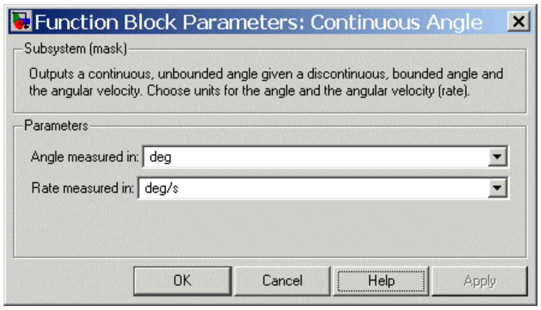
The Continuous Angle block converts a measured angle signal restricted to the semiopen interval (-180°, +180°] degrees or (-π,+π] radians to a continuous, unbounded angle not restricted to any interval. This block requires the angle and the angular velocity as input signals. The continuous, unbounded angle is the output signal.

Caution Each Continuous Angle block in a model adds an additional Simulink state to the model. Use this block with caution if you are trimming or linearizing your model.

The Continuous Angle block does not add additional mechanical states.

The Joint Sensor block outputs the absolute rotational measurement of revolute motion as a bounded angle in the interval (-180°, +180°] degrees or (-π,+π] radians. Motion that crosses the boundaries of this interval causes discontinuities in the measured angle, from +180° to -180° or vice versa. Use the Continuous Angle block if you want to convert this restricted angular measurement to an unbounded measurement.

Dialog Box and Parameters



Function Block Parameters: Continuous Angle

Subsystem (mask)

Outputs a continuous, unbounded angle given a discontinuous, bounded angle and the angular velocity. Choose units for the angle and the angular velocity (rate).

Parameters

Angle measured in: deg

Rate measured in: deg/s

The dialog has one active area, **Parameters**.

Parameters

Angle measured in

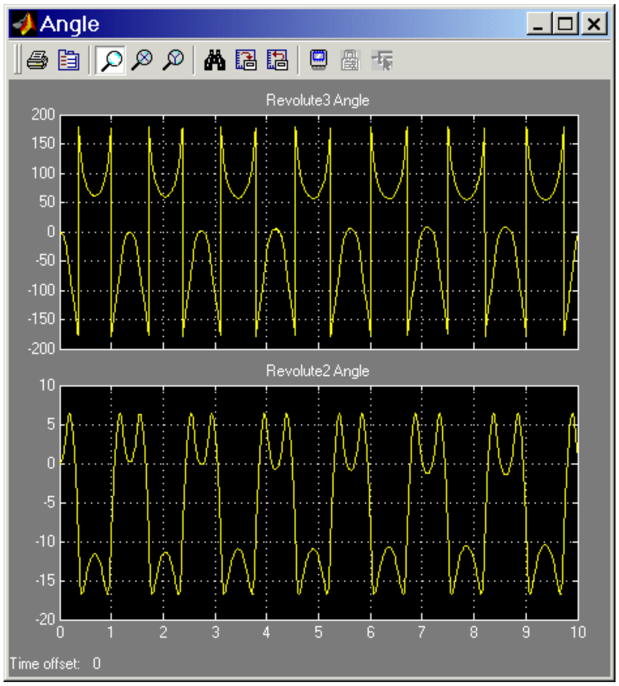
Choose the units for the input angle and the output continuous angle, either `deg` (degrees) or `rad` (radians). The default is `deg`.

Rate measured in

Choose the units for the input rate (angular velocity), either `deg/s` (degrees/second) or `rad/s` (radians/second). The default is `deg/s`

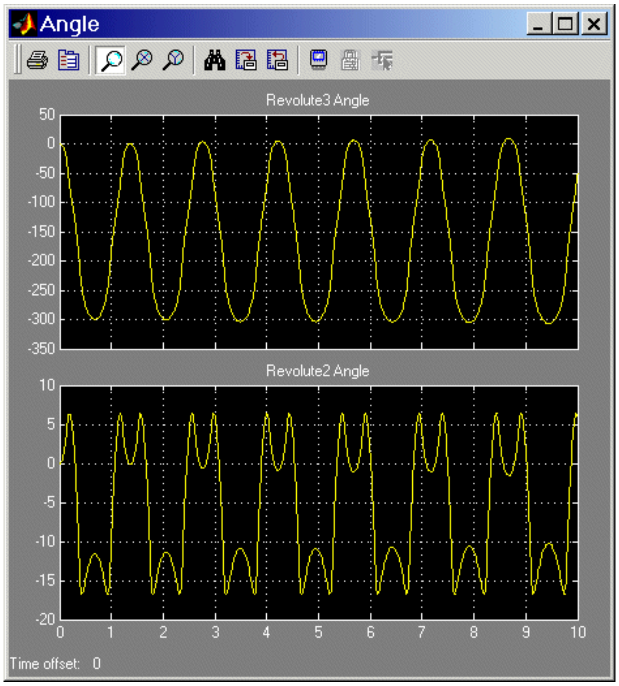
Example

The tutorial “Four Bar Mechanism” produces this angular motion output for the Revolute3 and Revolute 2 joints:



The Revolute3 angle is restricted to the interval (-180°, +180°], so values passing either limit of this interval are mapped to the opposite end of the interval. The Revolute2 angle is not restricted, but instead touches genuine turning points in its motion.

After passing the angles and angular velocities through Continuous Angle blocks, the Revolute3 angular motion appears different:



Revolute3's motion is unchanged, but its angle is now continuous, with no interval restriction. Revolute2's angle is unchanged.

See Also

Joint Sensor

See "Trimming Mechanical Models" on page 8-18 and "Linearizing Mechanical Models" on page 8-32 for more about states.

See "Utilities" on page 10-6.

Purpose

Represent customizable composite joint with up to three translational and up to three rotational degrees of freedom

Library

Joints

Description

The Custom Joint block is a composite joint that you can customize with a specified combination of primitives (prismatic, revolute, or spherical) representing the most general and unconstrained degrees of freedom (DoFs) in three dimensions:

- Up to three translational DoFs as three prismatic primitives
- Up to three rotational DoFs:
  - As a single spherical primitive
  - As one, two, or three revolute primitives

  The sense of rotational DoFs is defined by the right-hand rule. One spherical or three revolutes together form a right-handed system.

You can add, configure, and delete these primitives from the Custom Joint, with a minimum and default of one primitive. The properties of each primitive are the same as the individual Joints of the same names.

Caution A joint with two or three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). A joint with two or three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with errors in these cases.

A joint with three revolute primitives must be configured in the initial state with the three revolute primitive axes mutually orthogonal. There are no restrictions on the primitive axes once the simulation starts, except to prevent any two of the primitive axes from becoming parallel.

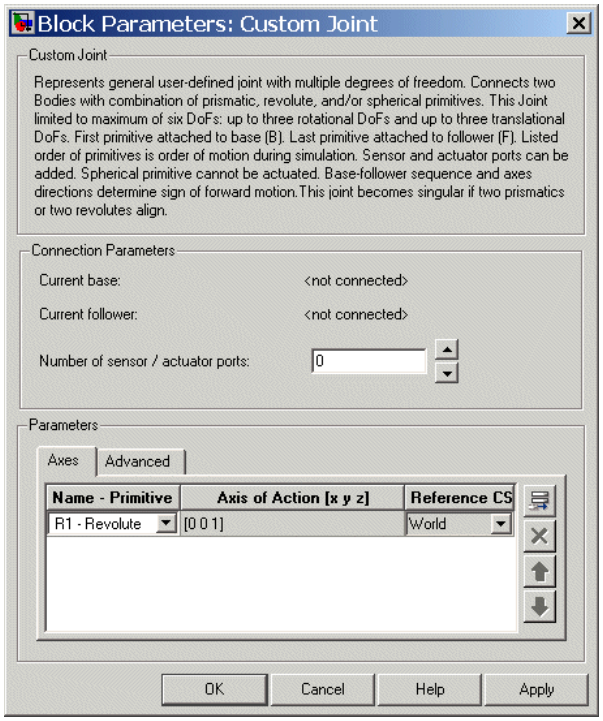
The Custom Joint block's primitives are assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, within assembly tolerances..

You can connect Actuator and Sensor blocks to a Custom Joint, with each Actuator and Sensor connecting to an individual primitive. You cannot connect an Actuator to a spherical primitive.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

Any Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axes.

Dialog Box and Parameters



Custom Joint

Represents general user-defined joint with multiple degrees of freedom. Connects two Bodies with combination of prismatic, revolute, and/or spherical primitives. This Joint limited to maximum of six DoFs: up to three rotational DoFs and up to three translational DoFs. First primitive attached to base (B). Last primitive attached to follower (F). Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Spherical primitive cannot be actuated. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if two prismatics or two revolutes align.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name - Primitive	Axis of Action [x y z]	Reference CS
R1 - Revolute	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion:

- Positive translation is the follower moving in the direction of the translation axis.
- Positive rotation is the follower rotating in the right-hand rule about the rotation axis.
- Positive rotation is the follower rotating in the right-hand rule as shown by the motion figure in the Spherical block reference page.

Current base

When you connect the base (B) connector port on the Custom Joint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Custom Joint Base and Follower Body Connector Ports on page 11-56.

The base Body is automatically connected to the first joint primitive in the primitive list in **Parameters**.

Current follower

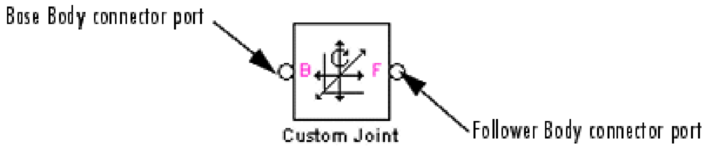
When you connect the follower (F) connector port on the Custom Joint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Custom Joint Base and Follower Body Connector Ports on page 11-56.

The follower Body is automatically connected to the last joint primitive in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`. A spherical primitive cannot be connected to an Actuator.

The motion of a prismatic primitive is specified in linear units. The motion of a revolute primitive is specified in angular units. The motion of a spherical primitive is three DoFs specified in quaternion form.

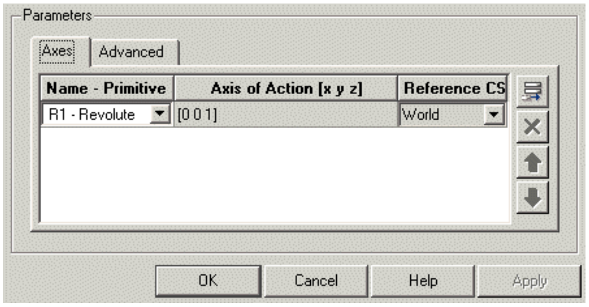


Custom Joint Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Custom Joint has an entry line. These lines specify the direction of the axes of action of the DoFs that the Custom Joint represents.



Name - Primitive	Axis of Action [x y z]	Reference CS
R1 - Revolute	[0 0 1]	World

Name - Primitive

In the pull-down menu, select a label and primitive type for this DoF. Up to three prismatic primitives `P1`, `P2`, `P3` are allowed. The rotational DoFs are represented by up to three revolute primitives: `R1`, `R2`, `R3`. A spherical primitive `S` can take all three allowed rotational DoFs instead.

The default value is `R1 - Revolute`.

Axis of Action [x y z]

Enter here as a three-component vector the directional axis defining the allowed motion of this primitive and its corresponding DoF:

- Prismatic: axis of translation
- Revolute: axis of rotation
- Spherical: field is not active

The default vector is `[0 0 1]`. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes and no two of the prismatic axes can be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.

The field is not active for a spherical primitive.

Managing the Joint Primitives List in a Custom Joint

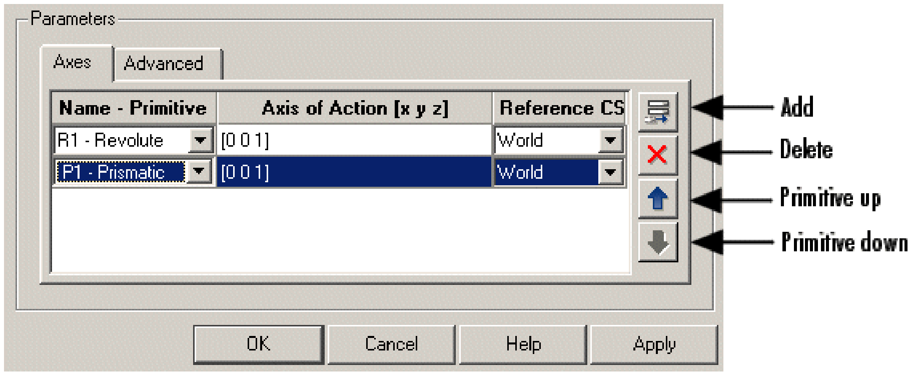
The Custom Joint primitives list controls (see the following figure, Custom Joint Primitives List Controls on page 11-59) allow you to add, reorder, and delete joint primitives in a Custom Joint block:

- To add a joint primitive to the primitives list:
  - Highlight an existing primitive name in the list.
  - Click on the **Add** button (see the following figure, Custom Joint Primitives List Controls).

    A new primitive will appear immediately below the primitive you highlighted.
- To change the position of a joint primitive in the list:
  - Highlight the primitive whose position you want to change.
  - Click on the **Up** or **Down** button (see the following figure, Custom Joint Primitives List Controls) until the primitive is where you want it.
- To delete a joint primitive from the list:
  - Highlight the primitive you want to delete.
  - Click on the **Delete** button (see the following figure, Custom Joint Primitives List Controls).

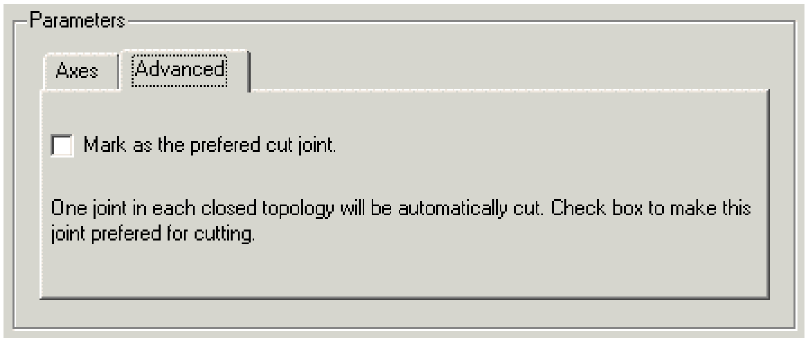
The primitive you highlighted disappears.

- Custom Joint requires at least one primitive, which you cannot delete.



Name - Primitive	Axis of Action [x y z]	Reference CS
R1 - Revolute	[0 0 1]	World
P1 - Prismatic	[0 0 1]	World

Custom Joint Primitives List Controls



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint
In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Bushing, Gimbal, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Prismatic, Revolute, Six-DoF, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Purpose

Represent composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes

Library

Joints

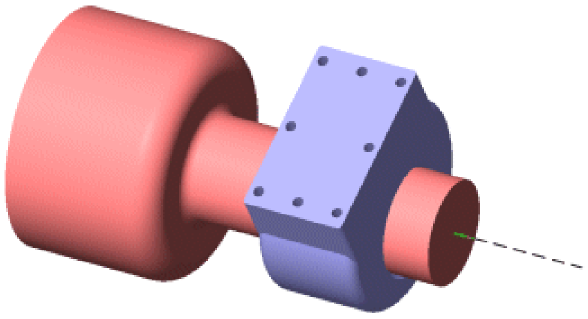
Description

The Cylindrical block represents a composite joint with one translational degrees of freedom (DoF) as one prismatic primitive and one rotational DoF as one revolute primitive. The translation and rotation axes must be parallel.

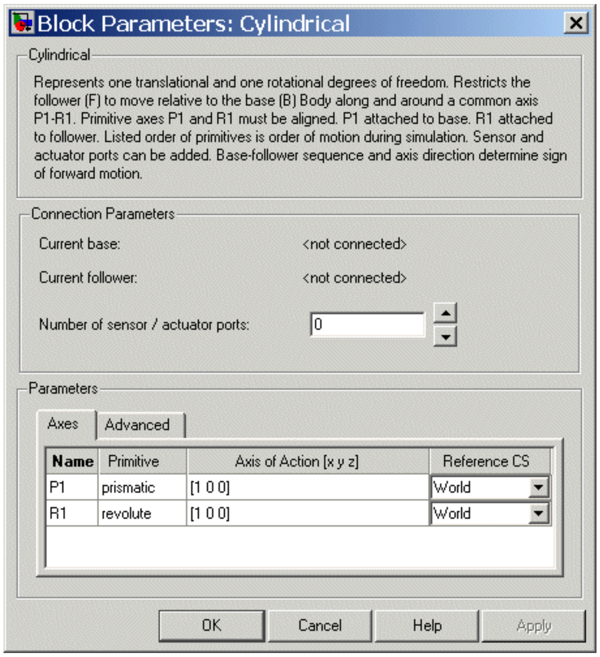
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Cylindrical block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Block Parameters: Cylindrical

Cylindrical

Represents one translational and one rotational degrees of freedom. Restricts the follower (F) to move relative to the base (B) Body along and around a common axis P1-R1. Primitive axes P1 and R1 must be aligned. P1 attached to base. R1 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[1 0 0]	World
R1	revolute	[1 0 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Cylindrical Base and Follower Body Connector Ports.

The base Body is automatically connected to the first joint primitive `P1` in the primitive list in **Parameters**.

Current follower

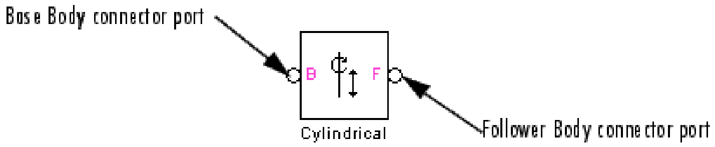
When you connect the follower (F) connector port on the Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Cylindrical Base and Follower Body Connector Ports.

The follower Body is automatically connected to the last joint primitive `R1` in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.



Cylindrical Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Cylindrical has an entry line. These lines specify the direction of the axes of action of the DoFs that the Cylindrical represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Cylindrical block: prismatic revolute `P1` and revolute primitive `R1`.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

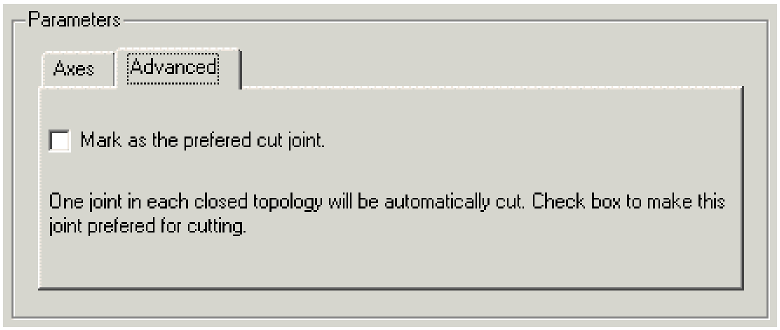
- Prismatic: axis of translation
- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axes are directed vectors whose overall sign matters.

The two axes `P1` and `R1` in Cylindrical must be aligned.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



[ ] Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Disassembled Cylindrical, Prismatic, Revolute, Screw

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Disassembled Cylindrical

Purpose

Represent disassembled cylindrical joint, with one translational DoF and one rotational DoF along and about misaligned axes, with no constraints

Library

Joints/Disassembled Joints

Description

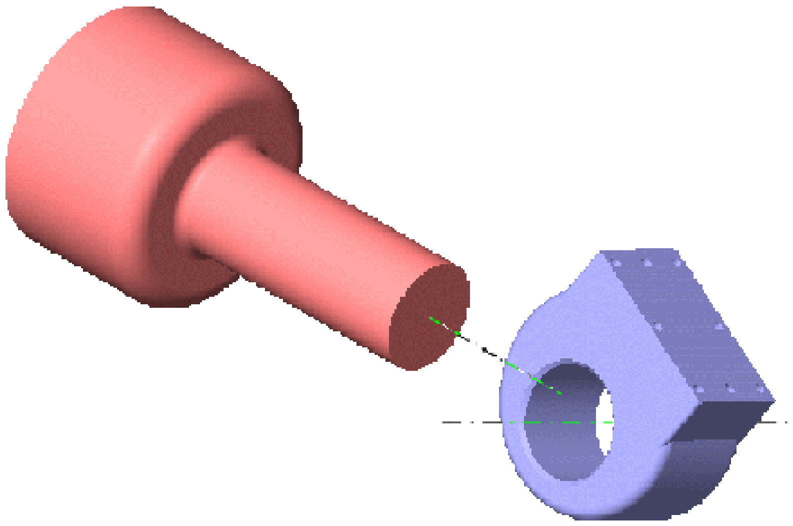
The Disassembled Cylindrical block represents a composite joint, one translational and one rotational degrees of freedom (DoF) along and about a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the two axes at simulation start when it defines a machine's assembled.

A cylindrical joint is composite, with two DoFs and a single specified axis: a prismatic primitive translating along the axis, and a revolute primitive rotating about the axis. There are no constraints between the two primitives.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not need to be spatially collocated points or lie along a joint axis.

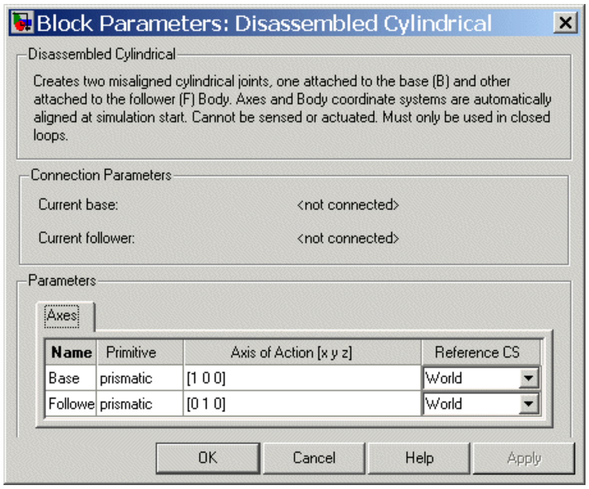
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled translation-rotation joint axes are associated with the base and follower Bodies, respectively.

You can only use a Disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



Disassembled Cylindrical Axes of Follower (blue) and Base (red)

Dialog Box and Parameters



Block Parameters: Disassembled Cylindrical

Disassembled Cylindrical

Creates two misaligned cylindrical joints, one attached to the base (B) and other attached to the follower (F) Body. Axes and Body coordinate systems are automatically aligned at simulation start. Cannot be sensed or actuated. Must only be used in closed loops.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

Parameters

Axes

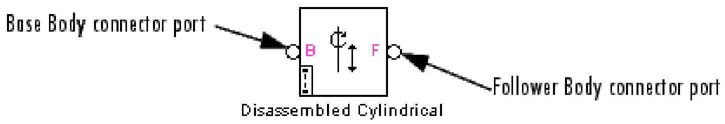
Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

Current base
: When you connect the base (B) connector port on the Disassembled Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Cylindrical Base and Follower Body Connector Ports on page 11-68.

Current follower
: When you connect the follower (F) connector port on the Disassembled Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Cylindrical Base and Follower Body Connector Ports on page 11-68.

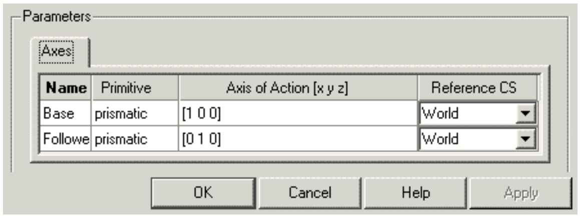


Disassembled Cylindrical Base and Follower Body Connector Ports

Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the translational-rotational DoFs that the Disassembled Cylindrical represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

Name

This column automatically displays the names of the two misaligned rotation axes attached to base and follower bodies, respectively.

Axis of Action [x y z]

Enter here as two three-component vectors the two misaligned directional axes along and about which the base and follower bodies respectively can translate and rotate. The default vectors are `[1 0 0]` and `[0 1 0]`, respectively. The axes are directed vectors whose overall signs matter.

Reference CS

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of translation-rotation are oriented with respect to. The defaults are `World`.

See Also

Cylindrical, Disassembled Prismatic, Disassembled Revolute, Disassembled Spherical

See "Modeling Joints" on page 4-20 for more on representing DoFs with Disassembled Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting disassembled joints.

Purpose

Represent disassembled prismatic joint with one translational degree of freedom along misaligned axes

Library

Joints/Disassembled Joints

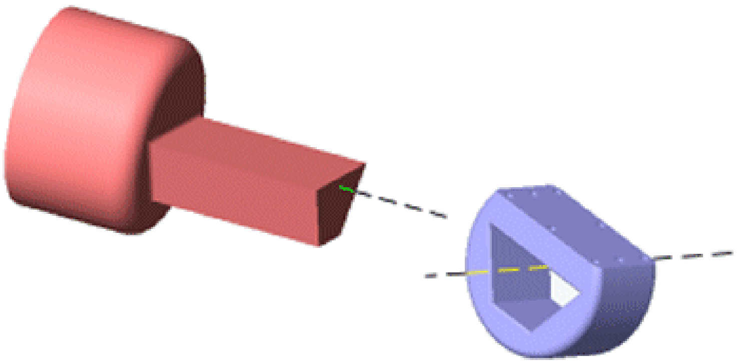
Description

The Disassembled Prismatic block represents a single translational degrees of freedom (DoF) along a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the translation axes at simulation start when it defines a machine's assembled configuration.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not have to lie along a joint axis. As with the Prismatic Joint, these Body CS origins do not need to be spatially collocated points either.

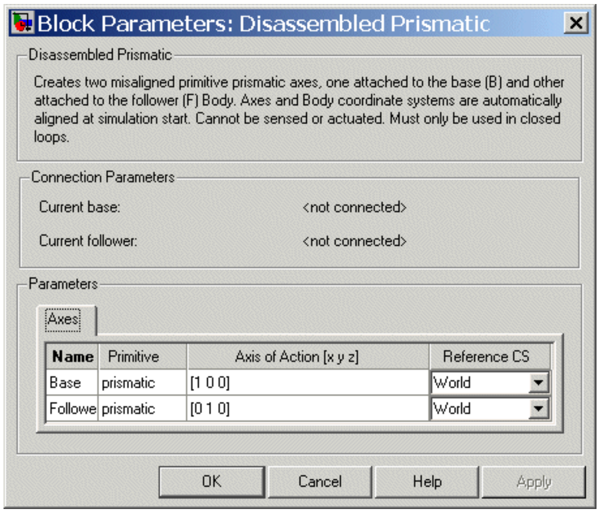
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint axes are associated with the base and follower Bodies, respectively.

You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



Disassembled Prismatic Axes of Follower (blue) and Base (red)

Dialog Box and Parameters



Disassembled Prismatic

Creates two misaligned primitive prismatic axes, one attached to the base (B) and other attached to the follower (F) Body. Axes and Body coordinate systems are automatically aligned at simulation start. Cannot be sensed or actuated. Must only be used in closed loops.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

Parameters

Axes

Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

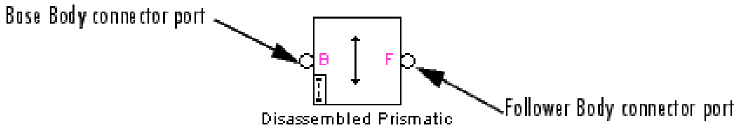
Connection Parameters

Current base

When you connect the base (B) connector port on the Disassembled Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Prismatic Base and Follower Body Connector Ports on page 11-72.

Current follower

When you connect the follower (F) connector port on the Disassembled Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Prismatic Base and Follower Body Connector Ports on page 11-72.

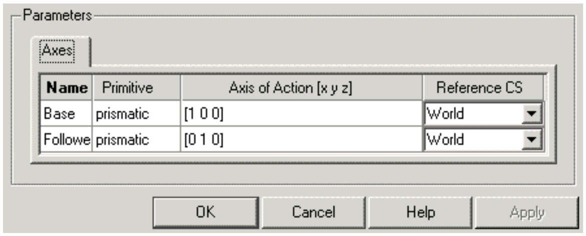


Disassembled Prismatic Base and Follower Body Connector Ports

Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the translational DoF that the Disassembled Prismatic represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

Name

This column automatically displays the names of the two misaligned translation axes attached to base and follower bodies, respectively.

Axis of translation [x y z]

Enter here as two three-component vectors the two misaligned directional axes along which the base and follower bodies respectively can translate. The default vectors are `[1 0 0]` and `[0 1 0]`, respectively. The axes are directed vectors whose overall signs matter.

Reference CS

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of translation are oriented with respect to. The defaults are `World`.

See Also

Disassembled Cylindrical, Disassembled Revolute, Disassembled Spherical, Prismatic

See "Modeling Joints" on page 4-20 for more on representing DoFs with Disassembled Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closing loops with disassembled joints.

Disassembled Revolute

Purpose Represent disassembled revolute joint with one rotational degree of freedom about misaligned axes

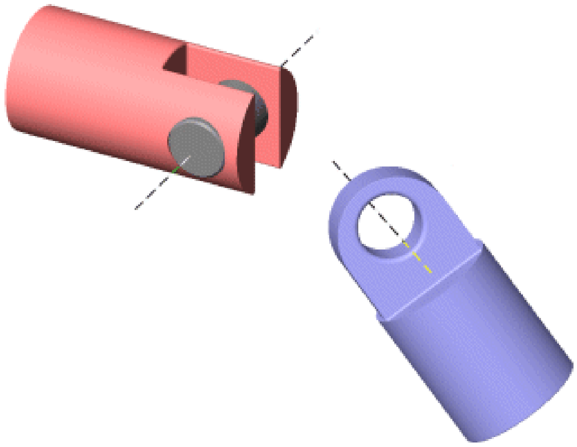
Library Joints/Disassembled Joints

Description The Disassembled Revolute block represents a single rotational degrees of freedom (DoF) along a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the rotation axes at simulation start when it defines a machine's assembled.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not need to be spatially collocated points or lie along a joint axis.

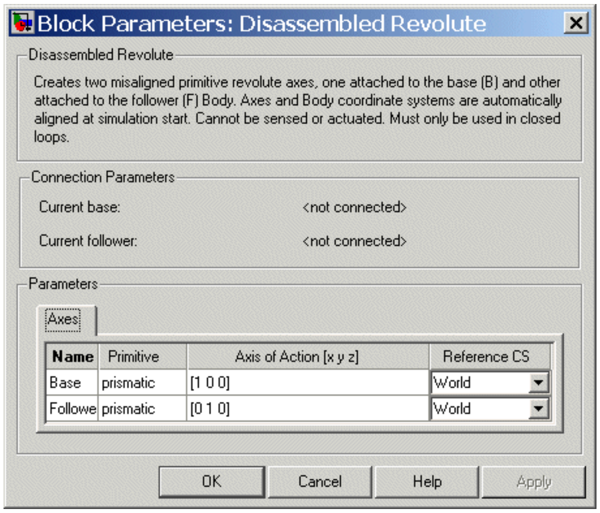
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint axes are associated with the base and follower Bodies, respectively.

You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



Disassembled Revolute Axes of Follower (blue) and Base (red)

Dialog Box and Parameters



Block Parameters: Disassembled Revolute

Disassembled Revolute

Creates two misaligned primitive revolute axes, one attached to the base (B) and other attached to the follower (F) Body. Axes and Body coordinate systems are automatically aligned at simulation start. Cannot be sensed or actuated. Must only be used in closed loops.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

Parameters

Axes

Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

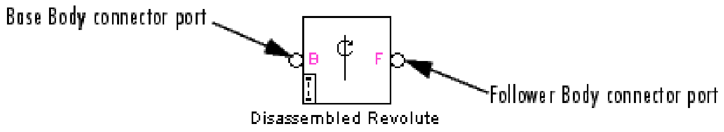
Connection Parameters

Current base

When you connect the base (B) connector port on the Disassembled Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Revolute Base and Follower Body Connector Ports on page 11-76.

Current follower

When you connect the follower (F) connector port on the Disassembled Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Revolute Base and Follower Body Connector Ports on page 11-76.

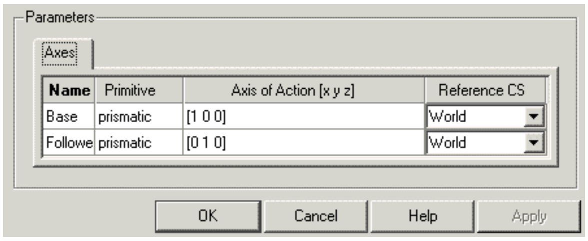


Disassembled Revolute Base and Follower Body Connector Ports

Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the rotational DoF that the Disassembled Revolute represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
Base	prismatic	[1 0 0]	World
Followe	prismatic	[0 1 0]	World

Name

This column automatically displays the names of the two misaligned rotation axes attached to base and follower bodies, respectively.

Axis of rotation [x y z]

Enter here as two three-component vectors the two misaligned directional axes about which the base and follower bodies respectively can rotate. The default vectors are `[1 0 0]` and `[0 1 0]`, respectively. The axes are directed vectors whose overall signs matter.

Reference CS

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of rotation are oriented with respect to. The defaults are `World`.

See Also

Disassembled Cylindrical, Disassembled Prismatic, Disassembled Spherical, Revolute

See “Modeling Joints” on page 4-20 for more on representing DoFs with Disassembled Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closing loops with disassembled joints.

Disassembled Spherical

Purpose

Represent disassembled spherical joint with three rotational degrees of freedom about dislocated pivots

Library

Joints/Disassembled Joints

Description

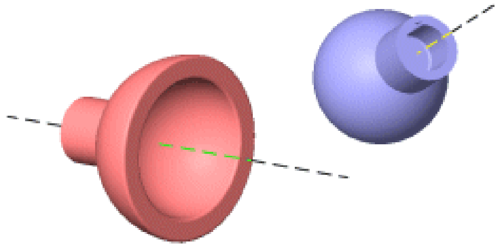
The Disassembled Spherical block represents three rotational degrees of freedom (DoF) about a pair of specified dislocated pivots at the two bodies, separated "ball-in-socket" joints. SimMechanics automatically assembles (collocates) the spherical pivots at simulation start when it defines a machine's assembled configuration.

Two rotational DoFs specify a directional axis, and a third rotational DoF specifies rotation about that directional axis. (See the motion figure in the Spherical block reference page.) The sense of each rotational DoF is defined by the right-hand rule. Unlike the Gimbal block, the Disassembled Spherical block cannot become singular.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs (the dislocated pivots) do not need to be spatially collocated points.

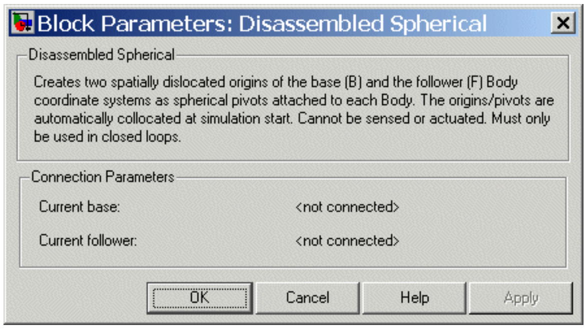
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint pivots are associated with the base and follower Bodies, respectively.

You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



Disassembled Spherical Pivots of Follower (blue) and Base (red)

Dialog Box and Parameters



Block Parameters: Disassembled Spherical

Disassembled Spherical

Creates two spatially dislocated origins of the base (B) and the follower (F) Body coordinate systems as spherical pivots attached to each Body. The origins/pivots are automatically collocated at simulation start. Cannot be sensed or actuated. Must only be used in closed loops.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

The dialog has one area, **Connection parameters**, which is inactive.

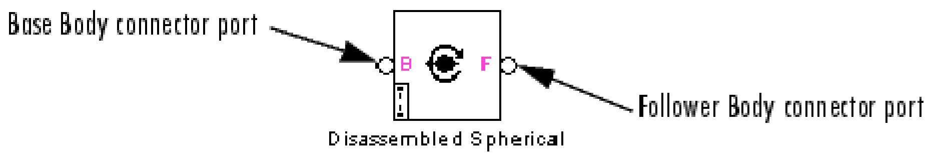
Connection Parameters

Current base

When you connect the base (B) connector port on the Disassembled Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Spherical Base and Follower Body Connector Ports on page 11-80.

Current follower

When you connect the base (F) connector port on the Disassembled Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Disassembled Spherical Base and Follower Body Connector Ports on page 11-80.



Disassembled Spherical Base and Follower Body Connector Ports

See Also

Disassembled Cylindrical, Disassembled Prismatic, Disassembled Revolute, Gimbal, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Disassembled Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closing loops with disassembled joints.

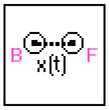
Purpose

Specify distance between two Body CS origins as function of time

Library

Constraints & Drivers

Description



The Distance Driver block drives the distance between the origins of two Body coordinate system (CS) as a function of time that you specify. This function must always remain nonnegative during the simulation.

Let $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ be the vector positions of the origins of CS1 on one Body and CS2 on the other Body, respectively. These vectors can be measured in any CS. The Distance Driver specifies the scalar distance $d = |\boldsymbol{r}_1 - \boldsymbol{r}_2|$ between these points as a function of time:

$$|\boldsymbol{r}_1 - \boldsymbol{r}_2| = d(t=0) + f(t)$$

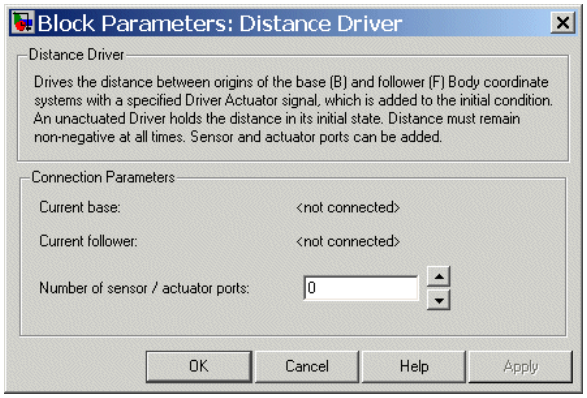
You connect the Distance Driver to a Driver Actuator block.

The Simulink input signal into the Driver Actuator specifies the time-dependent driving function *f(t)* and its first two derivatives, as well as their units. If you do not actuate Distance Driver, this block acts as a time-independent constraint that freezes the distance between the two Body CSs at its initial value $d(t=0)$ during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver and measure the reaction forces/torques between the driven bodies.

Dialog Box and Parameters


Block Parameters: Distance Driver

Distance Driver

Drives the distance between origins of the base (B) and follower (F) Body coordinate systems with a specified Driver Actuator signal, which is added to the initial condition. An unactuated Driver holds the distance in its initial state. Distance must remain non-negative at all times. Sensor and actuator ports can be added.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

The dialog has one active area, **Connection parameters**.

Connection Parameters

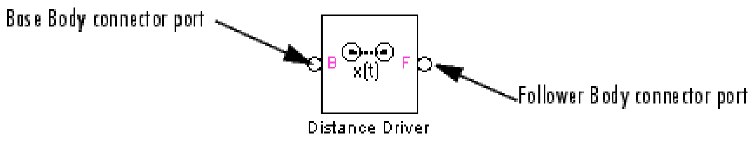
The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

Current base
: When you connect the base (B) connector port on the Distance Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Distance Driver Base and Follower Body Connector Ports on page 11-83.

Current follower
: When you connect the follower (F) connector port on the Distance Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Distance Driver Base and Follower Body Connector Ports on page 11-83.

Number of sensor/actuator ports
: Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is `0`.

To activate the Driver, connect a Driver Actuator.



Distance Driver Base and Follower Body Connector Ports

See Also

Constraint & Driver Sensor, Driver Actuator, Linear Driver, Weld

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Drivers.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on using drivers in closed loops.

See “Constraints and Drivers” on page 10-5.

Driver Actuator

Purpose

Apply relative motion between a pair of constrained bodies through driver

Library

Sensors & Actuators

Description

The Driver Actuator block actuates a Driver block connected between a pair of bodies. You connect the block to the Driver block connected between the Bodies. The Driver block represents a time-dependent (rheonomic) constraint on a relative degrees of freedom (DoF) between the two bodies. A Driver requires a time-dependent function to specify the relative position, velocity, and acceleration of the connected Bodies. The output of the Driver Actuator is this time-dependent function *f(t)* and its first two derivatives.

You specify these three functions as a bundled Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block, satisfying these conditions:

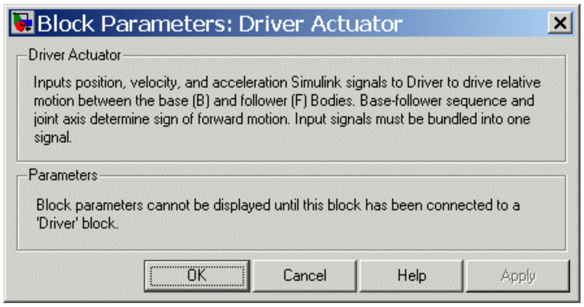
- The signal must consist of three bundled components.
  - *Exception:* The Velocity Driver requires two bundled components.
- The three components must be ordered [$f(t)$, $df(t)/dt$, $d^2f(t)/dt^2$].
  - *Exception:* The Velocity Driver requires a function and its derivative [$f(t)$, $df(t)/dt$].
- Each successive signal component must be the time derivative of the previous component.
- The specific meaning of *f(t)* depends on the connected Driver block being actuated. Select the specific Driver block for details.

Linear Motions	**Angular Motions**
Distance Driver	Angle Driver
Linear Driver	
Velocity Driver	

The Driver connects a base (B) and a follower (F) Body. The base-follower sequence determines the sense of the actuation signal. The inport is the Simulink input signal. The output is the follower you connect to the Driver block you want to actuate.

Note You do not have to connect a Driver Actuator to a Driver block. If you do not actuate a Driver, the Driver block acts as a time-independent constraint that freezes the driven relative DoF between the Bodies at its initial value during the simulation.

Dialog Box and Parameters



Block Parameters: Driver Actuator

Driver Actuator

Inputs position, velocity, and acceleration Simulink signals to Driver to drive relative motion between the base (B) and follower (F) Bodies. Base-follower sequence and joint axis determine sign of forward motion. Input signals must be bundled into one signal.

Parameters

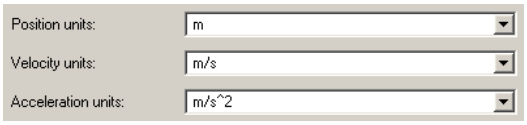
Block parameters cannot be displayed until this block has been connected to a 'Driver' block.

The dialog has one active area, **Actuation**. The block parameters are not displayed unless you connect it to a specific Driver block.

Actuation

The block dialog parameters depend on the specific Driver block to which you have connected it.

Driving Linear Motion



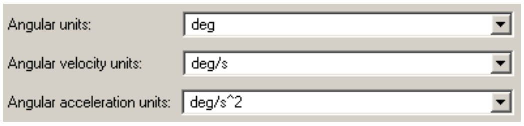
Position units:	m
Velocity units:	m/s
Acceleration units:	m/s^2

Position units
In the pull-down menu, choose the units of the actuating *f(t)* you apply to the relative motion of the bodies. The default is `m` (meters).

Velocity units
In the pull-down menu, choose the units of the actuating *df(t)/dt* you apply to the relative motion of the bodies. The default is `m/s` (meters/second).

Acceleration units
In the pull-down menu, choose the units of the actuating $d^2f(t)/dt^2$ you apply to the relative motion of the bodies. The default is `m/s`^{2} (meters/second^2).

Driving Angular Motion



Angular units: deg
Angular velocity units: deg/s
Angular acceleration units: deg/s^2

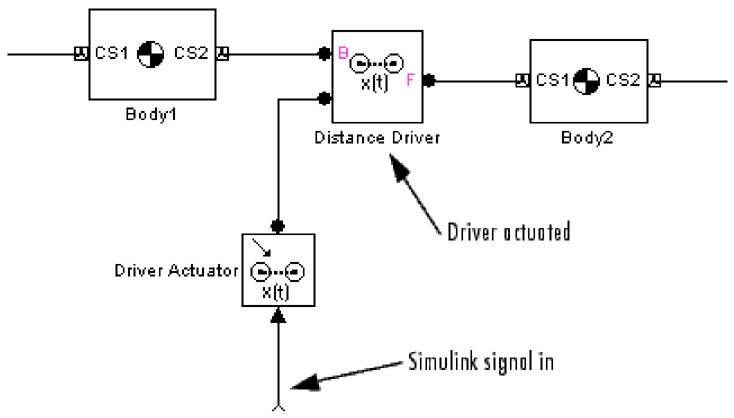
Angular units
In the pull-down menu, choose the units of the actuating *f(t)* you apply to the relative motion of the bodies. The default is `deg` (degrees).

Angular velocity units
In the pull-down menu, choose the units of the actuating *df(t)/dt* you apply to the relative motion of the bodies. The default is `deg/s` (degrees/second).

Angular acceleration units
In the pull-down menu, choose the units of the actuating $d^2f(t)/dt^2$ you apply to the relative motion of the bodies. The default is `deg/s`^{2} (degrees/second^2).

Example

Here is a Driver Actuator connected to a Distance Driver, which connects two Bodies:



You must add an Actuator port (connector port) to the Driver block to connect the Driver Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Driver determines the sense of the Driver Actuator data.

The Driver Actuator drives the relative motion between the two Bodies connected to the Driver. The nature of the connected Driver block determines the exact meaning of the actuation data, including the choice of units.

See Also

Body Actuator, Constraint & Driver Sensor, Joint Actuator, Mechanical Branching Bar

See “Constraints and Drivers” on page 10-5.

In Simulink, see the Signal Routing Library and the Sources Library.

Gear Constraint

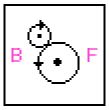
Purpose

Constrain rotational motion of two bodies to move along tangent pitch circles

Library

Constraints & Drivers

Description



The two Bodies connected by a Gear Constraint block are each restricted to turn relative to another along pitch circles centered at each body. The pitch circle centers are the origins of the Body coordinate systems (CSs) at which the Gear Constraint block is connected on either side. The pitch circles are tangent at one contact point.

Let $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ be the radius vectors of the two pitch circles and $\boldsymbol{\omega}_1$, $\boldsymbol{\omega}_2$ the angular velocity vectors of the two bodies. The Gear Constraint requires that:

$$\omega_1 \times \boldsymbol{r}_1 = \omega_2 \times \boldsymbol{r}_2$$

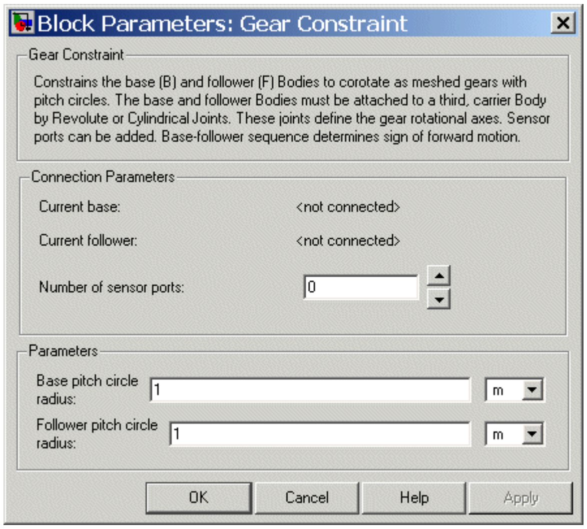
You specify the scalar radii r_1, r_2 of the pitch circles.

You must also connect the two Bodies connected by a Gear Constraint to a third, carrier Body by Revolute or Cylindrical Joints. (The third carrier body can be ground, but you must use two Ground blocks in this case, because a Ground has only one Body CS port. Both Grounds represent the same immobile body.) The constrained pair of Bodies rotate relative to one another about distinct rotational axes defined by the angular velocity vectors ω_1, ω_2. These axes do not have to be parallel.

Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

You can connect a Constraint & Driver Sensor to a Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

Dialog Box and Parameters


Block Parameters: Gear Constraint

Gear Constraint

Constrains the base (B) and follower (F) Bodies to corotate as meshed gears with pitch circles. The base and follower Bodies must be attached to a third, carrier Body by Revolute or Cylindrical Joints. These joints define the gear rotational axes. Sensor ports can be added. Base-follower sequence determines sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor ports: 0

Parameters

Base pitch circle radius: 1 m

Follower pitch circle radius: 1 m

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.

Current base

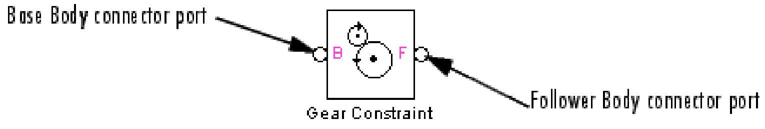
When you connect the base (B) connector port on the Gear Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Gear Constraint Base and Follower Body Connector Ports on page 11-90.

Current follower

When you connect the follower (F) connector port on the Gear Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Gear Constraint Base and Follower Body Connector Ports on page 11-90.

Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is 0.



Gear Constraint Base and Follower Body Connector Ports

Parameters

Base pitch circle radius

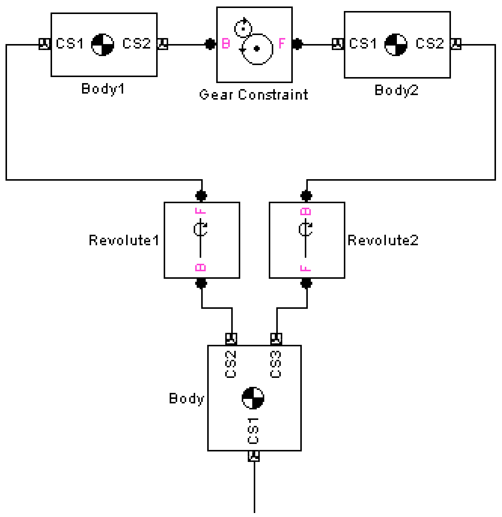
Enter a radius for the pitch circle centered at base Body CS. In the pull-down menu to the right, select units. The defaults are `1` and `m` (meters), respectively.

Follower pitch circle radius

Enter a radius for the pitch circle centered at follower Body CS. In the pull-down menu to the right, select units. The defaults are `1` and `m` (meters), respectively.

Example

A simple example of a valid part of a model with a Gear Constraint:



The Body CS origins CS2@Body1 and CS1@Body2 must be separated and oriented in such a way that the gear pitch circles are in contact and tangent at one point.

See Also

Body, Constraint & Driver Sensor, Cylindrical, Ground, Revolute

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Constraints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on using constraints in closed loops.

See “Constraints and Drivers” on page 10-5.

Gimbal

Purpose

Represent composite joint with three rotational DoFs

Library

Joints

Description

The Gimbal block represents a composite joint with three rotational degrees of freedom (DoFs) as three revolute primitives. There are no constraints among the primitives.

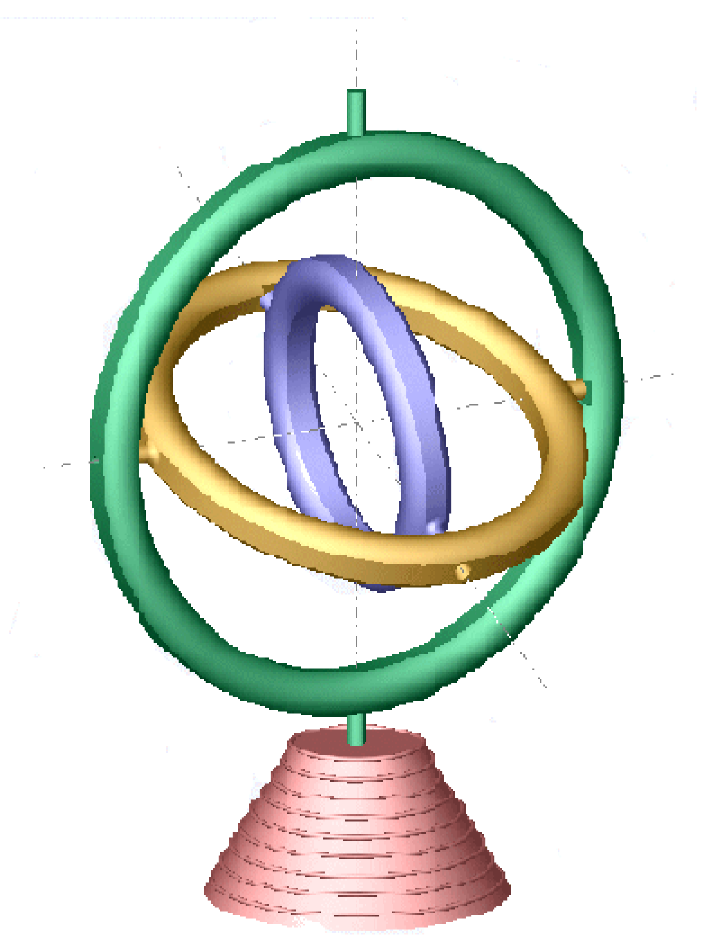
Caution A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). The simulation stops with an error in this case.

A joint with three revolute primitives must be configured in the initial state with the three revolute primitive axes mutually orthogonal. There are no restrictions on the primitive axes once the simulation starts, except to prevent any two of the primitive axes from becoming parallel.

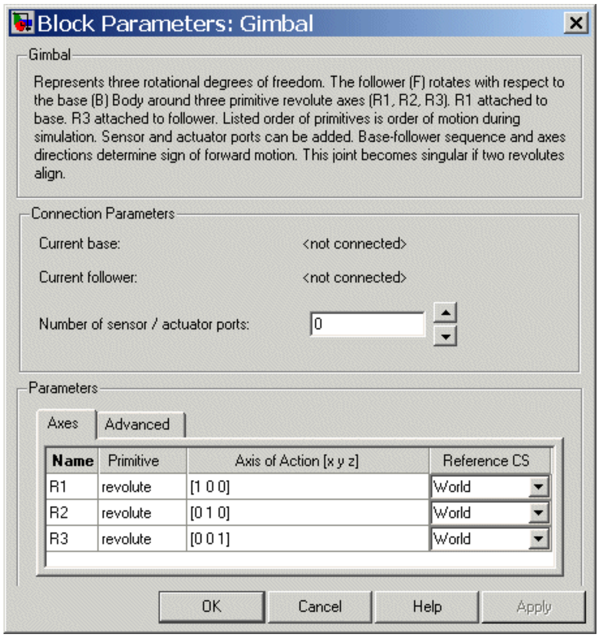
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Gimbal block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Block Parameters: Gimbal

Gimbal

Represents three rotational degrees of freedom. The follower (F) rotates with respect to the base (B) Body around three primitive revolute axes (R1, R2, R3). R1 attached to base. R3 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if two revolutes align.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[1 0 0]	World
R2	revolute	[0 1 0]	World
R3	revolute	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Gimbal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Gimbal Base and Follower Body Connector Ports on page 11-95.

The base Body is automatically connected to the first joint primitive `R1` in the primitive list in **Parameters**.

Current follower

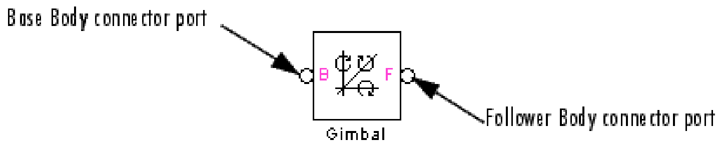
When you connect the follower (F) connector port on the Gimbal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Gimbal Base and Follower Body Connector Ports on page 11-95.

The follower Body is automatically connected to the last joint primitive R3 in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of revolute primitives is specified in angular units.



Gimbal Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Gimbal has an entry line. These lines specify the direction of the axes of action of the DoFs that the Gimbal represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Gimbal block: revolute primitives R1, R2, R3.

Axis of Action [x y z]

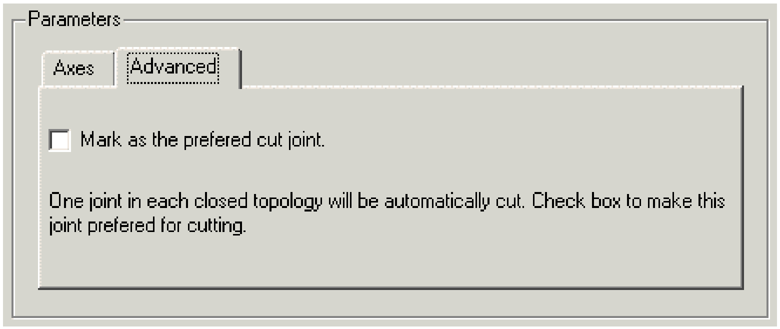
Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes can be parallel.

Reference CS
Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint
In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Revolute, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Ground

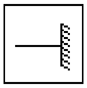
Purpose

Represent immobile point at rest in World

Library

Bodies

Description



A Ground block represents an immobile ground point at rest in the absolute inertial World reference frame. Connecting it to a Joint prevents one side of that Joint from moving. You can also connect a Ground block to a Machine Environment block.

Note Every valid SimMechanics machine *must* have at least one Ground block.

You *must* connect exactly one Ground block in each machine of your model to a Machine Environment block.

Ground is a type of Body, but you can connect only one side of a Ground to a Joint block. A Ground block automatically carries a grounded coordinate system (CS). This Grounded CS is inertial, at rest in the World reference frame, with coordinate axes parallel to the World axes:

$+x$ points right

$+y$ points up (gravity in $-y$ direction)

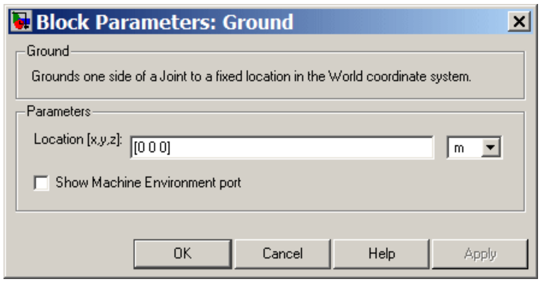
$+z$ points out of the screen, in three dimensions

But a Ground's origin is the ground point, which in general is shifted with respect to the World origin.

Multiple Ground blocks represent different fixed points in the global inertial World. In the topology of a machine model, multiple Ground blocks function as a single body.

You cannot connect a Sensor or Actuator to a Ground block, because the ground point cannot be moved.

Dialog Box and Parameters



Block Parameters: Ground

Ground

Grounds one side of a Joint to a fixed location in the World coordinate system.

Parameters

Location [x,y,z]: [0 0 0] m

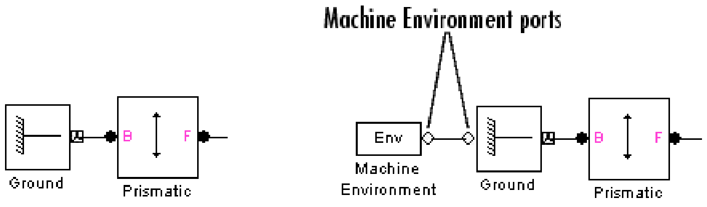
☐ Show Machine Environment port

Location [x,y,z]

Enter the position of the ground point translated from the origin of the World CS. The position is specified as a translation vector *(x,y,z)*, with components projected onto the fixed World CS axes. Set the Ground position units using the pull-down menu to the right. The defaults are `[0 0 0]` and `m` (meters).

Show Machine Environment port

Select to enable the Machine Environment port on the Ground block. This port allows you to connect a Machine Environment block to the Ground and the machine that the Ground is a part of. The default is not selected.



A Ground Without and With a Connected Machine Environment Block

Configuring the Mechanical Environment

If you connect a Machine Environment block to a Ground, you can adjust the mechanical environment for the machine of which that Ground is a part. Consult the Machine Environment block reference.

See Also

Body, Machine Environment, Shared Environment

See “Modeling Machines” on page 4-3, “Modeling Bodies and Grounds” on page 4-10, and “Modeling Joints” on page 4-20 for more on creating valid SimMechanics models and setting up Grounds.

See the relevant entries in the Glossary: **ground**, **grounded CS**, **machine**, and **World**.

Purpose

Represent composite joint with two translational DoFs

Library

Joints

Description

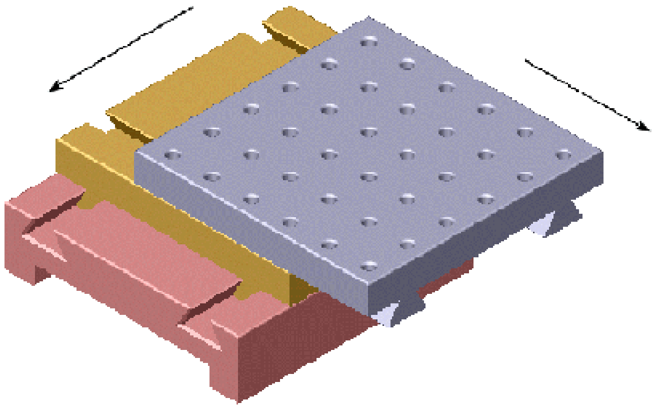
The In-Plane block represents a composite joint with two translational degrees of freedom (DoFs) as two prismatic primitives. There are no constraints among the primitives.

Caution A joint with two prismatic primitives becomes singular if the two translation axes become parallel. The simulation stops with an error in this case.

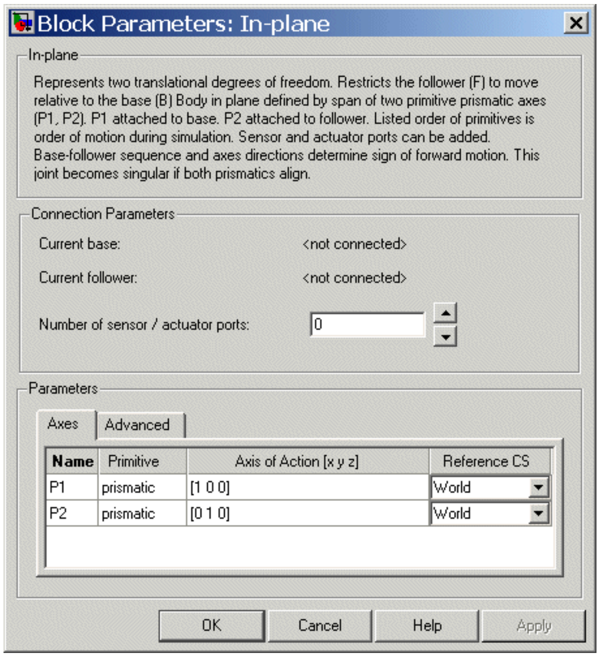
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The In-Plane block is assembled: the origins of these Body CSs must lie along the primitive axes, within assembly tolerancesBut the Body CS origins on either side of the Joint do not have to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Block Parameters: In-plane

In-plane

Represents two translational degrees of freedom. Restricts the follower (F) to move relative to the base (B) Body in plane defined by span of two primitive prismatic axes (P1, P2). P1 attached to base. P2 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if both prismatics align.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[1 0 0]	World
P2	prismatic	[0 1 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

Current base
: When you connect the base (B) connector port on the In-Plane block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, In-Plane Base and Follower Body Connector Ports on page 11-103.

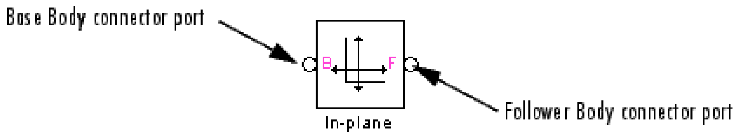
  The base Body is automatically connected to the first joint primitive `P1` in the primitive list in **Parameters**.

Current follower
: When you connect the follower (F) connector port on the In-Plane block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, In-Plane Base and Follower Body Connector Ports on page 11-103.

  The follower Body is automatically connected to the last joint primitive `P2` in the primitive list in **Parameters**.

Number of sensor/actuator ports
: Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

  The motion of prismatic primitives is specified in linear units.



In-Plane Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in In-Plane has an entry line. These lines specify the direction of the axes of action of the DoFs that the In-Plane represents.

Name - Primitive
: The primitive list states the names and types of joint primitives that make up the In-Plane block: prismatic primitives `P1`, `P2`.

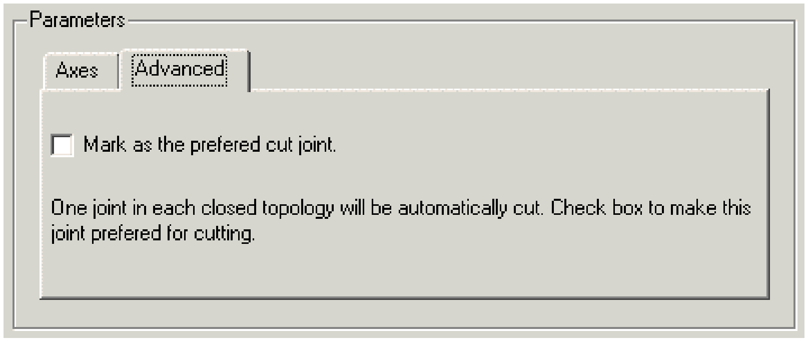
Axis of Action [x y z]
: Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

  - Prismatic: axis of translation

  The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

  To prevent singularities and simulation errors, the two prismatic axes cannot be parallel.

Reference CS
: Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Planar, Prismatic

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Joint Actuator

Purpose Apply force, torque, or motion to joint primitive

Library Sensors & Actuators

Description A joint between two bodies represents relative degrees of freedom (DoFs) between the bodies. The Joint Actuator block actuates a Joint block connected between two Bodies with one of these signals:

- A generalized force:
  - Force for translational motion along a prismatic joint primitive
  - Torque for rotational motion about a revolute joint primitive
- A motion:
  - Translational motion for a prismatic joint primitive, in terms of linear position, velocity, and acceleration. The velocity signal must be the derivative of the position signal, and the acceleration the derivative of the velocity.
  - Rotational motion for a revolute joint primitive, in terms of angular position, velocity, and acceleration. The angular velocity signal must be the derivative of the angle signal, and the angular acceleration the derivative of the angular velocity.

The generalized force or the motion is a function of time specified by a Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block.

The Joint Actuator applies the actuation signal along/about the joint axis in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog. The Joint connects a base and a follower Body. The base-follower sequence determines the sense of the actuation signal.

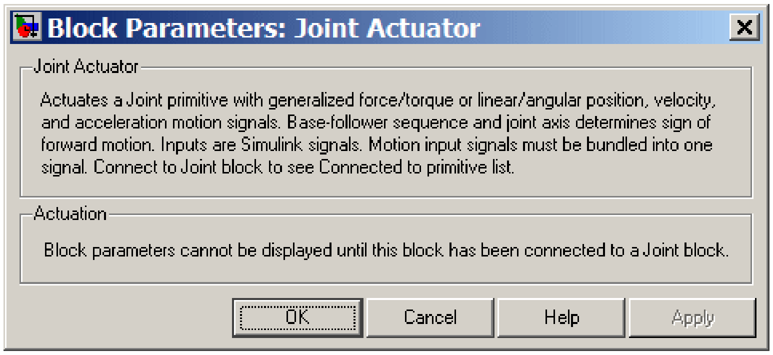
The inport is the Simulink input signal. The output is the connector port you connect to the Joint block you want to actuate. A Joint Actuator block actuates one joint primitive at a time:

- A primitive Joint (Prismatic or Revolute) has only one primitive within the Joint to actuate.
- A composite Joint has multiple joint primitives within, and you must choose which of those primitives to actuate with the Joint Actuator.

You cannot connect a Joint Actuator to a Spherical or spherical primitive.

Caution You cannot simultaneously actuate a joint primitive with Joint Actuator motion actuation and with a Joint Initial Condition Actuator.

Dialog Box and Parameters


Block Parameters: Joint Actuator

Joint Actuator

Actuates a Joint primitive with generalized force/torque or linear/angular position, velocity, and acceleration motion signals. Base-follower sequence and joint axis determines sign of forward motion. Inputs are Simulink signals. Motion input signals must be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Actuation

Block parameters cannot be displayed until this block has been connected to a Joint block.

The dialog has one active area, **Actuation**. The full block parameters are not displayed unless you connect it to a specific Joint block.

Actuation

Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint that you want to actuate with the Joint Actuator. A primitive Joint block has only one joint primitive.

You cannot connect a Joint Actuator to a spherical primitive.

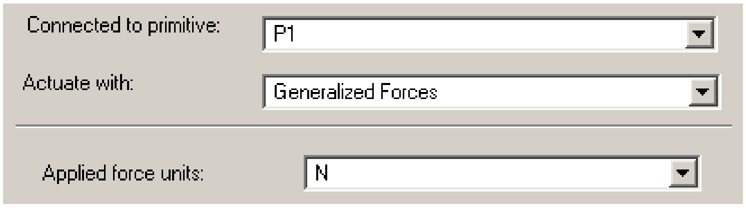
If the Joint Actuator is not connected to a Joint block, this menu is blank.

Actuate with

In the pull-down menu, choose one of two types of actuation, `Generalized Forces` or `Motion`.

Generalized Forces

This option interprets the actuation signal as a force or a torque between the Bodies connected by the Joint block you are actuating. Choose units depending on whether you are actuating a prismatic or revolute primitive.

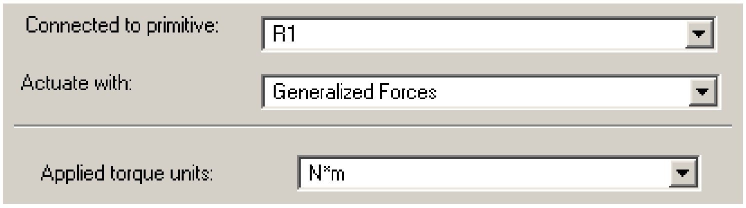


Connected to primitive: P1
Actuate with: Generalized Forces
Applied force units: N

Applied force units

In the pull-down menu, choose units for the actuation force. The default is `N` (newtons).

The Simulink input is a 1-component signal.



Connected to primitive: R1
Actuate with: Generalized Forces
Applied torque units: N*m

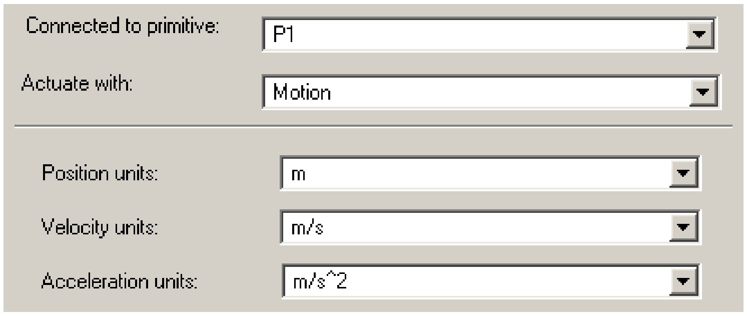
Applied torque units

In the pull-down menu, choose units for the actuation torque. The default is `N*m` (newton-meters).

The Simulink input is a 1-component signal.

Motion

This option interprets the actuation signal as a motion of the joint primitive to which you connect the Joint Actuator. Choose units depending on whether you are actuating a prismatic or revolute primitive. The Simulink input is a bundled 3-component signal with components in the order shown in the dialog.



Connected to primitive: P1

Actuate with: Motion

Position units: m

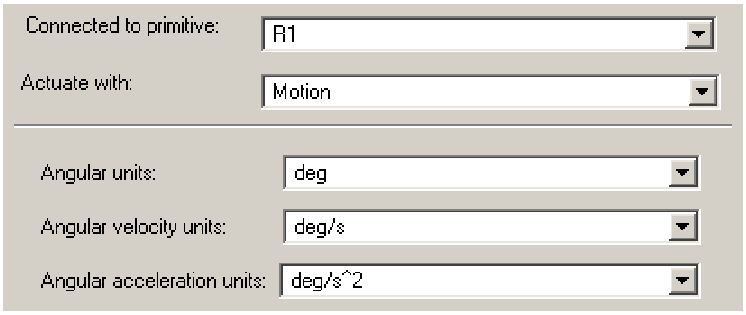
Velocity units: m/s

Acceleration units: m/s^2

Position units
: In the pull-down menu, choose units for the linear position motion actuation. The default is `m` (meters).

Velocity units
: In the pull-down menu, choose units for the linear velocity motion actuation. The default is `m/s` (meters/second).

Acceleration units
: In the pull-down menu, choose units for the linear acceleration motion actuation. The default is `m/s`^{2} (meters/second^2).


Connected to primitive: R1

Actuate with: Motion

Angular units: deg

Angular velocity units: deg/s

Angular acceleration units: deg/s^2

Angular units

In the pull-down menu, choose units for the angular motion actuation. The default is `deg` (degrees).

Angular velocity units

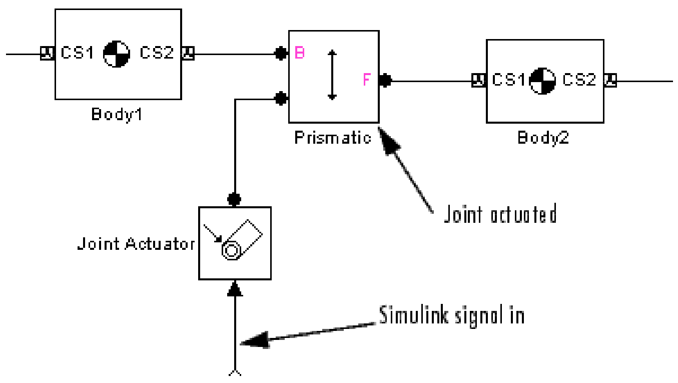
In the pull-down menu, choose units for the angular velocity motion actuation . The default is `deg/s` (degrees/second).

Angular acceleration units

In the pull-down menu, choose units for the angular acceleration motion actuation. The default is `deg/s`^{2} (degrees/second^2).

Example

Here is a Joint Actuator connected to a Prismatic that connects two Bodies:



You must add an Actuator port (connector port) to the Joint block to connect the Joint Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Actuator data.

See Also

Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute

See “Joints” on page 10-2.

In Simulink, see the Signal Routing Library and the Sources Library.

Joint Initial Condition Actuator

Purpose

Apply initial positions and velocities to primitives of Joint before starting simulation

Library

Sensors & Actuators

Description

IC

The Joint Initial Condition Actuator block supplies the prismatic and revolute joint primitives of a Joint block with initial value data. The initial values are the positions and velocities of the joint primitives and fully specify the initial state of motion (initial kinematic state) of those primitives.

You can set initial positions and velocities for two primitive types:

- Translational initial conditions for a prismatic primitive, in terms of linear position and velocity
- Rotational initial conditions for a revolute primitive, in terms of angular position and velocity

This block can actuate one, some, or all of the prismatic and revolute primitives of a Joint.

The Joint Initial Condition Actuator applies the initial state along/about the joint axis in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog. The Joint connects a base and a follower Body. The base-follower sequence determines the sense of the actuation signal.

The output is the connector port you connect to the Joint block whose initial conditions you want to set. You set the initial linear and/or angular positions and velocities in the block's dialog, so there is no input signal.

You cannot actuate a Spherical or spherical primitive with a Joint Initial Condition Actuator.

Caution You cannot simultaneously actuate a joint primitive with a Joint Initial Condition Actuator and with Joint Actuator motion actuation.

Initial Geometric Versus Kinematic Configuration

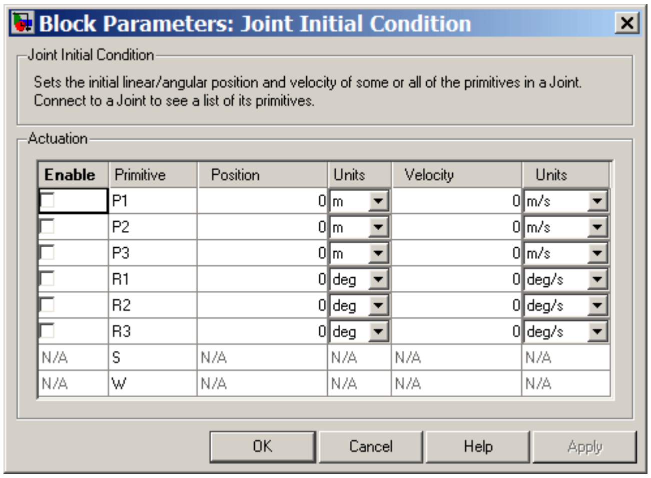
When you build your machine, the geometric configuration of the bodies (the home configuration) implicitly specifies the initial positions/angles of bodies relative to one another and to World. The Ground, Body, and Joint layout specifies only initial coordinates (degrees of freedom or DoFs), not their corresponding velocities. Starting a simulation in this state sets all initial velocities to zero. You can set the full initial kinematic state (the initial configuration), both positions and velocities, of joint primitives by using Joint Initial Condition Actuator blocks.

Initial Condition Actuation in Open and Closed Topologies

In a SimMechanics model, the DoFs represented by Joints are relative. Suppose you actuate a Joint with initial conditions, and that Joint has other Joints in a sequence connected to it through intermediate Bodies. Then the initial conditions applied to the first Joint change the absolute positions and velocities of the other Joints (as measured in World) because the initial conditions of the other Joints are defined relative to the first.

The one exception to this rule occurs if the actuated Joint is part of a closed loop. SimMechanics cuts one Joint in each closed loop. The initial conditions applied to a Joint indirectly affect the initial conditions of the other connected Joints only up to (but not including) the cut Joint.

Dialog Box and Parameters



Sets the initial linear/angular position and velocity of some or all of the primitives in a Joint. Connect to a Joint to see a list of its primitives.

Enable	Primitive	Position	Units	Velocity	Units
☐	P1	0	m	0	m/s
☐	P2	0	m	0	m/s
☐	P3	0	m	0	m/s
☐	R1	0	deg	0	deg/s
☐	R2	0	deg	0	deg/s
☐	R3	0	deg	0	deg/s
N/A	S	N/A	N/A	N/A	N/A
N/A	W	N/A	N/A	N/A	N/A

The dialog has one active area, **Actuation**.

Actuation

The menu choices are available for every primitive in the Joint to which the Joint Initial Condition Actuator is connected. If you connect the Actuator with its dialog open, the primitive list is automatically updated to reflect the connected Joint's primitives. If the Actuator is unconnected, all primitive types are shown, including two that cannot be actuated, spherical (S) and weld (W).

Enable
: Select this check box if you want to actuate the primitive with initial conditions. The default is not selected.

Primitive
: Displays the name of the primitive within the Joint. Not an active field.

Position
: Enter a value for the initial position of the primitive, either prismatic or revolute. The default is 0.

Units

In the pull-down menu, select units for the initial position. The defaults are `m` (meters) for prismatic primitives and `deg` (degrees) for revolute primitives.

Velocity

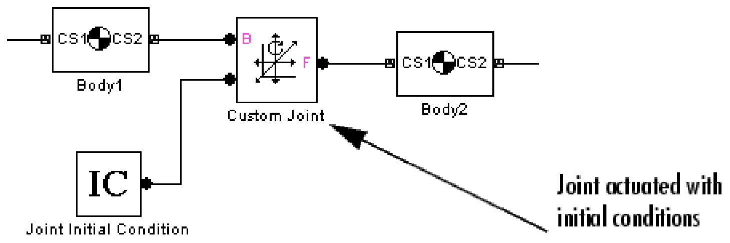
Enter a value for the initial velocity of the primitive, either prismatic or revolute. The default is `0`.

Units

In the pull-down menu, select units for the initial velocity. The defaults are `m/s` (meters/second) for prismatic primitives and `deg/s` (degrees/second) for revolute primitives.

Example

Here is a Joint Initial Condition Actuator connected to a Custom Joint, which connects two Bodies:



You must add an Actuator port (connector port) to the Joint block to connect the Joint Initial Condition Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Initial Condition Actuator data.

See Also

Joint Actuator, Joint Sensor, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute

See Chapter 3, “Representing Motion” for more details about a machine’s state of motion. “How SimMechanics Works” on page 5-15 explains how SimMechanics initializes a machine.

See “Using JICA Blocks” on page 4-58 for setting general initial conditions (positions and velocities) of DoFs in a machine. “Cutting Closed Loops” on page 4-36 and “Verifying Machine Topology” on page 4-74 discuss how SimMechanics cuts Joints in closed loops.

See “Joints” on page 10-2.

In Simulink, see the Signal Routing Library and the Sources Library.

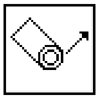
Purpose

Measure motion of and force or torque on joint primitive

Library

Sensors & Actuators

Description



The Joint Sensor block measures the position, velocity, and/or acceleration of a joint primitive in a Joint block.

The Joint Sensor measures the motion along/about the joint axis (or about the pivot point for a spherical primitive) in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog. The Joint connects a base and a follower Body. The base-follower sequence determines the sense of the motion.

Depending on the joint primitive being sensed, you measure one of these motion types:

- Translational motion for a prismatic joint primitive, in terms of linear position, velocity, and/or acceleration
- Rotational motion for a revolute joint primitive, in terms of angular position, velocity, and/or acceleration
- Spherical motion for a spherical joint primitive, in terms of a quaternion, quaternion derivative, and/or quaternion second derivative

The input is the connector port connected to the Joint being sensed. The outport is a set of Simulink signals or one bundled Simulink signal of the position, velocity, and/or acceleration of the joint primitive.

A Joint Sensor block measures one joint primitive at a time:

- A primitive Joint (Prismatic or Revolute) has only one primitive within the Joint to sense.
- A composite Joint has multiple joint primitives within, and you must choose which primitive to sense with the Joint Sensor.

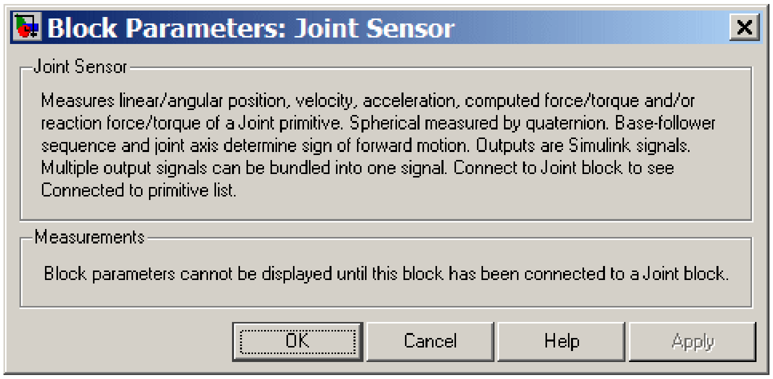
A body's orientation rotation matrix R relates vector components measured in the body CS and in the inertial World CS by $[R]\cdot\boldsymbol{v}_\mathrm{b} = \boldsymbol{v}_\mathrm{s}$.

The column vector $\boldsymbol{v}_b$ lists the vector $\boldsymbol{v}$'s three components measured in the body CS. The column vector $\boldsymbol{v}_s$ lists the vector $\boldsymbol{v}$'s three components measured in the World CS.

Joint Measurement and the Home Configuration

The Joint Sensor block measures the state of a degree of freedom, translational or rotational. It measures this state relative to the home configuration of the machine, the machine state *before* the application of initial condition actuators and assembly of disassembled joints. Thus the Joint Sensor includes the effect of the latter, which act before the simulation starts.

Dialog Box and Parameters


Block Parameters: Joint Sensor

Joint Sensor

Measures linear/angular position, velocity, acceleration, computed force/torque and/or reaction force/torque of a Joint primitive. Spherical measured by quaternion. Base-follower sequence and joint axis determine sign of forward motion. Outputs are Simulink signals. Multiple output signals can be bundled into one signal. Connect to Joint block to see Connected to primitive list.

Measurements

Block parameters cannot be displayed until this block has been connected to a Joint block.

The dialog has one active area, **Measurements**. The block parameters are not displayed unless you connect it to a specific Joint block.

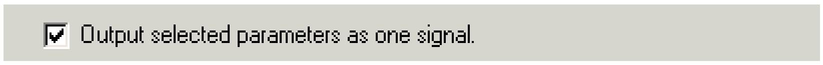
Measurements

Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint that you want to measure with the Joint Sensor. A primitive Joint block has only one joint primitive.

If the Joint Sensor is not connected to a Joint block, this menu is not shown.

Output selected parameters as one signal



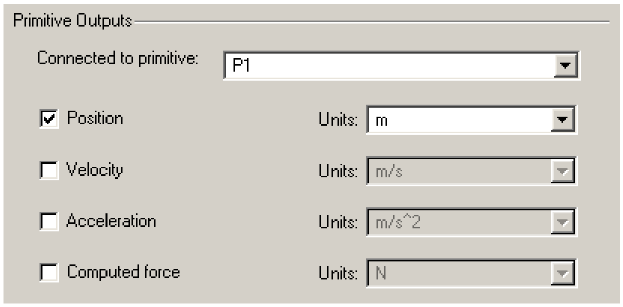
☑ Output selected parameters as one signal.

Select this check box to convert all the output signals into a single bundled signal. The default is selected. If you clear it, the Joint Sensor block will grow as many Simulink outports as there are active signals selected, in the same order top to bottom, in the dialog.

If the check box is selected, the Simulink signal out has all the active signals ordered into a single row vector. The order and type of the signal components depend on the joint primitive, as listed in the Simulink signal tables following.

The **Measurements** pane you see in the Joint Sensor dialog depends on the type of joint primitive to which you connect the Joint Sensor.

Measuring Prismatic Motion



Primitive Outputs

Connected to primitive: P1

☑ Position	Units: m
☐ Velocity	Units: m/s
☐ Acceleration	Units: m/s^2
☐ Computed force	Units: N

In the **Primitive Outputs** area, select the check box(es) for each of the possible measurements you want to make: **Position**, **Velocity**, **Acceleration**, and **Computed force**.

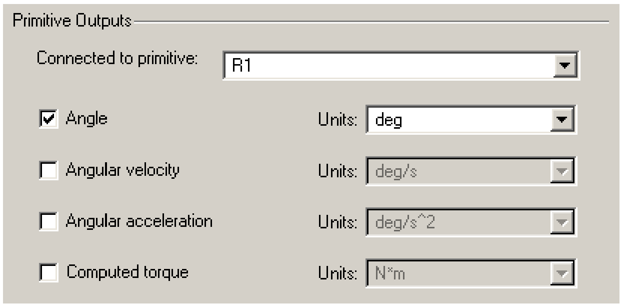
The *computed force* is that force along the prismatic axis which reproduces the follower motion with respect to base.

In the **Units** pull-down menus, choose the units for each of the measurements you want. The defaults are `m` (meters), `m/s` (meters/second), `m/s`^{2} (meters/second^2), `N` (newtons), respectively, for **Position**, **Velocity**, **Acceleration**, and **Computed force**.

The bundled Simulink output signal for a prismatic primitive has these measurements ordered in a row vector. Nonselected components are removed from the vector signal:

Position	Velocity	Acceleration	Computed Force	Reaction Torque (3-vector)	Reaction Force (3-vector)

Measuring Revolute Motion



Primitive Outputs

Connected to primitive: R1

[✓] Angle Units: deg

[ ] Angular velocity Units: deg/s

[ ] Angular acceleration Units: deg/s^2

[ ] Computed torque Units: N*m

In the **Primitive Outputs** area, select the check box(es) for each of the possible measurements you want to make: **Angle**, **Angular velocity**, **Angular acceleration**, and **Computed torque**.

The *computed torque* is that torque about the revolute axis which reproduces the follower motion with respect to base.

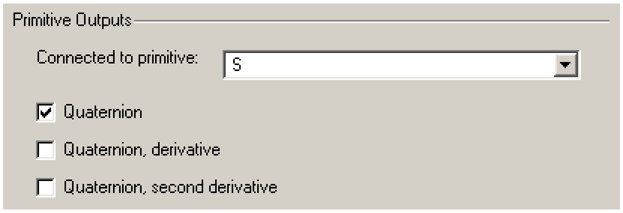
In the **Units** pull-down menus, choose the units for each of the measurements you want. The defaults are `deg` (degrees), `deg/s` (degrees/second), `deg/s`^{2} (degrees/second^2), `N*m` (newton-meters), and `N` (newtons), respectively, for **Angle**, **Angular Velocity**, **Angular Acceleration**, and **Computed torque**.

The bundled Simulink output signal for a revolute primitive has these measurements ordered in a row vector. Nonselected components are removed from the vector signal:

Angle	Angular Velocity	Angular Acceleration	Computed Torque	Reaction Torque (3-vector)	Reaction Force (3-vector)

Note The absolute angle of revolute motion is mapped on to the interval (-180^o, +180^o] degrees or (-π,+π] radians.

Measuring Spherical Motion



Primitive Outputs

Connected to primitive: S

[✓] Quaternion

[ ] Quaternion, derivative

[ ] Quaternion, second derivative

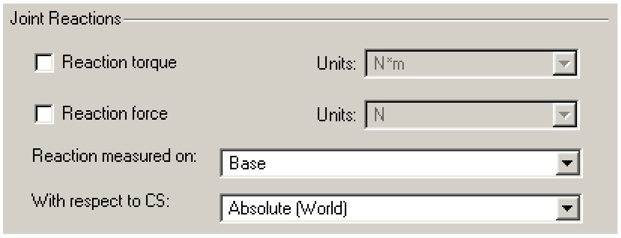
In the **Primitive Outputs** area, select the check box(es) for each of the possible measurements you want to make: **Quaternion**, **Quaternion, derivative**, and **Quaternion, second derivative**.

Quaternions are dimensionless, 4-component row vectors. The time unit for the derivatives is seconds.

The bundled Simulink output signal for a spherical primitive has these quaternion measurements ordered into a larger row vector. Nonselected components are removed from the vector signal:

Quaternion (4-vector)	Quaternion, derivative (4-vector)	Quaternion, second derivative (4-vector)	Reaction Torque (3-vector)	Reaction Force (3-vector)

Reaction Force and Torque



In the **Joint Reactions** area, select the check box(es) for each of the possible measurements you want to make. The *reaction force* and *torque* are 3-component vectors of the force and torque that the joint primitive transfers to the base or follower Body.

Reaction torque
Select the check box to output the reaction torque.

Reaction force
Select the check box to output the reaction force.

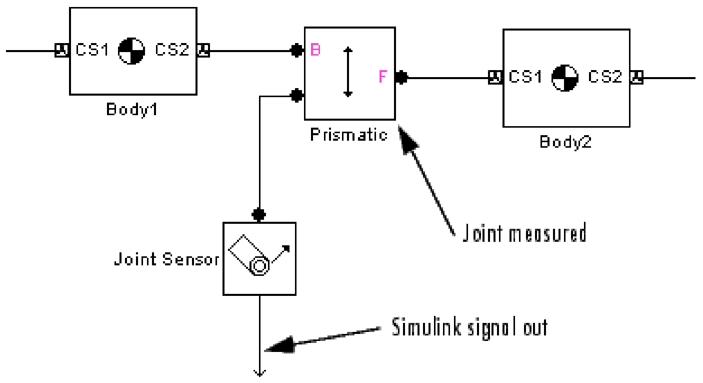
Reaction measured on
Choose the Body on which the reaction force and torque vectors are measured, `Base` or `Follower`. The default is `Base`.

With respect to CS
In the pull-down menu, choose the coordinate system in which the reaction torque and force vectors are measured: either the `Local (Body CS)` to which the Sensor is connected or the default `Absolute` (World).

In the `Absolute` case, the force and torque vectors have components measured relative to the inertial World CS axes. In the `Local` case, the same force and torque signals are premultiplied by the inverse orientation rotation matrix $R^{-1} = R^{T}$ for the Body selected in **Reactions measured on**.

Example

Here is a Joint Sensor connected to a Prismatic that connects two Bodies:



You must add an Sensor port (connector port) to the Joint block to connect the Joint Sensor to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Sensor data.

See Also

Body Sensor, Constraint & Driver Sensor, Joint Actuator, Joint Initial Condition Actuator, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute, Spherical

See “Kinematics and the Machine’s State of Motion” on page 3-2, “Body Motion in SimMechanics” on page 3-4, and “Modeling Sensors” on page 4-63.

In Simulink, see the Signal Routing Library and the Sinks Library.

Joint Spring & Damper

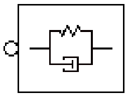
Purpose

Model damped linear oscillator force or torque on prismatic or revolute joint between two bodies

Library

Force Elements

Description



The Joint Spring & Damper block models a damped linear oscillator force acting along a prismatic primitive or a damped linear oscillator torque acting about a revolute primitive. The joint primitives are connected between two bodies, and the force or torque acts between these bodies. The sign of the force or torque is set by the base (B)-to-follower (F) sequence of the bodies. These models represent damped linear translational and torsional springs in the prismatic and revolute cases, respectively.

You connect this block to a Joint at one of the Joint's sensor/actuator ports. (If the Joint lacks a sensor/actuator port, open its dialog and create one.) The Joint represents any mixture of translational and rotational degrees of freedom (DoFs). With the Joint Spring & Damper block, you can then apply any combination of damped linear oscillator forces on any prismatics and damped linear torsion torques on any revolutes.

Note Each Joint Spring & Damper block connected to a revolute primitive adds a Simulink state to your model. These states are in addition to other normal Simulink states, such as those associated with Integrator and Transfer Fcn blocks. See the Simulink documentation for more about Simulink model states.

This feature does not change the mechanical states of your model, those states associated with SimMechanics joint blocks. See the `mech_stateVectorMgr` command reference for more about mechanical states.

Joint Spring and Damper Theory

Connect two Bodies with a Joint having some combination of prismatic and revolute primitives.

Caution The Joint Spring & Damper uses a Joint Sensor to measure the degree of freedom in the Joint. These values are measured relative to the home configuration of the DoF, its state *before* the application of initial condition actuators and assembly of disassembled joints.

Translational Case

If x represents the displacement along a prismatic axis, and $v = dx/dt$ is the prismatic DoF's linear speed, then the damped spring force acting along this prismatic and between the Bodies connected by this Joint is

$$F = -k(x - x_0) - bv$$

The model parameters are the spring constant k, the natural spring length (offset) x_0, and the damping constant b. The natural length is the spring's length with no forces acting on it and should be nonnegative: $x_0 \geq 0$. A stable spring requires $k > 0$. A damping representing dissipation and respecting the second law of thermodynamics requires $b \geq 0$. You can use a negative b to represent energy pumping.

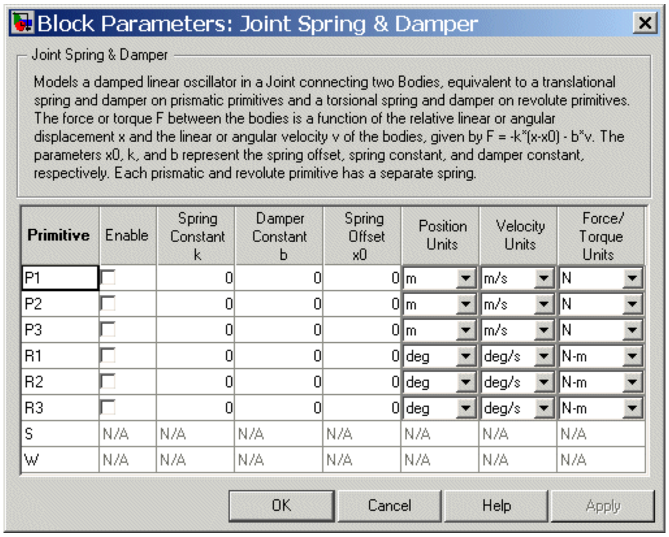
Rotational Case

If θ represents the displacement about a revolute axis, and $\omega = d\theta/dt$ is the revolute DoF's angular speed, then the damped torsion torque acting about this revolute and between the Bodies connected by this Joint is

$$\tau = -k(\theta - \theta_0) - b\omega$$

The model parameters are the torsion constant k, the natural torsion angle (offset) θ_0, and the damping constant b. The natural angle is the torsion balance's direction with no torques acting on it and can have any sign. A stable torsion requires $k > 0$. A damping representing dissipation and respecting the second law of thermodynamics requires $b \geq 0$. You can use a negative b to represent energy pumping.

Dialog Box and Parameters



Models a damped linear oscillator in a Joint connecting two Bodies, equivalent to a translational spring and damper on prismatic primitives and a torsional spring and damper on revolute primitives. The force or torque F between the bodies is a function of the relative linear or angular displacement x and the linear or angular velocity v of the bodies, given by F = -k*(x-x0) - b*v. The parameters x0, k, and b represent the spring offset, spring constant, and damper constant, respectively. Each prismatic and revolute primitive has a separate spring.

Primitive	Enable	Spring Constant k	Damper Constant b	Spring Offset x0	Position Units	Velocity Units	Force/ Torque Units
P1	☐	0	0	0	m	m/s	N
P2	☐	0	0	0	m	m/s	N
P3	☐	0	0	0	m	m/s	N
R1	☐	0	0	0	deg	deg/s	N-m
R2	☐	0	0	0	deg	deg/s	N-m
R3	☐	0	0	0	deg	deg/s	N-m
S	N/A	N/A	N/A	N/A	N/A	N/A	N/A
W	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Actuation

The menu lists all the active primitives in the Joint to which the Joint Spring & Damper block is connected. If you connect the Joint Spring & Damper with its dialog open, the primitive list is automatically updated to reflect the connected Joint's primitives.

Primitive
Lists the active primitives in the Joint to which the block is connected. `P` represents a prismatic primitive, `R` a revolute primitive, `S` a spherical primitive, and `W` a weld primitive.

Enable
To enable force or torque actuation on any particular primitive in the Joint, select the **Enable** check box next to that primitive's name in the Primitive column. You cannot actuate spherical or weld primitives.

Spring Constant k

Enter the spring or torsion constant *k*, for a prismatic or revolute primitive, respectively. The default is `0`.

The units for *k* are derived implicitly from your choice of position and force/torque units.

Damper Constant b

Enter the spring or torsion damping constant *b*, for a prismatic or revolute primitive, respectively. The default is `0`.

The units for *b* are derived implicitly from your choice of velocity and force/torque units.

Spring Offset x0

Enter the natural spring length x_0 or the natural torsion angle θ_0, for a prismatic or revolute primitive, respectively. The default is `0`.

Position Units

In the pull-down menu, select linear or angular units for prismatic or revolute primitives, respectively. The default is `m` (meters) or `deg` (degrees).

Velocity Units

In the pull-down menu, select linear or angular velocity units for prismatic or revolute primitives, respectively. The default is `m/s` (meters/second) or `deg/s` (degrees/second).

Force/Torque Units

In the pull-down menu, select force or torque units for prismatic or revolute primitives, respectively. The default is `N` (newtons) or `N*m` (newton-meters).

See Also

Body, Body Spring & Damper, Custom Joint, Joint Actuator, Joint Sensor, Prismatic, Revolute

See “Modeling Force Elements” on page 4-69.

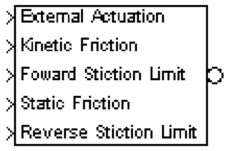
Purpose

Apply classical friction to joint primitive

Library

Sensors & Actuators

Description



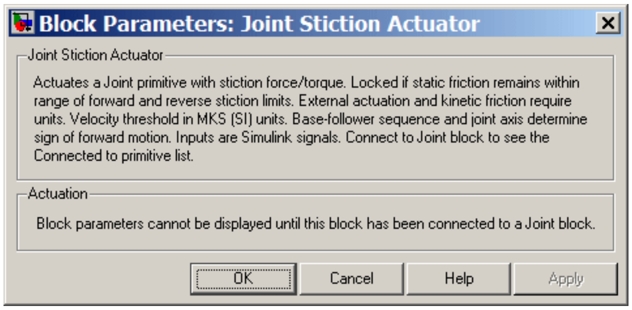
The Joint Stiction Actuator block applies stiction (classical friction) to a prismatic or revolute joint primitive. The stiction is regulated by a friction model whose parameters you specify. (See "Stiction Theory" on page 11-132.) The Joint Stiction Actuator applies stiction to the joint primitive as a relative force/torque between the joint's connected Bodies. The bodies can experience additional forces independent of the applied stiction.

The inports are Simulink signals. The output is a connector port. You cannot connect a Joint Stiction Actuator to a Spherical block or spherical primitive. Restrictions on simultaneous actuators and sensors include:

- You cannot actuate a joint primitive simultaneously with a Joint Stiction Actuator and a Joint Actuator. But with the Joint Stiction Actuator inport `External Actuation`, you can apply to the joint primitive an external (nonfrictional) force/torque actuation signal equivalent to applying a Joint Actuator.
- You can simultaneously actuate a joint primitive with a Joint Stiction Actuator and a Joint Initial Condition Actuator.
- You can also simultaneously actuate a joint primitive with a Joint Stiction Actuator and measure the force/torque along/around the joint primitive with a Joint Sensor.

Caution You cannot trim or linearize a SimMechanics model that contains a Joint Stiction Actuator block.

Dialog Box and Parameters



Block Parameters: Joint Stiction Actuator

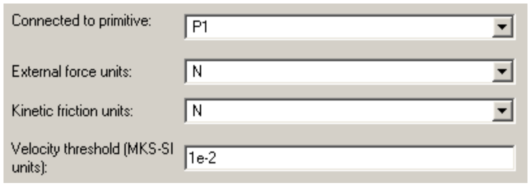
Joint Stiction Actuator

Actuates a Joint primitive with stiction force/torque. Locked if static friction remains within range of forward and reverse stiction limits. External actuation and kinetic friction require units. Velocity threshold in MKS (SI) units. Base-follower sequence and joint axis determine sign of forward motion. Inputs are Simulink signals. Connect to Joint block to see the Connected to primitive list.

Actuation

Block parameters cannot be displayed until this block has been connected to a Joint block.

The dialog has one active area, **Actuation**. The block parameters are not displayed unless you connect it to a specific Joint block.



Connected to primitive: P1

External force units: N

Kinetic friction units: N

Velocity threshold (MKS-SI units): 1e-2

Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint that you want to actuate with the Joint Stiction Actuator. A primitive Joint block has only one joint primitive.

You cannot connect a Joint Stiction Actuator to a spherical primitive.

If the Joint Stiction Actuator is not connected to a Joint block, this menu displays `Unknown`.

External force units

In the pull-down menu, choose units for the external nonfrictional force/torque F_{ext}. The default is `N` (newtons) if connected to a prismatic primitive and `N*m` (newton-meters) if connected to a revolute primitive.

Kinetic friction units

In the pull-down menu, choose units for the kinetic friction force/torque F_K. The default is `N` (newtons) if connected to a prismatic primitive and `N*m` (newton-meters) if connected to a revolute primitive.

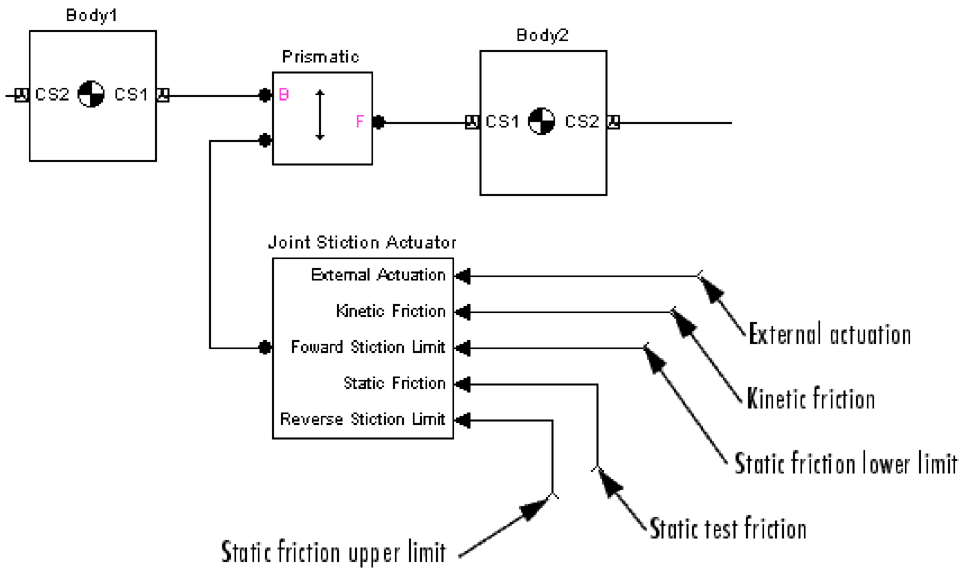
Velocity threshold (MKS-SI units)

Enter the positive relative speed of the joint primitive below which the joint locks by static friction. Above that speed, the joint is unlocked.

The units must be MKS or SI: for a prismatic primitive, meters/second; for a revolute primitive, radians/second.

Summary of Joint Stiction Actuator Inport Signals

All the Simulink inports are one-component signals. Here is an example of a prismatic joint connected between two bodies and actuated with stiction:



Joint Stiction Actuator Simulink Inport Signals

Simulink Inport	Friction Model	Description
`External Actuation`	F_{ext}	External nonfrictional force/torque
`Kinetic Friction`	F_{K}	Kinetic friction
`Forward Stiction Limit`	$F_{\mathrm{S}}^{\mathrm{f}} < 0$	Static friction lower limit
`Static Test Friction`	F_{test}	Static test friction
`Reverse Stiction Limit`	$F_{\mathrm{S}}^{\mathrm{r}} > 0$	Static friction upper limit

Units

You specify units in the dialog only for the external nonfrictional and kinetic friction forces/torques, F_{ext} and F_{K}. These two friction signals are used to integrate the motion of the joint and have physical significance in the model. Thus units are necessary for F_{ext} and F_{K}.

The other three signals are compared only to one another in the locking condition $F_{\mathrm{S}}^{\mathrm{f}} < F_{\mathrm{test}} < F_{\mathrm{S}}^{\mathrm{r}}$. These friction signals are not used to integrate motion and thus do not have units set in the dialog. But they must have the same implicit units for a valid comparison.

Caution The threshold velocity v_{th} must be set greater than the **Absolute tolerance** in the **Solver** node of your model's **Configuration Parameters** dialog to avoid a meaningless threshold value.

Never set **Absolute tolerance** to `auto` if stiction actuators are present in a model. A recommended setting is to make v_{th} at least 10 times the **Absolute tolerance** value.

See "Controlling the Simulation" on page 5-11 for a discussion of setting simulation parameters.

Example

The `mech_dpen_sticky` model in the Demos library has two revolute joints actuated with stiction. See "Joint Stiction Actuator Example: Mixed Static and Kinetic Friction" on page 4-56.

Stiction Theory

Kinematics

v and a are the velocity and acceleration along or around a joint primitive axis. These quantities are relative between the two bodies at the joint ends and signed ± to indicate forward or reverse. The joint directionality is set by the base (B)-to-follower (F) Body sequence of Bodies attached to the joint primitive being actuated.

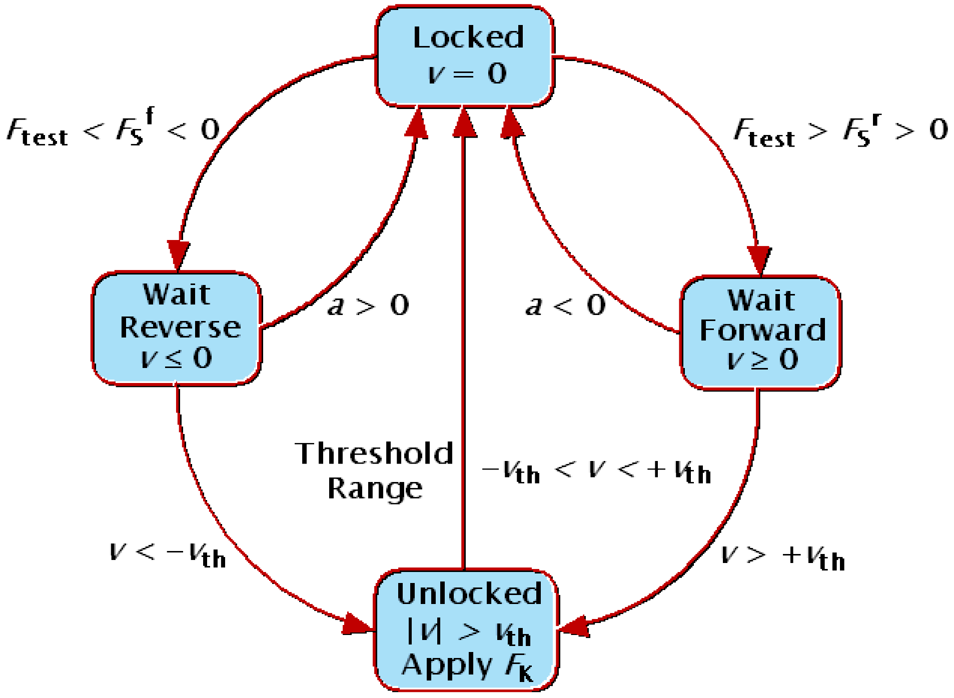
Continuous Motion

A joint subject to stiction, if unlocked, moves in continuous motion. During this motion, you can apply two forces/torques at the joint primitive:

- A kinetic friction force/torque F_K:
  - $F_K < 0$ retards forward motion
  - $F_K > 0$ retards reverse motion
- An external, nonfrictional force/torque F_{ext}

Discrete Modes: Locked, Wait, Unlocked

Besides its continuous motion mode, a joint actuated by stiction has two other discrete modes. The Joint Stiction Actuator switches a joint primitive between locked and unlocked modes. In one mode, the joint locks rigidly; in the other, it moves with the kinetic friction and external nonfrictional forces/torques applied. The joint can also be in a wait mode, between locked and unlocked.



Joint Stiction Modes and Transition Conditions

Unlocking

You specify the unlocking criteria by a two-condition threshold, constructed from four user-specified inputs.

- Joint unlocking threshold velocity $v_{th} > 0$ via the block dialog.
- *Static* friction limits $F_S{}^f < 0$ and $F_S{}^r > 0$ for forward and reverse motion, and a *static test friction* F_{test}, all three specified via Simulink signals. The static test friction F_{test} and forward/reverse limits $F_S{}^f$ and $F_S{}^r$ can be functions of the machine state and/or time.

The static test and kinetic frictions F_{test} and F_K can be discontinuous, but should be physically sensible.

Locking
You specify the locking criterion with the velocity threshold alone.

- Joint locking threshold velocity $v_{\mathrm{th}} > 0$ via the block dialog.

Locked Mode

In this mode, v and a of the joint are zero. The static computed force/torque F_{S} at the joint is internally computed to maintain this mode: $F_{\mathrm{ext}} + F_{\mathrm{S}} + F_{\mathrm{F}} - F_{\mathrm{B}} = 0$. The forces/torques F_{B}, F_{F} are the forces/torques on the base and follower Bodies apart from those forces/torques acting at the joint.

The joint remains locked as long as $F_{\mathrm{S}}^{\mathrm{f}} < F_{\mathrm{test}} < F_{\mathrm{S}}^{\mathrm{r}}$.

In most realistic friction models, you would set F_{test} equal to the computed F_{S}.

Wait Mode

If the static test friction F_{test} leaves the static friction range $[F_{\mathrm{S}}^{\mathrm{f}}, F_{\mathrm{S}}^{\mathrm{r}}]$, the joint has passed the first condition for unlocking, and the simulation enters wait mode, suspending the mechanical motion.

A search begins for a consistent state of all stiction-actuated joints in your model.

- The potential direction of motion after unlocking is determined by all the nonfrictional forces on the bodies.
- During the search, the net force/torque $F = F_{\mathrm{ext}} + F_{\mathrm{K}}$ at the joint primitive is computed, where F_{K} is the kinetic friction, and a is determined.
- For potential motion in the forward (reverse) direction, if $a < 0$ ($a > 0$), the search returns to the locked mode.

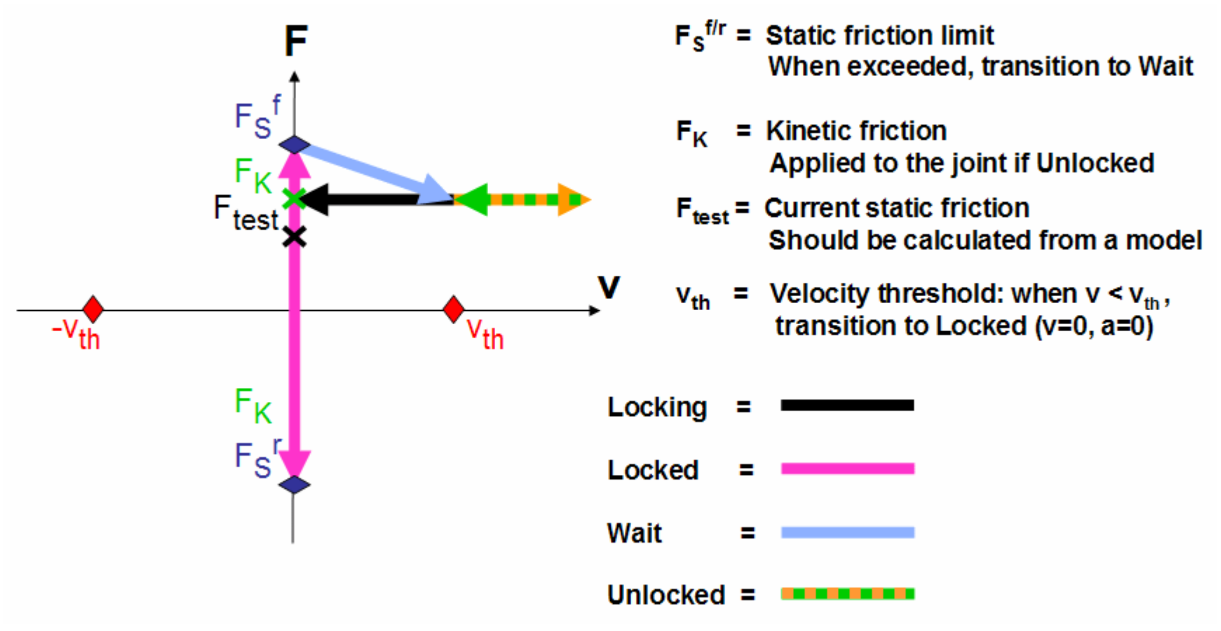
Once a consistent state for all stiction-actuated joints are found, mechanical motion restarts. The simulation integrates a to obtain v. When $|v|$ exceeds v_{th}, the second condition, the joint unlocks.

The wait mode prevents infinite cycling between locked and unlocked modes, although it can noticeably slow down the simulation. The mode search uses a nonphysical algebraic loop, which displays warnings at the MATLAB command line.

Unlocked Mode

In the unlocked mode, the joint primitive moves, actuated by the sum of the external, nonfrictional force/torque F_{ext} and the kinetic friction F_{K}.

The joint returns to the locked mode if v falls into the range $-v_{\text{th}} < v < +v_{\text{th}}$. If the simulation steps in time over this velocity range, it instead catches the zero of velocity with Simulink zero-crossing detection.


F
Fs^f
FK
Ftest
V
-Vth
Vth
FK
Fs^r
Fs^f/r = Static friction limit
When exceeded, transition to Wait
FK = Kinetic friction
Applied to the joint if Unlocked
Ftest = Current static friction
Should be calculated from a model
Vth = Velocity threshold: when v < Vth,
transition to Locked (v=0, a=0)
Locking =
Locked =
Wait =
Unlocked =

Static and Kinetic Friction and Relative Velocity

See Also

Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Mechanical Branching Bar, Prismatic, Revolute

See “Actuating a Joint” on page 4-52. For trimming and linearization, see Chapter 8, “Analyzing Motion”.

In Simulink, see the Signal Routing Library and the Sources Library, and “Zero-Crossing Detection”.

Linear Driver

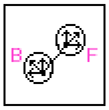
Purpose

Specify component of vector difference of two Body CS origins as function of time

Library

Constraints & Drivers

Description



The Linear Driver block specifies a component of the vector difference of Body coordinate system (CS) origins as a function of time.

Let $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ be the vector positions of the origins of CS1 on one Body, CS2 on the other Body, and $\boldsymbol{R} = \boldsymbol{r}_1 - \boldsymbol{r}_2$. The Linear Driver block specifies one of the vector components of R = (X,Y,Z), projected on to the World CS axes, as a function of time:

$$X, Y, \text{ or } Z = X(t=0), Y(t=0), \text{ or } Z(t=0) + f(t)$$

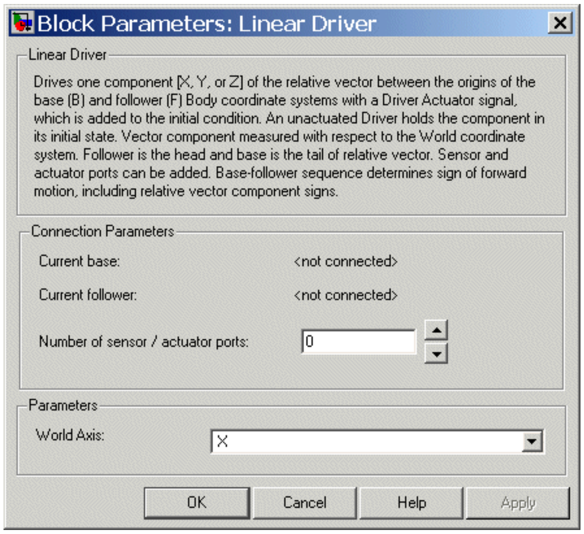
You connect a Driver Actuator block to the Linear Driver.

The Simulink input signal into the Driver Actuator specifies the time-dependent driving function $f(t)$ and its first two derivatives, as well as their units. If you do not actuate Linear Driver, this block acts as a time-independent constraint that freezes the vector component between the two Body CS origins at its initial value $X(t=0)$, $Y(t=0)$, or $Z(t=0)$ during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver and measure the reaction forces/torques between the driven bodies.

Dialog Box and Parameters



Linear Driver

Drives one component [X, Y, or Z] of the relative vector between the origins of the base (B) and follower (F) Body coordinate systems with a Driver Actuator signal, which is added to the initial condition. An unactuated Driver holds the component in its initial state. Vector component measured with respect to the World coordinate system. Follower is the head and base is the tail of relative vector. Sensor and actuator ports can be added. Base-follower sequence determines sign of forward motion, including relative vector component signs.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

World Axis: X

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

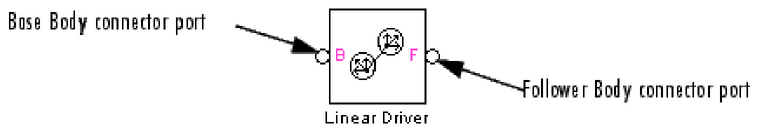
Current base
: When you connect the base (B) connector port on the Linear Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Linear Driver Base and Follower Body Connector Ports on page 11-140.

Current follower
: When you connect the follower (F) connector port on the Linear Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Linear Driver Base and Follower Body Connector Ports on page 11-140.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is `0`.

To activate the Driver, connect a Driver Actuator.



Linear Driver Base and Follower Body Connector Ports

Parameters

World Axis

In the pull-down menu, choose the component of the vector difference ***R*** between the Body CS origins that you want to drive as a function of time. The components are measured with respect to the World CS axes. The choices are `X`, `Y`, or `Z`. The default is `X`.

See Also

Constraint & Driver Sensor, Distance Driver, Driver Actuator

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Drivers.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15.

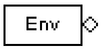
Purpose

Set up mechanical environment for machine

Library

Bodies

Description



The Machine Environment block allows you to view and change the mechanical environment settings for one machine in your model.

Note A SimMechanics model consists of one or more machines. A machine is a complete, connected diagram of SimMechanics blocks topologically distinct from other complete SimMechanics block diagrams. Each machine must have one or more Ground blocks.

A machine can be a composite of submachines connected by Shared Environment blocks. Each submachine must have one or more Ground blocks.

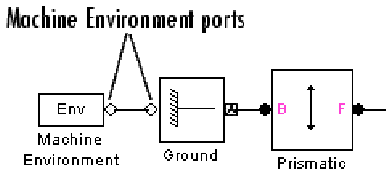
Exactly one Ground per machine, simple or composite, must be connected to a Machine Environment block for your SimMechanics model to be valid.

This block determines the following settings for the machine to which it is connected:

- Parameters that control how the machine is simulated
- Settings to control how constraints are interpreted
- Settings to control how linearization is implemented
- Whether the machine is displayed in SimMechanics visualization

The Machine Environment Port and Connecting the Block

You connect this block to a Ground by enabling that Ground's Machine Environment port from the Ground dialog.



Gravity as a Simulink Signal

This block also allows you to input gravity as a variable Simulink signal. If you choose to do this, a Simulink inport > also appears on the block for connection to a three-component Simulink signal line.

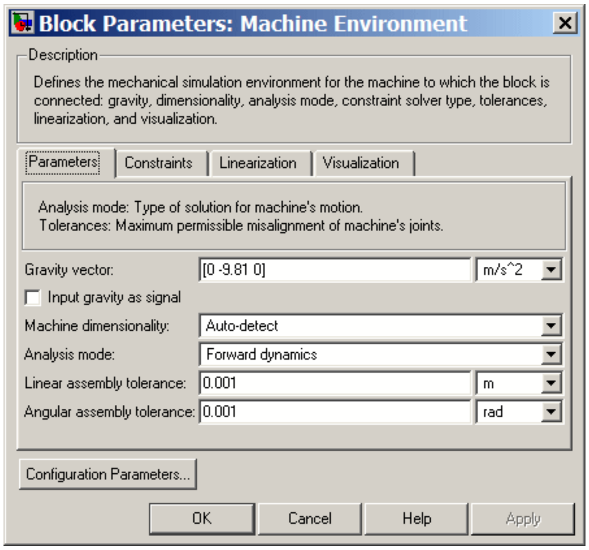
Opening the Simulink Configuration Parameters Dialog

You can open the Simulink **Configuration Parameters** dialog for viewing and editing by clicking the **Open Configuration Parameters** button on the lower left of the dialog.

Dialog Box and Parameters

In the lower half of its dialog, the Machine Environment block has four active panes that you can view and modify by selecting the corresponding tabs. You can apply your settings at any time by clicking **Apply** or **OK**.

- **Parameters**
- **Constraints**
- **Linearization**
- **Visualization**



Block Parameters: Machine Environment

Description

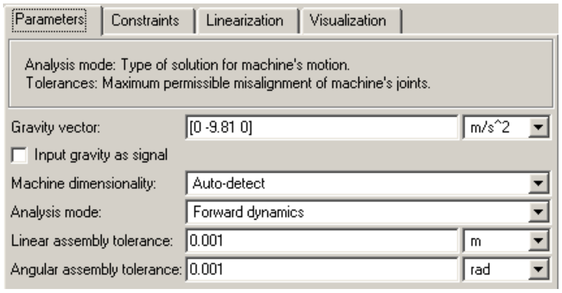
Defines the mechanical simulation environment for the machine to which the block is connected: gravity, dimensionality, analysis mode, constraint solver type, tolerances, linearization, and visualization.

Parameters | Constraints | Linearization | Visualization

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 -9.81 0] m/s^2
☐ Input gravity as signal
Machine dimensionality: Auto-detect
Analysis mode: Forward dynamics
Linear assembly tolerance: 0.001 m
Angular assembly tolerance: 0.001 rad

Configuring the Dynamics



Parameters | Constraints | Linearization | Visualization

Analysis mode: Type of solution for machine's motion.
Tolerances: Maximum permissible misalignment of machine's joints.

Gravity vector: [0 -9.81 0] m/s^2
☐ Input gravity as signal
Machine dimensionality: Auto-detect
Analysis mode: Forward dynamics
Linear assembly tolerance: 0.001 m
Angular assembly tolerance: 0.001 rad

In this pane, you configure settings that control the mechanical dynamics.

Gravity vector

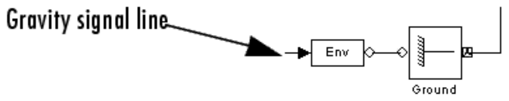
The value of this parameter is a MATLAB vector that specifies the magnitude and direction of gravitational acceleration in the model's world coordinate system. It must be a three-component vector. The default vector is `[0 -9.81 0]`. This field is disabled if you choose to input gravity as a signal.

The default units are `m/s`^{2} (meters per square second). Use the pull-down menu to the right if you want to reset the units.

Input gravity as signal

Select this check box if you want to disable the **Gravity vector** field and instead input gravity as a variable Simulink signal. The default is not selected.

If you select this check box, a Simulink inport appears on the block in addition to the existing Machine Environment port. You input the gravity vector as a three-component Simulink signal to this port. The components are, respectively, x, y, and z.



Machine dimensionality

In the pull-down menu, select in how many dimensions you want SimMechanics to simulate your machine: in `3D Only` or `2D Only`, or let SimMechanics choose for you with `Auto`. The default is `3D Only`.

You must take care, if you choose `2D Only`, that the machine actually moves in only two dimensions. If it does not, the simulation stops with an error.

Analysis mode

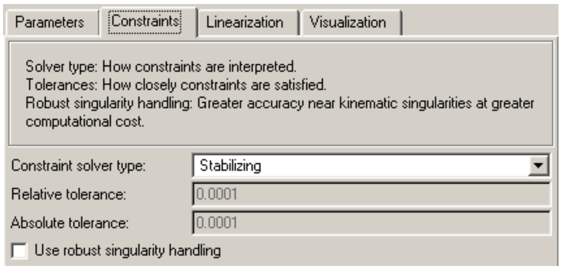
Specifies the type of analysis to be performed during the simulation. Choose one from the pull-down menu.

Analysis Mode	Description
`Forward dynamics`	Computes the positions and velocities of the system's bodies, given forces, torques, and initial conditions. This is the default mode.
`Inverse dynamics`	Computes the forces and torques required to produce the specified motions of an open system.
`Kinematics`	Computes the forces and torques required to produce the specified motions of a closed-loop system.
`Trimming`	Variant of Forward Dynamics mode to be used with the Simulink `trim` command. Determines steady-state or other points in system state space.

Linear assembly tolerance
Maximum position error allowed between bodies connected by prismatic joints. The default is `1e-3 m`. Use the menu on the right to set the units.

Angular assembly tolerance
Maximum angular error allowed between bodies connected by revolute joints. Default is `1e-3 rad`. Use the menu on the right to set the units.

Implementing Constraints



Solver type: How constraints are interpreted.
Tolerances: How closely constraints are satisfied.
Robust singularity handling: Greater accuracy near kinematic singularities at greater computational cost.

Constraint solver type: Stabilizing
Relative tolerance: 0.0001
Absolute tolerance: 0.0001
☐ Use robust singularity handling

In this pane, you tell SimMechanics how to interpret constraints in machines that contain blocks from the Constraints & Drivers library, cut Joint, Constraint, and Driver blocks in closed loops, or both.

Constraint solver type

Type of solver used to solve constraints on the mechanical system's states specified by the machine's constraint and driver blocks. Choose one from the pull-down menu.

Solver Type	Description
`Stabilizing`	Adds a self-correcting term to the state equations to be solved that stabilizes the numerical solution, i.e., causes it to evolve toward, rather than drift away from, the actual solution. This is the default.
`Tolerancing`	Solves the constraints on the system's states to a specified degree of accuracy.
`Machine precision`	Solves the constraints to the numerical precision of the computer on which the simulation is running.

Relative tolerance

The relative tolerance used by the tolerancing constraint solver to determine when to stop refining a solution. Default is `1e-4`.

Enabled only if **Constraint solver type** is set to `Tolerancing`.

Absolute tolerance

The absolute tolerance used by the tolerancing constraint solver to determine when to stop refining the solution of a machine state. Default is `1e-4`.

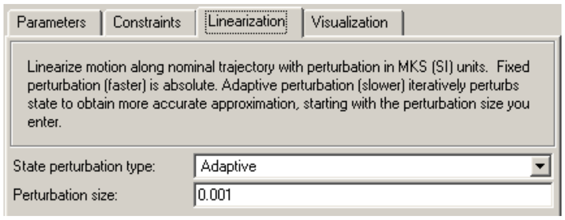
Enabled only if **Constraint solver type** is set to `Tolerancing`.

Use robust singularity handling

Select this check box if you want Simulink to take extra steps to handle singularities in a system's equations of motion. The default is not selected.

This option increases the length of time required to solve a system's equations of motion regardless of whether they have singularities. Hence, you should select this option only as a last resort, i.e., only if the Simulink solvers cannot otherwise solve the system's equations of motion or require an excessively long time to do so.

Configuring Linearization



Linearize motion along nominal trajectory with perturbation in MKS (SI) units. Fixed perturbation (faster) is absolute. Adaptive perturbation (slower) iteratively perturbs state to obtain more accurate approximation, starting with the perturbation size you enter.

State perturbation type: Adaptive

Perturbation size: 0.001

In this pane, you tell SimMechanics how to linearize your machine.

State perturbation type

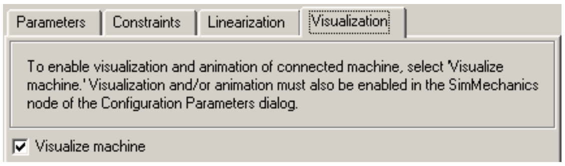
Specifies the type of state perturbation used by `linmod` to linearize a machine. The default is `Fixed`.

- `Adaptive` recomputes the size of the perturbation used at each step in the linearization process to ensure accurate computation of the linearization coefficients. It starts with the entry in the **Perturbation size** field as an initial guess.
- `Fixed` uses the perturbation size specified in the **Perturbation size** field for every step.

Perturbation size

Specifies the relative size of the perturbation used by the `Fixed perturbation` option. Specifies the relative size of the initial guess perturbation used by the `Adaptive` perturbation type. The perturbation size is relative to the size of the state being perturbed. The default is `1e-5`.

Turning Machine Visualization On or Off



To enable visualization and animation of connected machine, select 'Visualize machine.' Visualization and/or animation must also be enabled in the SimMechanics node of the Configuration Parameters dialog.

☑ Visualize machine

In this pane, you determine whether SimMechanics visualization displays this machine.

Visualize machine
Select this check box if you want the machine to which this block is connected to appear in the SimMechanics visualization window. The default is selected.

See Also

Ground, Shared Environment

For more about SimMechanics models and machines, see “Modeling Machines” on page 4-3. For more about using Grounds and creating valid SimMechanics models, see “Modeling Bodies and Grounds” on page 4-10. For more about modeling constraints, see “Modeling Constraints and Drivers” on page 4-38.

For more about running SimMechanics with Simulink, see “Running SimMechanics Models in Simulink” on page 5-2, “Configuring a Machine’s Mechanical Environment” on page 5-3, and “Controlling the Simulation” on page 5-11.

Chapter 6, “Visualizing and Animating Machines” discusses starting, configuring, and using SimMechanics visualization. See “Starting SimMechanics Visualization” on page 6-2.

Chapter 8, “Analyzing Motion” presents an in-depth look at the various motion analysis modes available in SimMechanics, starting with

"Analyzing Motion". For the Inverse Dynamics and Kinematics modes, see "Finding Forces from Motions" on page 8-7. For the Trimming mode, see "Trimming Mechanical Models" on page 8-18. To learn how to linearize mechanical models, see "Linearizing Mechanical Models" on page 8-32.

See the relevant entries in the Glossary: **dynamics**, **ground**, **kinematics**, **machine**, **machine precision constraint**, **stabilizing constraint**, and **tolerancing constraint**.

For more about configuring simulations in Simulink, consult the section on the Configuration Parameters dialog in the Simulink documentation.

Mechanical Branching Bar

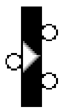
Purpose

Map multiple sensor or actuator lines to one sensor or actuator port on Joint, Constraint, or Driver, or to one Body coordinate system port on Body

Library

Utilities

Description



The Mechanical Branching Bar bundles multiple actuator and sensor connection lines into one line, allowing you to connect multiple actuators and/or sensors to a single connector port on a Joint, Constraint, or Driver, or to a single Body coordinate system (CS) port on a Body. You can choose any number of sensor/actuator ports on the Mechanical Branching Bar.

- In the case of a Body, a single Body CS port represents a single Body CS. If the needed Body CS port does not exist, open the Body dialog and create one. You can connect the selected Body CS to multiple Body Actuators and Sensors through the Mechanical Branching Bar.
- In the case of a Joint, you need a single sensor/actuator port on the Joint. If the needed port does not exist, open the Joint's dialog and create one. You can connect this sensor/actuator port to multiple Actuators and Sensors through the Mechanical Branching Bar.

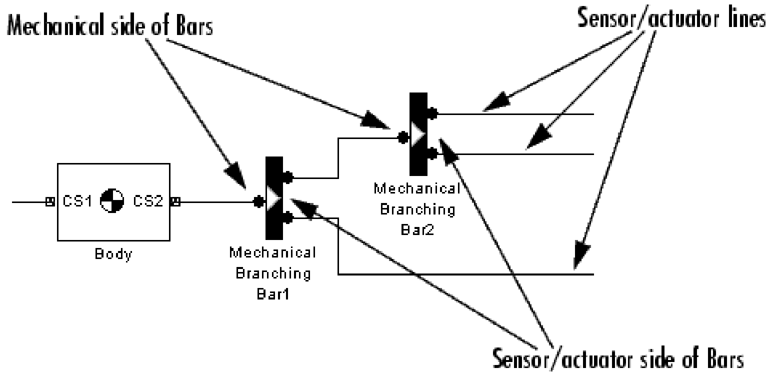
  Using the Mechanical Branching Bar, you can connect a Joint block to any combination of Joint Sensors, Joint Actuators, Joint Initial Condition Actuators, and Joint Stiction Actuators. The Actuator and Sensor dialogs display the Joint's primitives as if they were directly connected to the Joint.
- The procedure for Constraints and Drivers is the same as it is for Joints, except that you need to choose to measure reaction forces/torques or to actuate motions.

Cascading Mechanical Branching Bars and Avoiding Closed Loops

You can connect multiple Mechanical Branching Bar blocks in series, creating a cascade. Connect the mechanical side of the first Branching Bar to a Joint, Constraint, Driver, or Body. Then connect its sensor/actuator side to the mechanical side of the second Branching Bar, and so on.

The only restriction on cascading Mechanical Branching Bars is that you must avoid connecting them into closed loops.

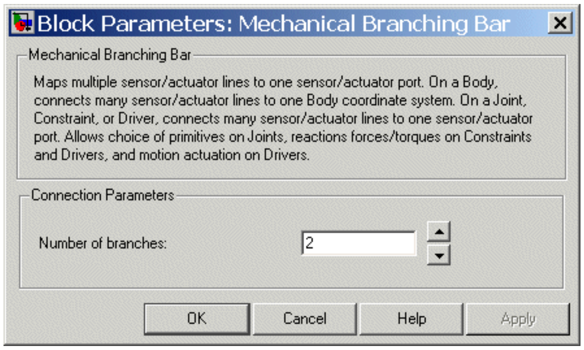
The following diagram shows a cascade, starting at a Body.


Mechanical side of Bars
Sensor/actuator lines
CS1
CS2
Body
Mechanical Branching Bar1
Mechanical Branching Bar2
Sensor/actuator side of Bars

Caution To avoid simulation errors, you should not create a cascade of Mechanical Branching Bars that closes on itself in a loop.

You should not connect the mechanical side of one Mechanical Branching Bar to the mechanical side of another Mechanical Branching Bar. You should also not connect the sensor/actuator side of one Mechanical Branching Bar to the sensor/actuator side of another Mechanical Branching Bar.

Dialog Box and Parameters



Mechanical Branching Bar

Maps multiple sensor/actuator lines to one sensor/actuator port. On a Body, connects many sensor/actuator lines to one Body coordinate system. On a Joint, Constraint, or Driver, connects many sensor/actuator lines to one sensor/actuator port. Allows choice of primitives on Joints, reactions forces/torques on Constraints and Drivers, and motion actuation on Drivers.

Connection Parameters

Number of branches: 2

The dialog has one active area, **Connection parameters**.

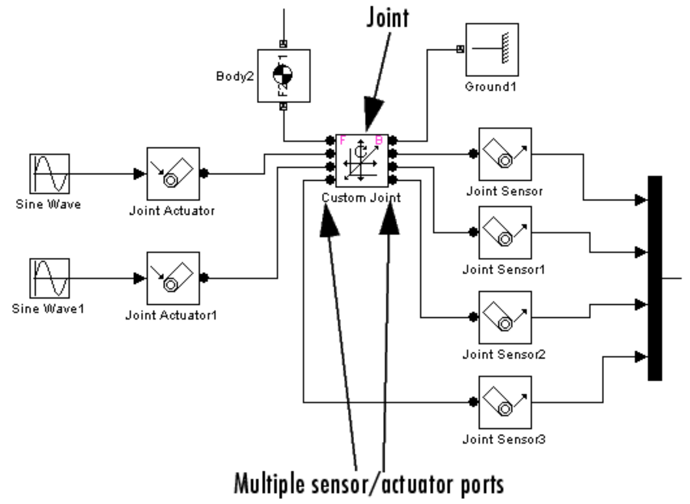
Connection Parameters

Number of branches

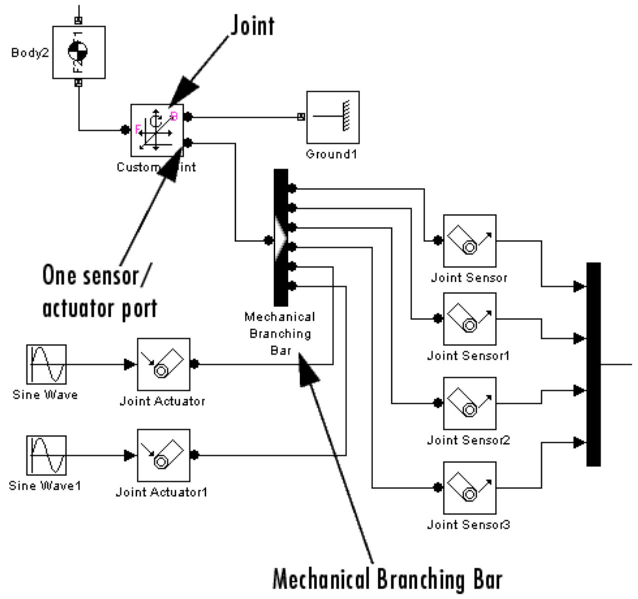
Using this spinner menu, you can set the number of extra connector ports needed for connecting Actuator and Sensor blocks to the Mechanical Branching Bar. The default is 2.

Example

Without the Mechanical Branching Bar, you must connect multiple Sensors and Actuators to a Joint by creating a sensor/actuator port on the Joint for each Sensor and each Actuator:



With the Mechanical Branching Bar block, you can combine all the sensor and actuator ports for a single Joint into one sensor/actuator port:



See Also

Body, Body Actuator, Body Sensor, Constraint & Driver Sensor, Driver Actuator, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator

Parallel Constraint

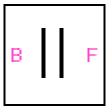
Purpose

Constrain body axis vectors of two bodies to be parallel

Library

Constraints & Drivers

Description



The two Bodies connected by a Parallel Constraint are restricted in their relative rotational motion. The Parallel Constraint is connected on either side to a Body CS, one on each Body. A vector $\boldsymbol{a}_\mathrm{B}$ defined in one Body CS on the base body remains parallel to a second vector $\boldsymbol{a}_\mathrm{F}$ defined in another Body CS on the follower body.

The Parallel Constraint block requires that:

$$|\boldsymbol{a}_B \cdot \boldsymbol{a}_F|/(|\boldsymbol{a}_B||\boldsymbol{a}_F|)=1$$

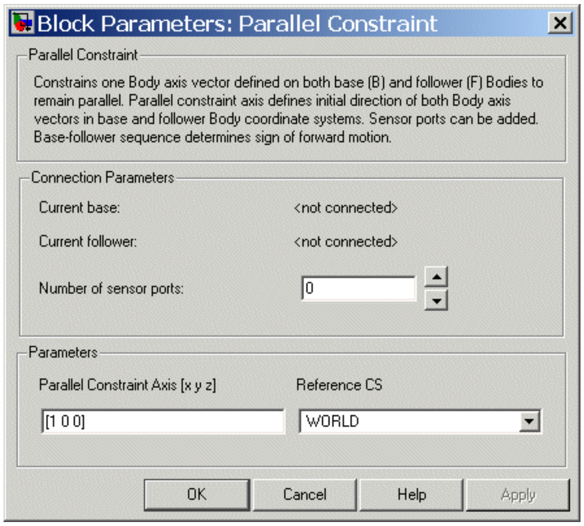
You specify the initial direction to which both vectors must remain parallel.

Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

Parallel Constraint is assembled: the Body CS origin on the base body must be initially collocated with the Body CS origin on the follower body, to within assembly tolerance.

You can connect a Constraint & Driver Sensor to any Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

Dialog Box and Parameters


Block Parameters: Parallel Constraint

Parallel Constraint

Constrains one Body axis vector defined on both base (B) and follower (F) Bodies to remain parallel. Parallel constraint axis defines initial direction of both Body axis vectors in base and follower Body coordinate systems. Sensor ports can be added. Base-follower sequence determines sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor ports: 0

Parameters

Parallel Constraint Axis [x y z]: [1 0 0]

Reference CS: WORLD

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.

Current base

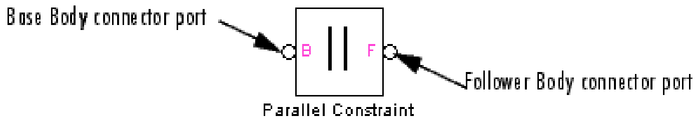
When you connect the base (B) connector port on the Parallel Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Parallel Constraint Base and Follower Body Connector Ports.

Current follower

When you connect the follower (F) connector port on the Parallel Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Parallel Constraint Base and Follower Body Connector Ports.

Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is `0`.



Parallel Constraint Base and Follower Body Connector Ports

Parameters

Parallel Constraint Axis [x y z]

Enter the axis vector defining the initial direction of the two body axis vectors $\boldsymbol{a}_b$, $\boldsymbol{a}_f$. These body axis vectors are restricted to always remain parallel to this initial axis. The default is `[1 0 0]`.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the initial **Parallel constraint axis** is oriented with respect to. This CS also determines the absolute meaning of reaction forces/torques at this Constraint. The default is `WORLD`.

See Also

Angle Driver, Constraint & Driver Sensor, Velocity Driver

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Constraints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on using constraints in closed loops.

See “Constraints and Drivers” on page 10-5.

Planar

Purpose

Represent composite joint with two translational DoFs and one rotational DoF, with rotational axis orthogonal to plane of translational axes

Library

Joints

Description

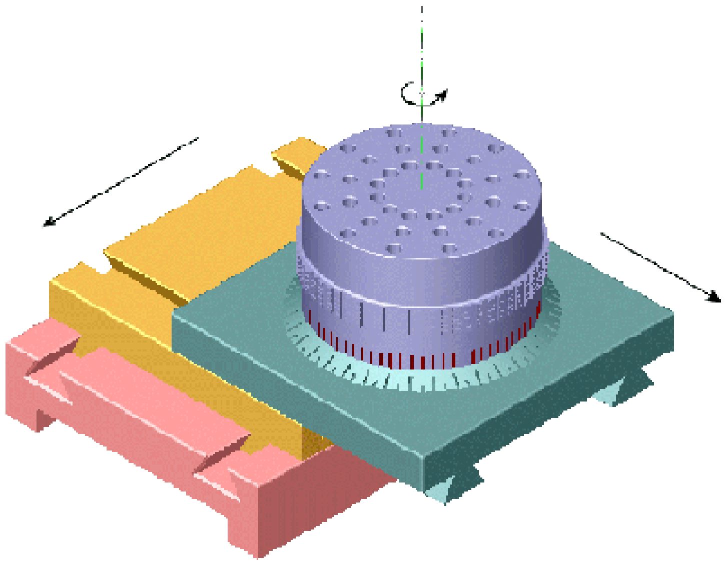
The Planar block represents a composite joint with two translational degrees of freedom (DoFs) as two prismatic primitives and one rotational DoFs as one revolute primitives. The rotation axis must be orthogonal to the plane defined by the two translation axes.

Caution A joint with two prismatic primitives becomes singular if the two translation axes become parallel. The simulation stops with an error in this case.

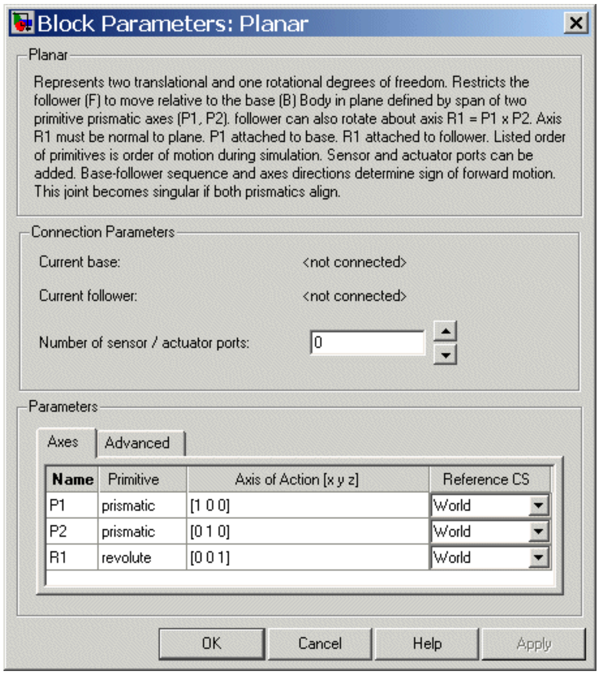
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Planar block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Planar

Represents two translational and one rotational degrees of freedom. Restricts the follower (F) to move relative to the base (B) Body in plane defined by span of two primitive prismatic axes (P1, P2). follower can also rotate about axis R1 = P1 x P2. Axis R1 must be normal to plane. P1 attached to base. R1 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if both prismatics align.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[1 0 0]	World
P2	prismatic	[0 1 0]	World
R1	revolute	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Planar block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Planar Base and Follower Body Connector Ports.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

Current follower

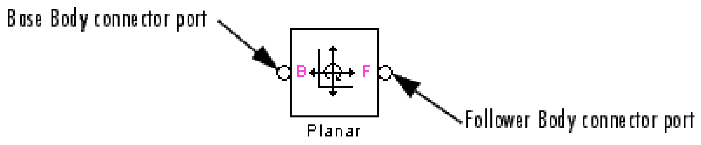
When you connect the follower (F) connector port on the Planar block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Planar Base and Follower Body Connector Ports.

The follower Body is automatically connected to the last joint primitive R1 in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.



Planar Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Planar has an entry line. These lines specify the direction of the axes of action of the DoFs that the Planar represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Planar block: prismatic primitives P1, P2 and revolute primitives R1.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

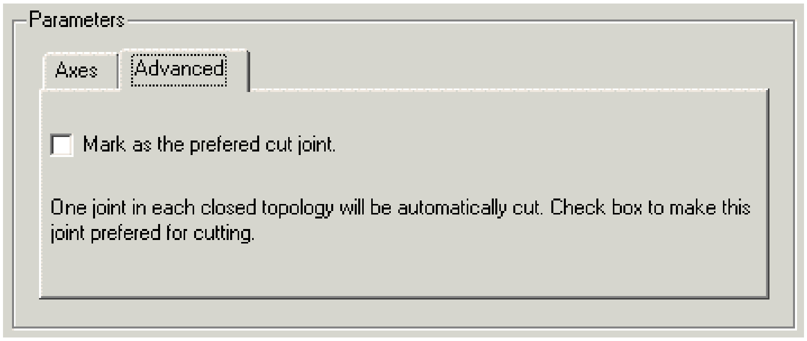
- Prismatic: axis of translation
- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, the two prismatic axes cannot be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

In-Plane, Prismatic, Revolute

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Point-Curve Constraint

Purpose

Constrain motion of point on one body to be along curve on another body

Library

Constraints & Drivers

Description

The two Bodies connected by a Point-Curve Constraint can only move relative to one another if a point on one body moves along a curve on the other body. The point on one body is the origin of the Body coordinate system (CS) to which one side of the Point-Curve Constraint is connected. The corresponding curve starting point on the other body is the origin of the Body CS to which the other side of the Point-Curve Constraint is connected.

Specifying the Curve You specify the curve function on the second body as a *spline* with break points and end conditions. The spline is a piecewise cubic polynomial, with the pieces joined at user-specified *breakpoints:*

$$(x_1,y_1,z_1)\ ,\ (x_2,y_2,z_2)\ ,\ \ldots\ ,\ (x_N,y_N,z_N)$$

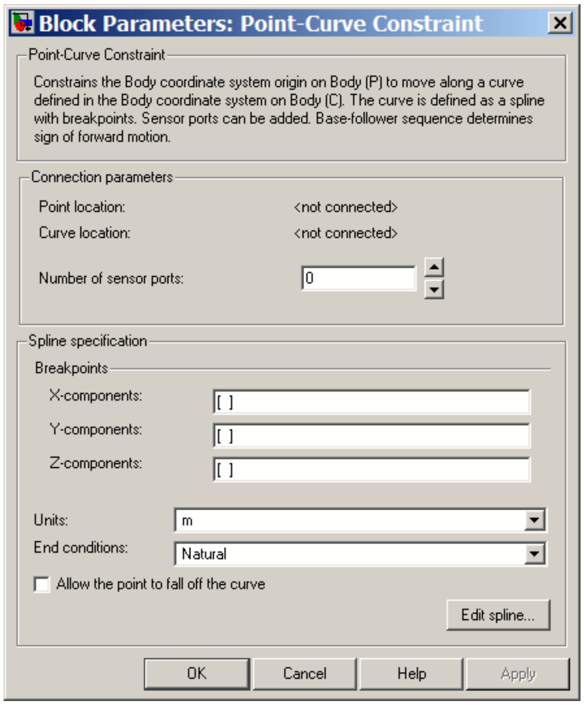
and boundary conditions applied at the spline's *endpoints*, (x_0,y_0,z_0) and $(x_{N+1},y_{N+1},z_{N+1})$. The spline curve and its first two derivatives are continuous at each breakpoint.

Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

For the Point-Curve Constraint, the base (P) is the Body carrying the point, and the follower (C) is the Body carrying the curve. The Point-Curve Constraint is assembled: the Body CS origin on the base (Point) body must be initially collocated with the Body CS origin on the follower (Curve) body, to within assembly tolerance.

You can connect a Constraint & Driver Sensor to any Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

Dialog Box and Parameters


Block Parameters: Point-Curve Constraint

Point-Curve Constraint

Constrains the Body coordinate system origin on Body (P) to move along a curve defined in the Body coordinate system on Body (C). The curve is defined as a spline with breakpoints. Sensor ports can be added. Base-follower sequence determines sign of forward motion.

Connection parameters

Point location: <not connected>

Curve location: <not connected>

Number of sensor ports: 0

Spline specification

Breakpoints

X-components: [ ]

Y-components: [ ]

Z-components: [ ]

Units: m

End conditions: Natural

☐ Allow the point to fall off the curve

The dialog has two active areas, **Connection parameters** and **Spline specification**. It stores the defining information of a single spline for the constraint.

Connection Parameters

The base (P)-follower (C) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

Point location

When you connect the base (P) connector port on the Point-Curve Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Point-Curve Constraint Base and Follower Body Connector Ports.

This Body CS origin is the point of the Point-Curve Constraint.

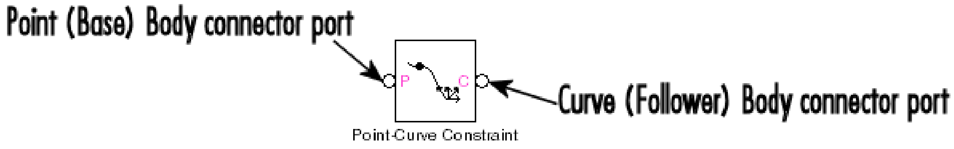
Curve location

When you connect the follower (C) connector port on the Point-Curve Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Point-Curve Constraint Base and Follower Body Connector Ports.

This Body CS origin is the starting point of the curve of the Point-Curve Constraint.

Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is `0`.


Point (Base) Body connector port
P
C
Curve (Follower) Body connector port
Point-Curve Constraint

Point-Curve Constraint Base and Follower Body Connector Ports

Specifying the Spline

The Point-Curve Constraint dialog gives you two ways to specify the spline curve. The first way is entering in this dialog the coordinates of breakpoints and endpoints on the follower and is valid for defining curves in up to three dimensions.

The second way is graphically displaying and editing the spline in the **Edit spline** editor (see following), valid only for two-dimensional curves on the follower.

Breakpoints

List here the *x*-components, *y*-components, and *z*-components, respectively, of the breakpoints and endpoints that define the spline:

X-components: enter $(x_0, x_1, ..., x_{N+1})$ as a vector.

Y-components: enter $(y_0, y_1, ..., y_{N+1})$ as a vector.

Z-components: enter $(z_0, z_1, ..., z_{N+1})$ as a vector.

All three fields require nonnull entries. The number of components in each vector should be the same. *Exception and shortcut:* if all the ***Z*** components are the same, just enter one number in the ***Z*** vector. The **Breakpoints** list replicates this number to expand out a full vector.

If there are no ***X*** and/or ***Y*** components, you must still enter `[0 ... 0]` in that/those field(s). If there are no ***Z*** components, you must still enter at least `[0]` in the ***Z*** field (using the replication/expansion shortcut).

The pull-down menu for each spatial dimension lists the history of those previous breakpoints created by the graphical spline editor (see following) within a single dialog session. Closing the dialog destroys this history, and only the current breakpoint list is retained.

Units

In the pull-down menu, choose the linear units for distances on the constrained bodies. The default is `m` (meters).

End conditions

In the pull-down menu, choose the type of end (boundary) condition on the spline curve. The possible conditions are:

End Condition	Definition	Minimum Number of Points	Notes
`Natural`	Match each endslope to the slope of the cubic that fits the first four points at that end	Two points	Default
`Not-a-knot`	Only the curve and its first derivative are continuous at first and last interior points	Four points	
`Periodic`	Match the first and second derivatives of the two endpoints	Two points (three recommended)	This choice closes the spline by connecting the endpoints

Allow the point to fall off the curve

If the check box is selected, the base point continues with unconstrained motion if it reaches an endpoint and leaves the spline on the follower. The direction of motion at the instant the base point leaves the constraint is tangent to the spline.

If the check box is not selected, and the base point attempts to leave the spline on the follower, the simulation stops with an error. The default is not selected.

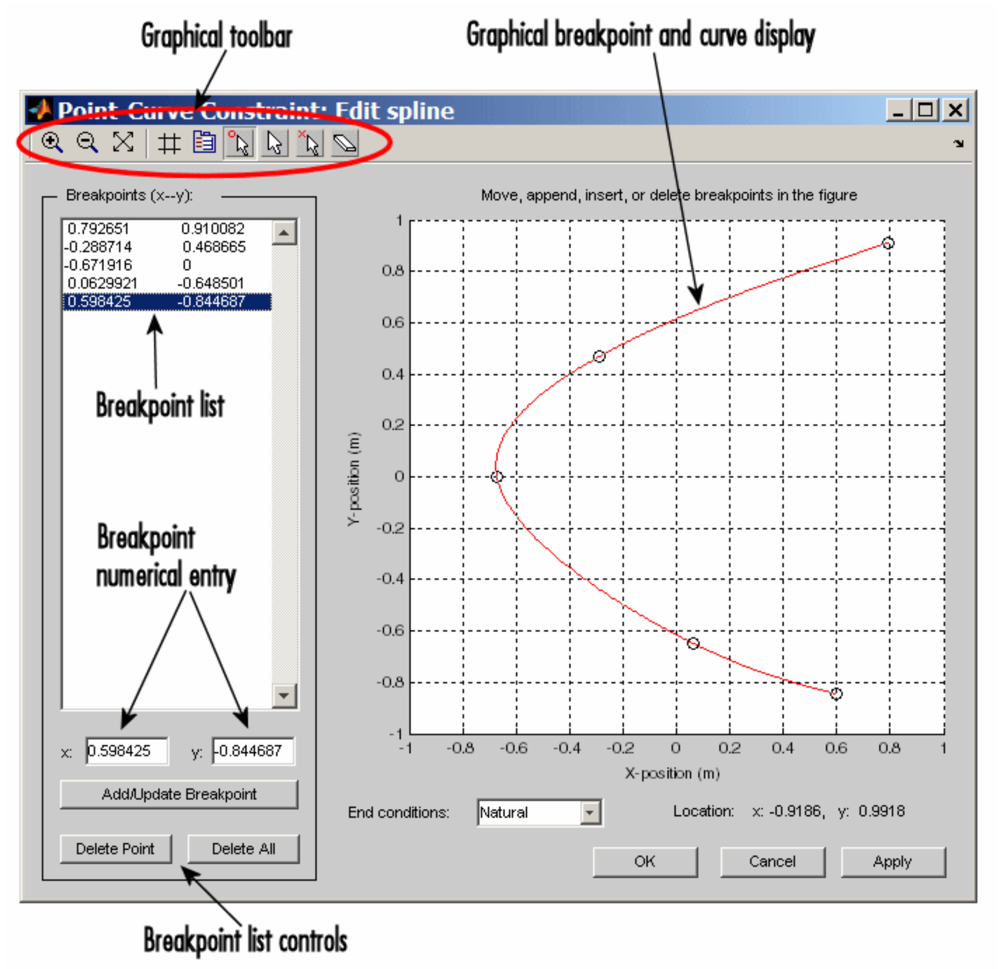
Edit spline...

Click here to open the optional **Edit spline** dialog.

The **Edit spline** dialog provides alternative numerical entry and graphical editing methods for defining the constraint spline. But it can define only two-dimensional curves in the *x*-*y* coordinate directions on the follower Body. The **Edit spline** editor ignores any *z*-components in existing breakpoints.

Edit Spline

The numerical entry area lies on the left side of the **Edit spline** dialog, the graphical editing area on the right side.



0.792651	0.910082
-0.288714	0.468665
-0.671916	0
0.0629921	-0.648501
0.598425	-0.844687

Point-Curve Constraint Spline Editor

Graphical Editing of Spline Points

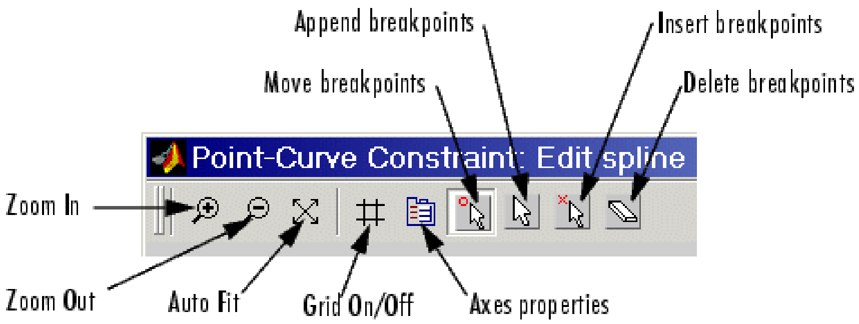
1 To place a breakpoint in the graphical display, place your cursor at the position where you want the breakpoint. The **Location** display

in the lower right indicates your current cursor coordinates in the curve display.

2 Then click at the desired point. A circle appears where you clicked, and simultaneously, the breakpoint is listed in the **Breakpoints (x-y)** list.

Continuing to add breakpoints generates the spline (red curve).

3 Use the Graphical toolbar controls to edit the spline graphically in the display:



- Remove points by clicking on the **Delete breakpoints** icon. Your cursor turns into an eraser symbol. With it, select and click the breakpoints you want to delete.
- Insert new (interior) breakpoints by clicking on the **Insert breakpoints** icon. Your cursor acquires a small circle. Click on the positions, near the existing curve, where you want the new breakpoints. The editor modifies the spline to fit the new breakpoints.
- Add new endpoints and extend the curve by clicking on the **Append breakpoints** icon. Your cursor acquires a small circle. Click on the positions, near the existing endpoints, to where you want to extend the curve. The editor modifies the spline to fit the new endpoints.

- Move existing endpoints by clicking the **Move breakpoints** icon. Click and drag the breakpoints you want to move, then drop them where you want them.

The editor modifies both the spline red curve in the graphical display and the **Breakpoints (x-y)** list as you make these changes.

Additional graphical toolbar controls:

- **Zoom In/Zoom Out** and **Auto Fit**: Standard MATLAB Graphics zooming and auto resizing of graphics display.
- **Axes properties**: Edit properties of graphical display.
- **Grid On/Off**: Turn the graphical display *x*-*y* grid on or off.

Numerical Editing of Spline Points

Use the numerical entry controls, instead of the graphical editing tools, to edit breakpoints by text entry.

Breakpoints (x-y)

You can also add, delete, and edit the breakpoints via this breakpoints list:

- Select an existing breakpoint by highlighting it with your cursor.
- Add a breakpoint by moving the highlighted selection to the empty line below the last breakpoint with your cursor control.
- In the **x:** and **y:** fields, enter the *x*- and *y*-coordinates of the currently selected breakpoint.

Add/Update Breakpoint

After editing an existing breakpoint or entering a new one in the **x:–y:** fields, update the breakpoint list by clicking here.

The new or changed breakpoint appears in the graphical display as a circle.

Delete Point
Click here to delete the currently selected breakpoint.

Delete All
Click here to delete all the breakpoints in the breakpoint list.

End conditions
In the pull-down menu, choose the type of end (boundary) condition on the spline curve. The possible conditions are `natural`, `not-a-knot`, and `periodic`. The default is `natural`.

Closing the Edit Spline Dialog

Clicking **Apply** or **OK** updates the breakpoints stored in the main Point-Curve Constraint dialog.

Previous breakpoint lists are stored in the history pull-down menus of the main Point-Curve Constraint dialog's **Breakpoints** list. This history is destroyed if you close the main dialog, and only the current breakpoint list is retained.

See Also

Constraint & Driver Sensor

See "Modeling Constraints and Drivers" on page 4-38 for more on restricting DoFs with Constraints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on using constraints in closed loops.

See "Constraints and Drivers" on page 10-5.

For more about representing curves as splines, see the Spline Toolbox User's Guide.

Prismatic

Purpose

Represent prismatic joint with one translational degree of freedom

Library

Joints

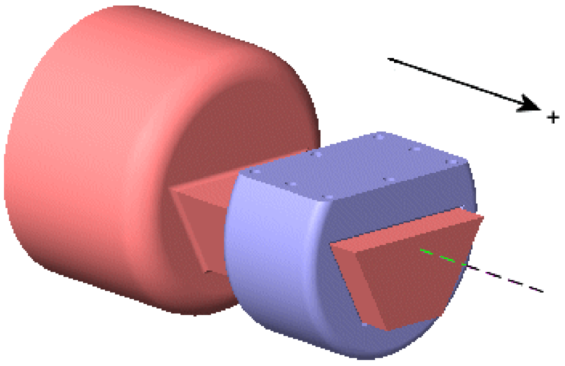
Description

The Prismatic block represents a single translational degrees of freedom (DoF) along a specified axis between two bodies. A prismatic joint is one of SimMechanics primitive joints, along with revolute and spherical.

The Prismatic block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must lie along the prismatic axis, to within assembly tolerances. These Body CS origins do not need to be collocated in space.

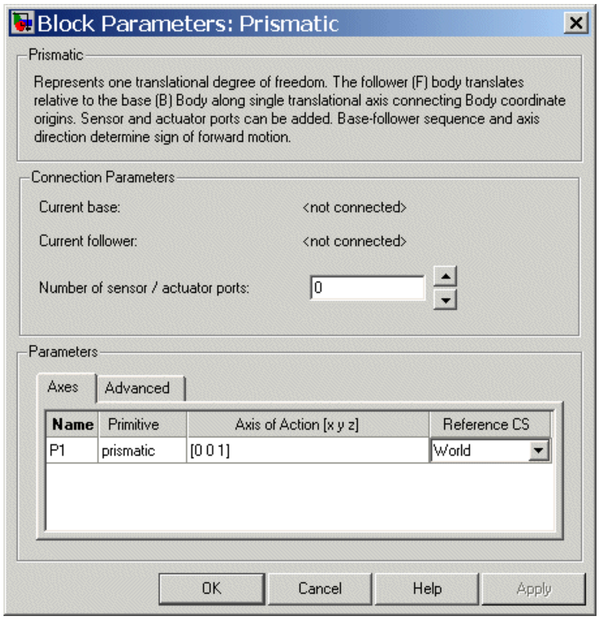
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



Prismatic Motion of Follower (blue) Relative to Base (red)

Dialog Box and Parameters


Block Parameters: Prismatic

Prismatic

Represents one translational degree of freedom. The follower (F) body translates relative to the base (B) Body along single translational axis connecting Body coordinate origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

Current base

When you connect the base (B) connector port on the Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Prismatic Base and Follower Body Connector Ports.

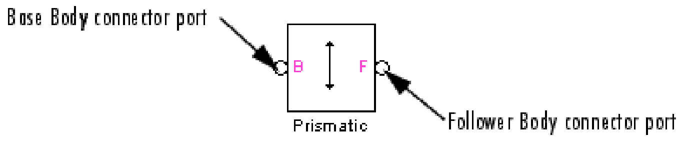
Current follower

When you connect the follower (F) connector port on the Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Prismatic Base and Follower Body Connector Ports.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

The motion of a Prismatic is specified in linear units.

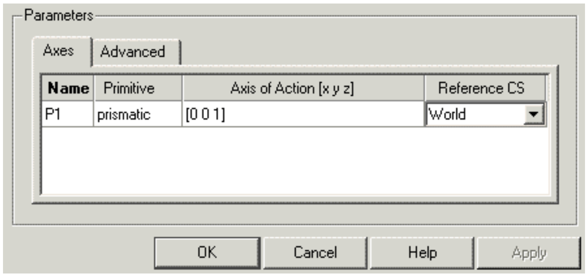


Prismatic Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the translational DoF that the Prismatic represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[0 0 1]	World

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Prismatic, there is only one primitive joint, a prismatic, labeled `P1`.

Primitive

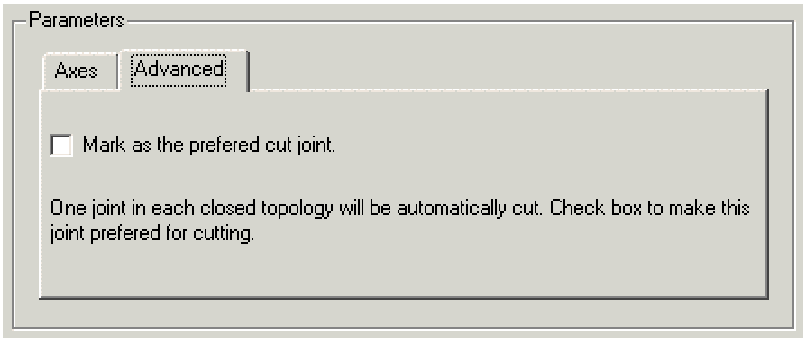
This column automatically displays the type of each primitive joint contained in the Joint block. For Prismatic, there is only one primitive type, labeled `Prismatic`.

Axis of translation [x y z]

Enter here as a three-component vector the directional axis along which this translational DoF can move. The default vector is `[0 0 1]`. The axis is a directed vector whose overall sign matters.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of translation is oriented with respect to. This CS also determines the absolute meaning of force and motion along the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Disassembled Prismatic, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Revolute, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

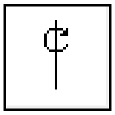
Purpose

Represent assembled revolute joint with one rotational degree of freedom

Library

Joints

Description

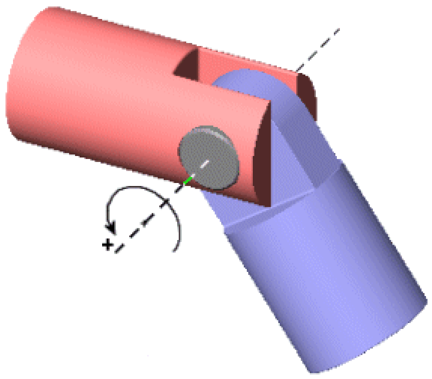


The Revolute block represents a single rotational degrees of freedom (DoF) about a specified axis between two bodies. The rotational sense is defined by the right-hand rule. A revolute joint is one of SimMechanics primitive joints, along with prismatic and spherical.

The Revolute block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, to within assembly tolerances.

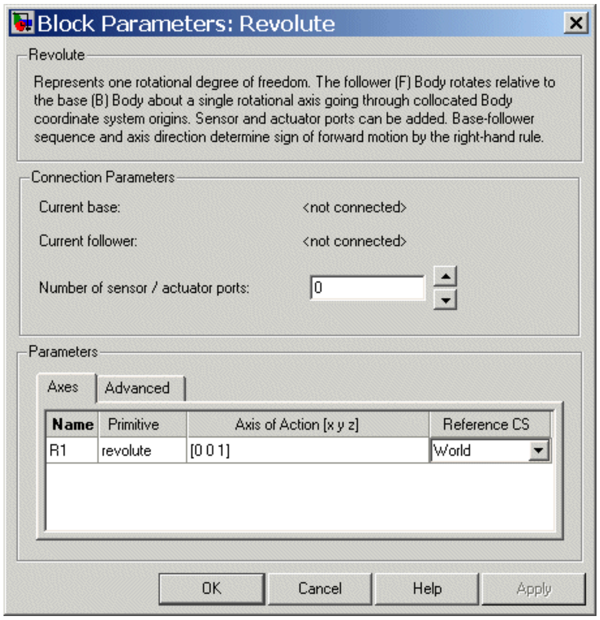
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



Revolute Motion of Follower (blue) Relative to Base (red)

Dialog Box and Parameters


Block Parameters: Revolute

Revolute

Represents one rotational degree of freedom. The follower (F) Body rotates relative to the base (B) Body about a single rotational axis going through collocated Body coordinate system origins. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion by the right-hand rule.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-hand rule about the rotation axis.

Current base

When you connect the base (B) connector port on the Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute Base and Follower Body Connector Ports.

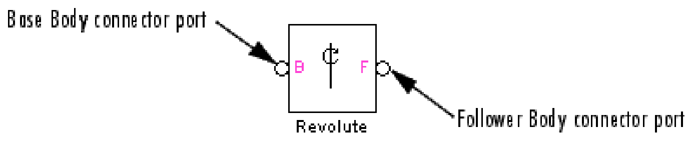
Current follower

When you connect the follower (F) connector port on the Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute Base and Follower Body Connector Ports.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

The motion of a Revolute is specified in angular units.

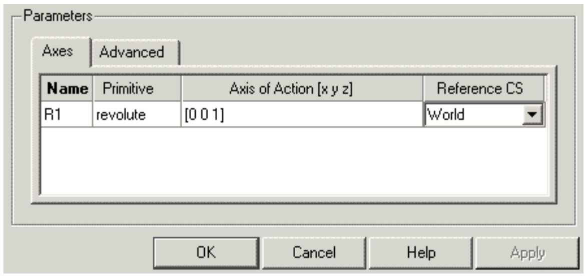


Revolute Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axis of the DoF that the Revolute represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute, there is only one primitive joint, a revolute, labeled `R1`.

Primitive

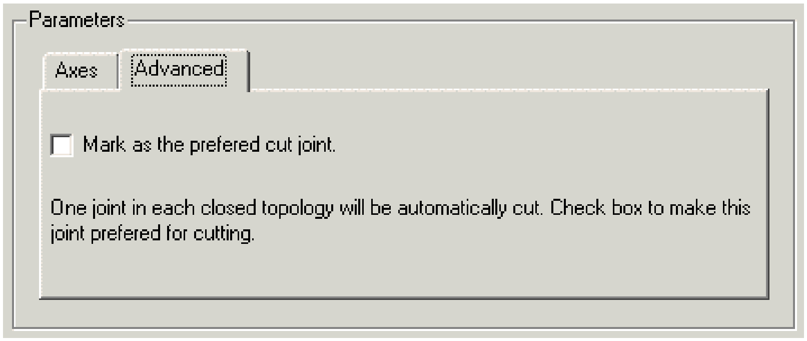
This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute, there is only one primitive type, labeled `Revolute`.

Axis of rotation [x y z]

Enter here as a three-component vector the directional axis about which this rotational DoF can move. The default vector is `[0 0 1]`. The axis is a directed vector whose overall sign matters.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of rotation is oriented with respect to. This CS also determines the absolute meaning of torque and motion about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Disassembled Revolute, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Prismatic, Spherical

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Revolute-Revolute

Purpose

Represent composite joint composed of two revolute primitives spatially separated by massless connector of constant length

Library

Joints/Massless Connectors

Description

The Revolute-Revolute block represents a composite joint composed of two revolute joint primitives. The Body coordinate systems (CSs) on either side of the Joint are each connected to a revolute primitive. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both revolute primitives are assembled.

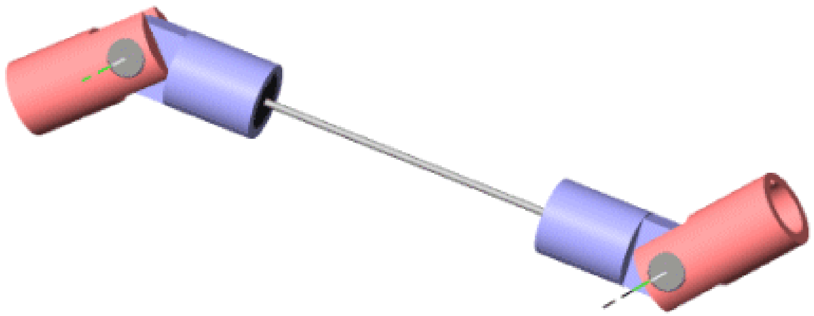
Caution This joint becomes singular if the two revolute primitive axes align with the vector separating the primitives. The simulation stops with an error in this case.

You specify the two revolute axes of these two joint primitives in the dialog. The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

You cannot connect an Actuator or Sensor to a Massless Connector.

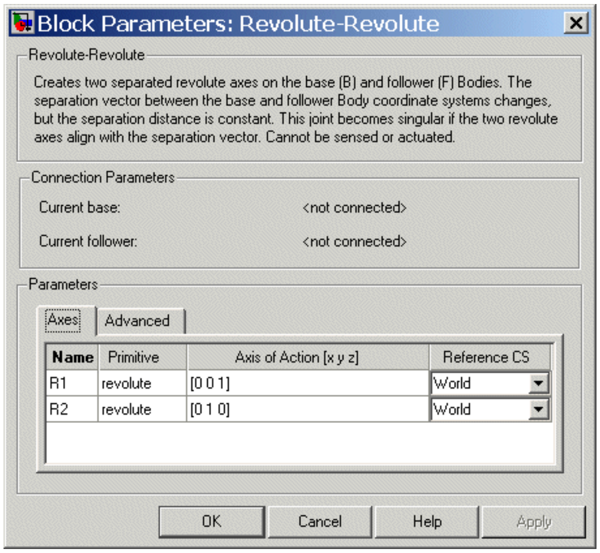
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the directions of the joint axes.



Massless Connector Between Revolute and Revolute Joints

Dialog Box and Parameters



Block Parameters: Revolute-Revolute

Revolute-Revolute

Creates two separated revolute axes on the base (B) and follower (F) Bodies. The separation vector between the base and follower Body coordinate systems changes, but the separation distance is constant. This joint becomes singular if the two revolute axes align with the separation vector. Cannot be sensed or actuated.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World
R2	revolute	[0 1 0]	World

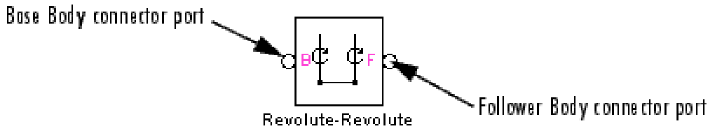
The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base or follower rotating in the right-handed sense about its respective rotation axis.

Current base
: When you connect the base (B) connector port on the Revolute-Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute-Revolute Base and Follower Body Connector Ports.

Current follower
: When you connect the follower (F) connector port on the Revolute-Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute-Revolute Base and Follower Body Connector Ports.

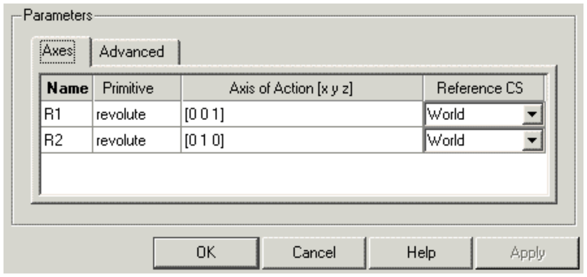


Revolute-Revolute Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axes of these DoFs that the Revolute-Revolute represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World
R2	revolute	[0 1 0]	World

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute-Revolute, there are two revolute primitives, labeled `R1` and `R2`, connecting to base and follower, respectively.

Primitive

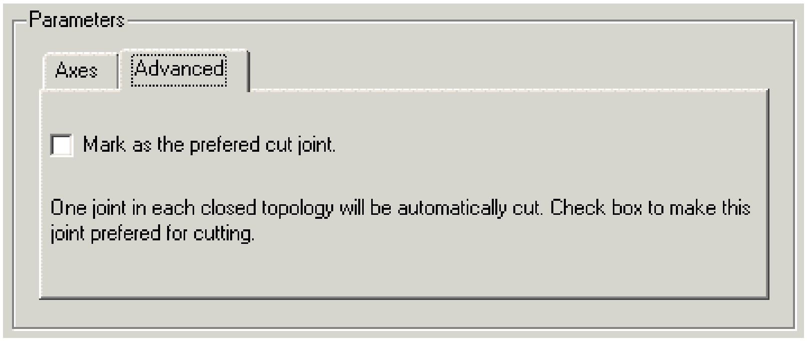
This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute-Revolute, there is only one primitive type, labeled `Revolute`.

Axis of Action [x y z]

Enter here as a three-component vector the directional axis about which these rotational DoFs can move. The default vectors are `[0 0 1]` and `[0 1 0]`. The axes are directed vectors whose overall signs matter.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axes of rotation are oriented with respect to. These CSs also determine the absolute meaning of torque and motion about the primitive axes. The defaults are `World`.


Parameters

Axes | Advanced

☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Revolute

See “Modeling Joints” on page 4-20 for more on representing DoFs with Massless Connectors.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Revolute-Spherical

Purpose

Represent composite joint composed of revolute and spherical primitives spatially separated by massless connector of constant length

Library

Joints/Massless Connectors

Description

The Revolute-Spherical block represents a composite joint composed of a revolute and a spherical joint primitive. The base Body coordinate system (CS) on one side of the Joint is connected to the revolute primitive, and the follower Body CS is connected to the spherical primitive. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both primitives are assembled.

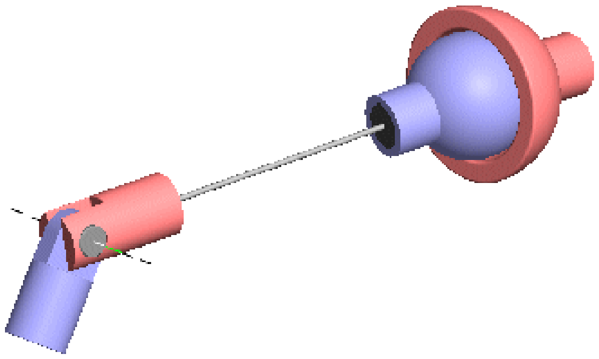
Caution This joint becomes singular if the revolute primitive axis aligns with the vector separating the primitives. The simulation stops with an error in this case.

You specify the revolute axis of the revolute joint primitives in the dialog. The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

You cannot connect an Actuator or Sensor to a Massless Connector.

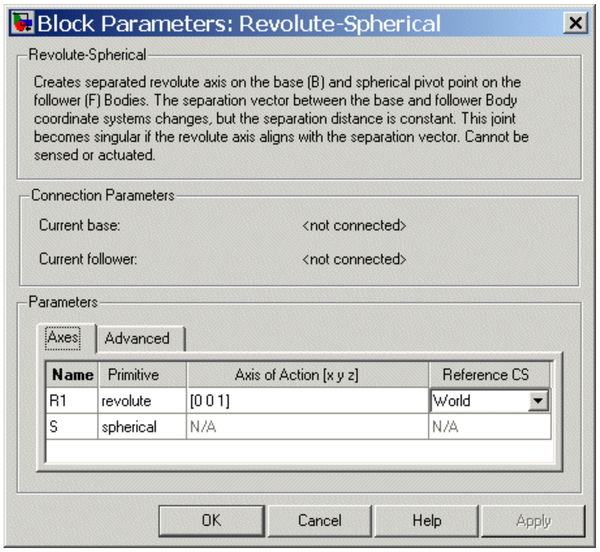
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



Massless Connector Between Revolute and Spherical Joints

Dialog Box and Parameters



Revolute-Spherical

Creates separated revolute axis on the base (B) and spherical pivot point on the follower (F) Bodies. The separation vector between the base and follower Body coordinate systems changes, but the separation distance is constant. This joint becomes singular if the revolute axis aligns with the separation vector. Cannot be sensed or actuated.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World
S	spherical	N/A	N/A

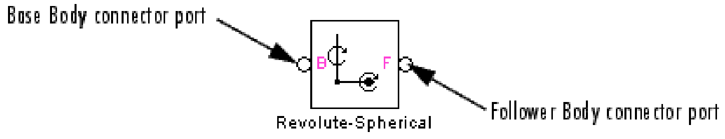
The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base rotating in the right-handed sense about its rotation axis or the follower pivoting as shown for the Spherical Joint.

Current base
: When you connect the base (B) connector port on the Revolute-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute-Spherical Base and Follower Body Connector Ports.

Current follower
: When you connect the follower (F) connector port on the Revolute-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Revolute-Spherical Base and Follower Body Connector Ports.

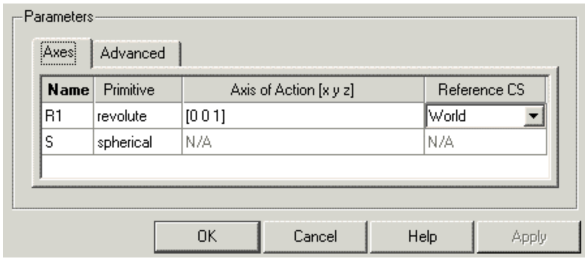


Revolute-Spherical Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axis of one of the DoFs that Revolute-Spherical represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World
S	spherical	N/A	N/A

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute-Spherical, there are one revolute and one spherical primitive, labeled `R1` and `S`, connecting to base and follower, respectively.

Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute-Spherical, there are two primitive types, labeled `Revolute` and `Spherical`.

Axis of Action [x y z]

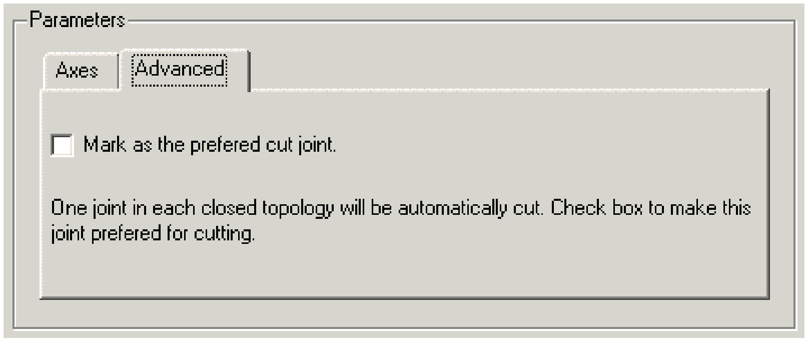
Enter here as a three-component vector the directional axis about which the rotational DoF can move. The default vector is `[0 0 1]`. The axis is a directed vector whose overall sign matters.

This field is not active for the Spherical primitive.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of rotation is oriented with respect to. This CS also determines the absolute meaning of torque and motion about the primitive axis. The default is `World`.

This field is not active for the Spherical primitive.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Revolute, Spherical

See "Modeling Joints" on page 4-20 for more on representing DoFs with Massless Connectors.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

RotationMatrix2VR

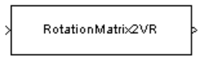
Purpose

Convert 3-by-3 rotation matrix to equivalent VRML form of rotation axis and angle

Library

Utilities

Description

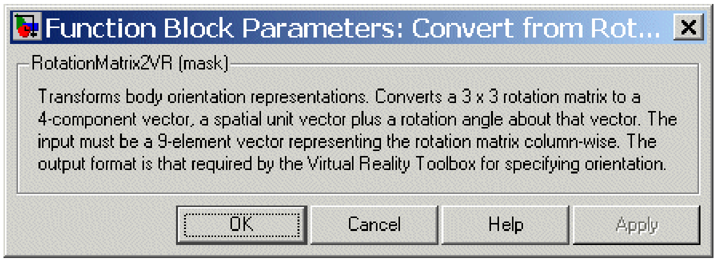


A rotation with respect to an initial orientation has many equivalent representations. A common and important one is the 3-by-3 orthogonal rotation matrix R, where $R^{-1} = R^T$ and $R^T R = RR^T = I$, the 3-by-3 identity matrix. Another important representation is the combination of rotation axis (a unit vector **n**) and angle of rotation θ about that axis. The sign of rotation follows the right-hand-rule.

The RotationMatrix2VR block converts the 3-by-3 rotation matrix representation of orientation to its equivalent representation as a rotation axis and angle about that axis, the form used in Virtual Reality Modeling Language (VRML) for orienting bodies. The input and output signals are bundled Simulink signals.

The most common use of rotations is to represent the orientation of a body with respect to some coordinate system (CS) axes.

Dialog Box and Parameters



Transforms body orientation representations. Converts a 3 x 3 rotation matrix to a 4-component vector, a spatial unit vector plus a rotation angle about that vector. The input must be a 9-element vector representing the rotation matrix column-wise. The output format is that required by the Virtual Reality Toolbox for specifying orientation.

The dialog has no active areas.

Representations of Rotation Signals

The rotation matrix R has the form:

$$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

The input signal to the RotationMatrix2VR block is the R matrix components passed column-wise and bundled into a single 9-component Simulink signal: `[R11 R21 R31 R12 ...]`.

The output signal is the equivalent rotation represented as the axis of rotation, a unit vector $\boldsymbol{n} = (n_x,n_y,n_z)$, with

$$\boldsymbol{n}\cdot\boldsymbol{n} = n_x^2 + n_y^2 + n_z^2 = 1,$$

and the angle of rotation θ about that axis. The sign of the rotation follows the right-hand rule. The output signal is bundled into a single 4-component Simulink signal:

$[n_x\ n_y\ n_z\ \theta]$.

See Also

Body

See "Body Motion in SimMechanics" on page 3-4 and "How SimMechanics Represents Body Orientation" on page 3-11 for more details on representing body rotations.

See entries on **axis-angle rotation**, **Euler angles**, **quaternion**, and **rotation matrix** in the Glossary for summaries of body orientation representations.

For more on virtual reality and VRML, see the "Virtual Reality Toolbox User's Guide".

Screw

Purpose

Represent composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes and linear pitch constraint between translational and rotational motion

Library

Joints

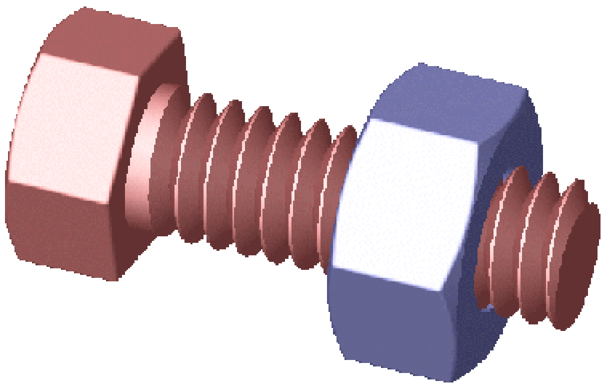
Description

The Screw block represents a composite joint with one translational degrees of freedom (DoF) as one prismatic primitive and one rotational DoF as one revolute primitive. The translation and rotation axes are parallel. The translational and rotational DoFs are constrained by a pitch constraint to have proportional motion.

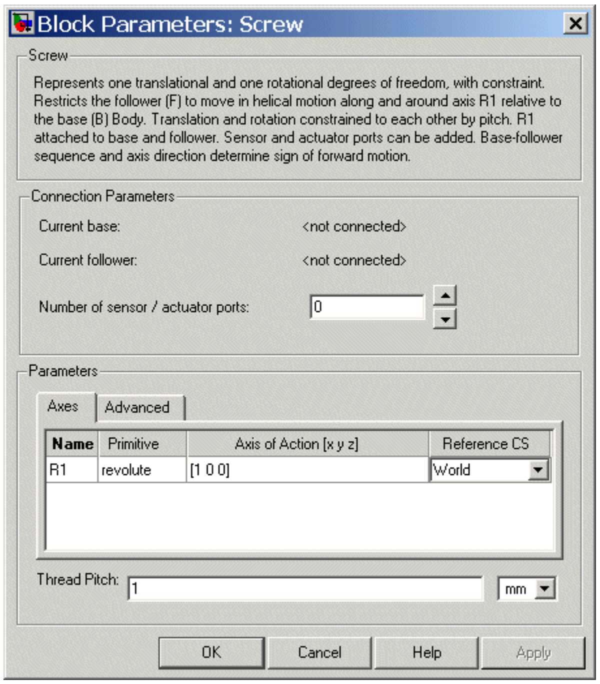
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Screw block is assembled: the origins of these Body CSs must lie along the primitive axes. But the Body CS origins on either side of the Joint do not need to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



Dialog Box and Parameters



Screw

Represents one translational and one rotational degrees of freedom, with constraint. Restricts the follower (F) to move in helical motion along and around axis R1 relative to the base (B) Body. Translation and rotation constrained to each other by pitch. R1 attached to base and follower. Sensor and actuator ports can be added. Base-follower sequence and axis direction determine sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[1 0 0]	World

Thread Pitch: 1 mm

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base
: When you connect the base (B) connector port on the Screw block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Screw Base and Follower Body Connector Ports.

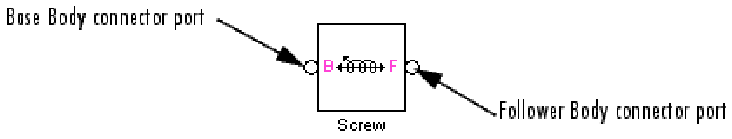
  The base Body is automatically connected to the joint primitive `R1` in the primitive list in **Parameters**.

Current follower
: When you connect the follower (F) connector port on the Screw block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Screw Base and Follower Body Connector Ports.

  The follower Body is automatically connected to the joint primitive `R1` in the primitive list in **Parameters**.

Number of sensor/actuator ports
: Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

  The motion of revolute primitives is specified in angular units.



Screw Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Screw has an entry line. These lines specify the direction of the axes of action of the DoFs that the Screw represents.

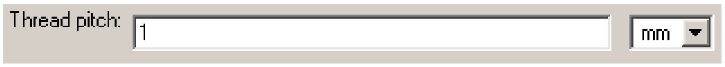
Name - Primitive
: The primitive list states the name and type of the joint primitive that makes up the Screw block: revolute primitive `R1`.

Axis of Action [x y z]
: Enter here as a three-component vector the directional axes defining the allowed motions of this primitive and its corresponding DoF:

- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

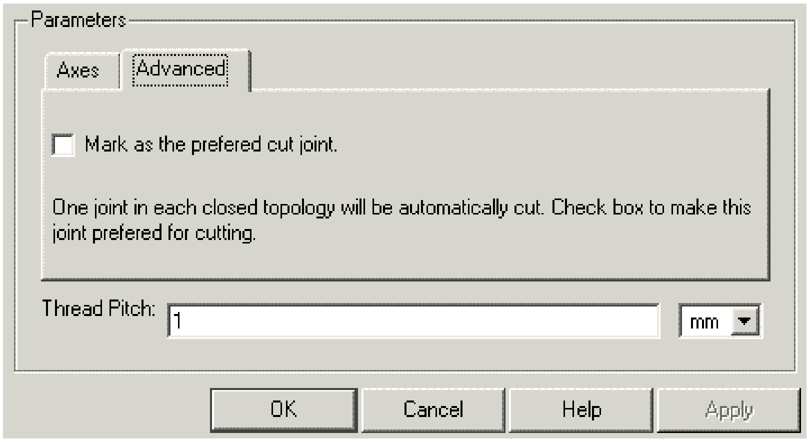
Reference CS
: Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.


Thread pitch: 1 mm

The thread pitch controls the amount of translation for each turn of the screw.

Thread pitch
: Linear distance the screw travels along the screw axis for each complete revolution of 2π radians (360°). The default is `1`.

In pull-down menu, select units. The default is `mm` (millimeters).



Parameters

Axes | Advanced

☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

Thread Pitch: 1 mm

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Cylindrical, Prismatic, Revolute

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

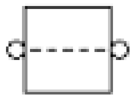
Purpose

Connect two mechanical components so that they share same mechanical environment

Library

Bodies

Description



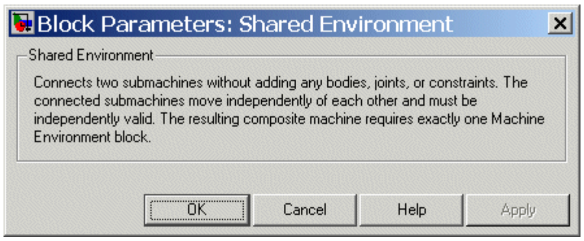
The Shared Environment block provides a nonphysical connection between two independent mechanical block diagrams, or submachines. The block carries no inertia, adds no joints, imposes no constraints, and transfers no motion, force, or torque between the SimMechanics blocks to which it is connected.

You can use this block to connect two independent machines into one machine, so that the two submachines then share the same machine environment. Making this connection does not change the structure or dynamics of either submachine.

Caution The two connected submachines have to be independently valid, and *each* submachine requires at least one Ground block.

The resulting composite machine needs exactly one Machine Environment block, not two.

Dialog Box and Parameters



Block Parameters: Shared Environment

Shared Environment

Connects two submachines without adding any bodies, joints, or constraints. The connected submachines move independently of each other and must be independently valid. The resulting composite machine requires exactly one Machine Environment block.

This block has no parameters.

See Also

Ground, Machine Environment

Six-DoF

Purpose

Represent composite joint with three translational and three rotational DoFs

Library

Joints

Description

The Six-DoF block represents a composite joint with three translational degrees of freedom (DoFs) as three prismatic primitives and three rotational DoFs as one spherical primitives. There are no constraints among the primitives. Unlike Bushing, Six-DoF represents the rotational DoFs as one spherical, rather than as three revolutes.

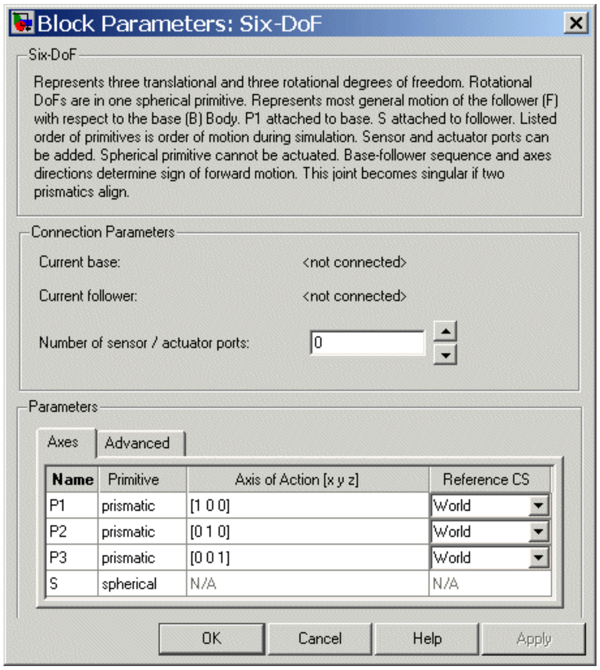
Caution A joint with three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with as error in this case.

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Six-DoF block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.

Dialog Box and Parameters



Six-DoF

Represents three translational and three rotational degrees of freedom. Rotational DoFs are in one spherical primitive. Represents most general motion of the follower (F) with respect to the base (B) Body. P1 attached to base. S attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Spherical primitive cannot be actuated. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if two prismatics align.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
P1	prismatic	[1 0 0]	World
P2	prismatic	[0 1 0]	World
P3	prismatic	[0 0 1]	World
S	spherical	N/A	N/A

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive spherical motion is the follower rotating in the right-handed sense as shown in the Spherical block figure.

Current base

When you connect the base (B) connector port on the Six-DoF block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Six-DoF Base and Follower Body Connector Ports.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

Current follower

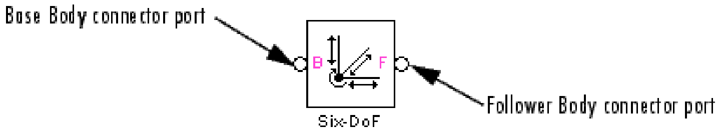
When you connect the follower (F) connector port on the Six-DoF block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Six-DoF Base and Follower Body Connector Ports.

The follower Body is automatically connected to the last joint primitive S in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of prismatic primitives is specified in linear units. The motion of spherical primitives is specified by a dimensionless quaternion.



Six-DoF Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Six-DoF has an entry line. These lines specify the direction of the axes of action of the DoFs that the Six-DoF represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Six-DoF block: prismatic primitives P1, P2, P3, and spherical primitive S.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

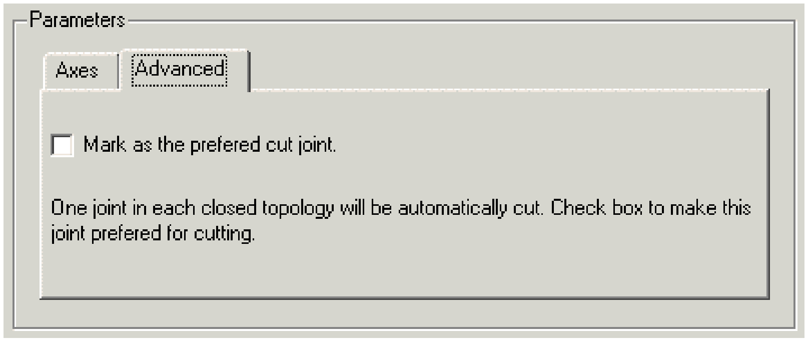
- Prismatic: axis of translation
- Spherical: field is not active

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the prismatic axes can be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Bushing, Gimbal, Prismatic, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

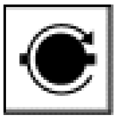
Purpose

Represent assembled spherical joint with three rotational degrees of freedom

Library

Joints

Description



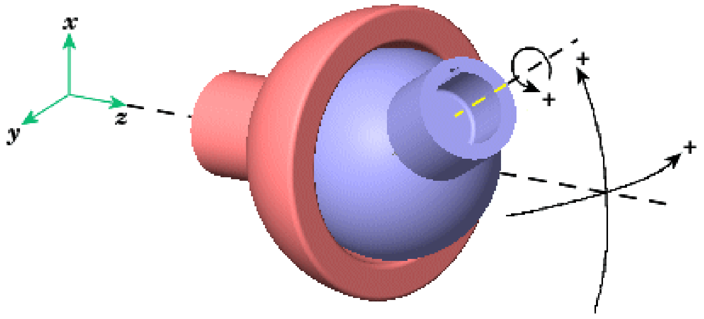
The Spherical block represents three rotational degrees of freedom (DoFs) at a single pivot point, a "ball-in-socket" joint. Two rotational DoFs specify a directional axis, and a third rotational DoF specifies rotation about that directional axis. The sense of each rotational DoF is defined by the right-hand rule, and the three rotations together form a right-handed system. A spherical joint is one of the SimMechanics primitive joints, along with prismatic and revolute.

The Spherical block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, within assembly tolerances.

You cannot connect an Actuator to a Spherical. Unlike the Gimbal block, the Spherical block cannot become singular.

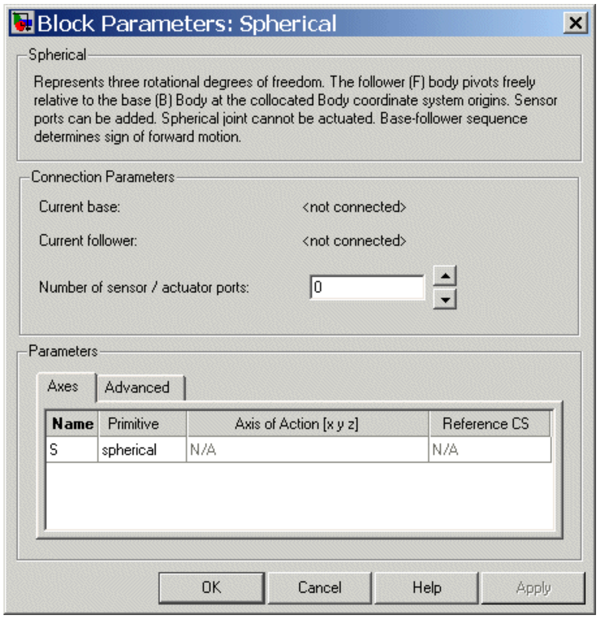
You can connect all Joint blocks to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

Any Joint block represents only the abstract relative motion of two bodies, not the bodies themselves.



Spherical Motion of Follower (blue) Relative to Base (red)

Dialog Box and Parameters



Spherical

Represents three rotational degrees of freedom. The follower (F) body pivots freely relative to the base (B) Body at the collocated Body coordinate system origins. Sensor ports can be added. Spherical joint cannot be actuated. Base-follower sequence determines sign of forward motion.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
S	spherical	N/A	N/A

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

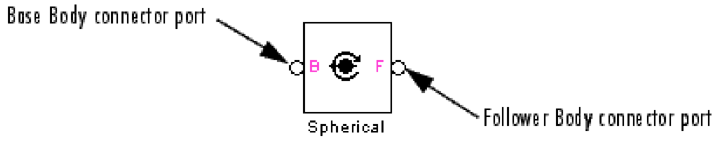
The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-hand sense as shown in the figure above.

Current base
When you connect the base (B) connector port on the Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Spherical Base and Follower Body Connector Ports.

Current follower
When you connect the follower (F) connector port on the Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Spherical Base and Follower Body Connector Ports.

Number of sensor/actuator ports
Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Sensor blocks to this Joint. The default is `0`. A Spherical cannot be connected to a Joint Actuator.

The motion of a Spherical is three DoFs specified in quaternion form.

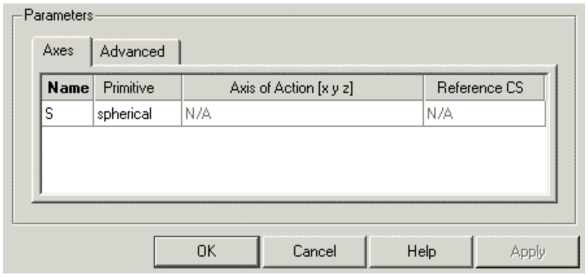


Spherical Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are automatic. They specify the orientation of the spherical DoF that the Spherical represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
S	spherical	N/A	N/A

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Spherical, there is only one primitive joint, a spherical, labeled `S`.

Primitive

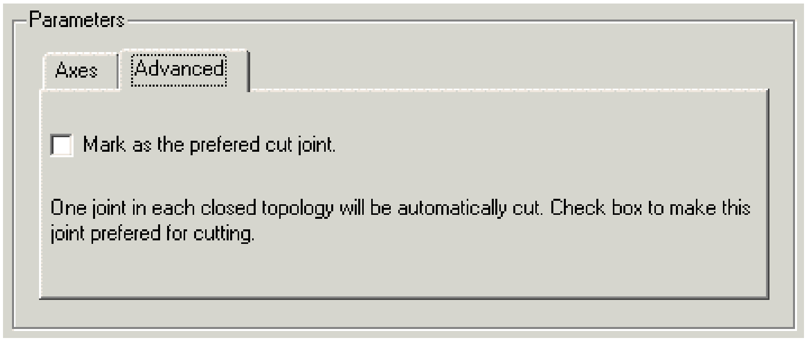
This column automatically displays the type of each primitive joint contained in the Joint block. For Spherical, there is only one primitive type, labeled `Spherical`.

Reference orientation [x y z]

This field is not active.

Reference CS

This field is not active.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Disassembled Spherical, Gimbal, Joint Sensor, Prismatic, Revolute

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Spherical-Spherical

Purpose

Represent composite joint composed of two spherical primitives spatially separated by massless connector of constant length

Library

Joints/Massless Connectors

Description

The Spherical-Spherical block represents a composite joint composed of two spherical joint primitives. The Body coordinate system (CSs) on either side of the Joint are connected to the spherical primitives. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both primitives are assembled.

The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

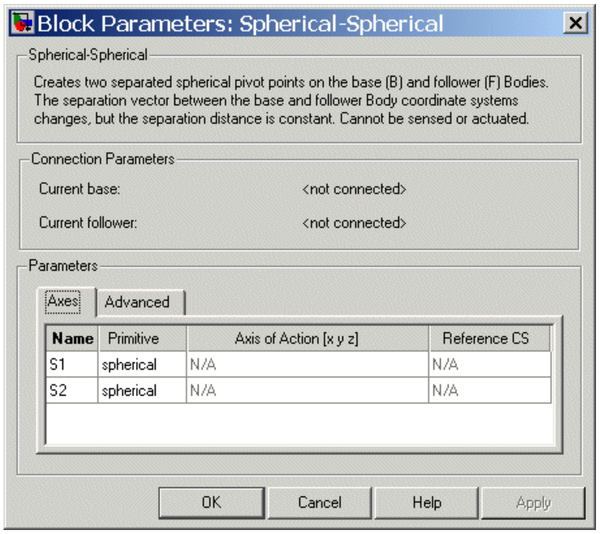
You cannot connect an Actuator or Sensor to a Massless Connector.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the directions of the joint axes.

Massless Connector Between Spherical and Spherical Joints

Dialog Box and Parameters



Block Parameters: Spherical-Spherical

Spherical-Spherical

Creates two separated spherical pivot points on the base (B) and follower (F) Bodies. The separation vector between the base and follower Body coordinate systems changes, but the separation distance is constant. Cannot be sensed or actuated.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
S1	spherical	N/A	N/A
S2	spherical	N/A	N/A

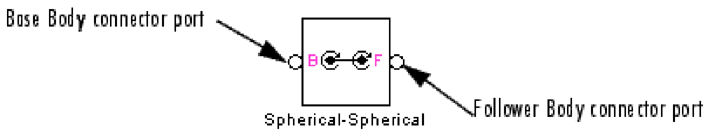
The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base or follower pivoting as shown by the motion figure in the Spherical block reference page.

Current base
: When you connect the base (B) connector port on the Spherical-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Spherical-Spherical Base and Follower Body Connector Ports.

Current follower
: When you connect the follower (F) connector port on the Spherical-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Spherical-Spherical Base and Follower Body Connector Ports.

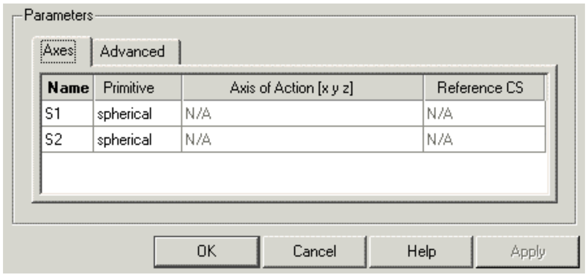


Spherical-Spherical Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are automatic. They specify the orientation of the spherical DoFs that the Spherical-Spherical represents.



Name	Primitive	Axis of Action [x y z]	Reference CS
S1	spherical	N/A	N/A
S2	spherical	N/A	N/A

Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Spherical-Spherical, there are two spherical primitives, labeled `S1` and `S2`, connecting to base and follower, respectively.

Primitive

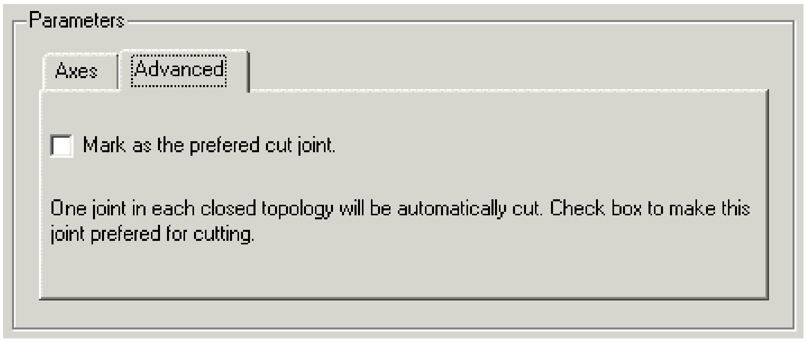
This column automatically displays the type of each primitive joint contained in the Joint block. For Spherical-Spherical, there is only one primitive type, labeled `Spherical`.

Axis of Action [x y z]

These fields are not active.

Reference CS

These fields are not active.


Parameters

Axes | Advanced

☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Massless Connectors.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

Purpose

Represent composite joint with one translational and three rotational DoFs

Library

Joints

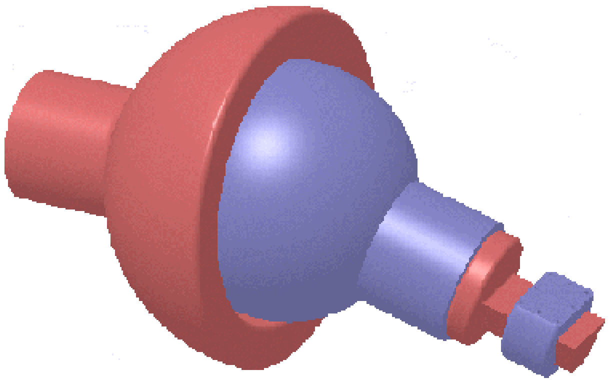
Description

The Telescoping block represents a composite joint with one translational degrees of freedom (DoF) as one prismatic primitive and three rotational DoFs as one spherical primitive. There are no constraints among the primitives. Unlike Bearing, Telescoping represents the rotational DoFs as one spherical, rather than as three revolutes.

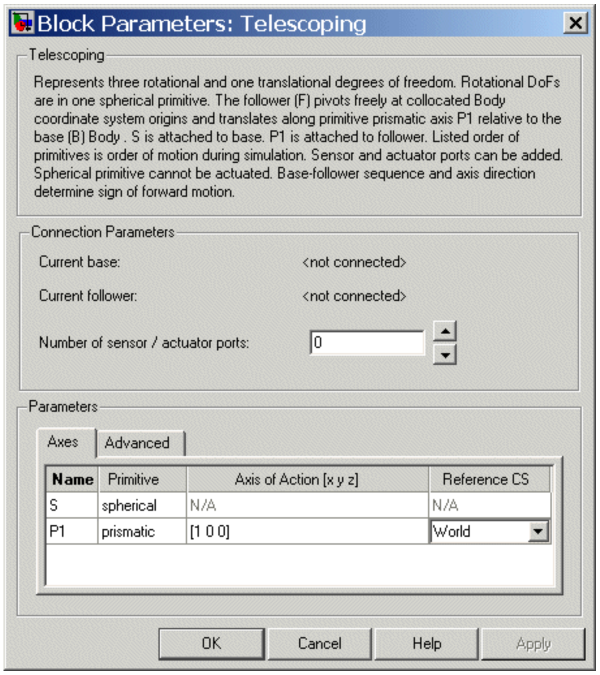
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Telescoping block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



Dialog Box and Parameters



Telescoping

Represents three rotational and one translational degrees of freedom. Rotational DoFs are in one spherical primitive. The follower (F) pivots freely at collocated Body coordinate system origins and translates along primitive prismatic axis P1 relative to the base (B) Body . S is attached to base. P1 is attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Spherical primitive cannot be actuated. Base-follower sequence and axis direction determine sign of forward motion.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
S	spherical	N/A	N/A
P1	prismatic	[1 0 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive spherical motion is the follower rotating in the right-handed sense as shown in the Spherical block figure.

Current base

When you connect the base (B) connector port on the Telescoping block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Telescoping Base and Follower Body Connector Ports.

The base Body is automatically connected to the first joint primitive `S` in the primitive list in **Parameters**.

Current follower

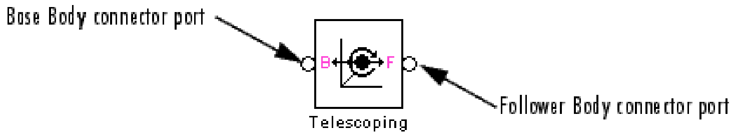
When you connect the follower (F) connector port on the Telescoping block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Telescoping Base and Follower Body Connector Ports.

The follower Body is automatically connected to the last joint primitive `P1` in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

The motion of prismatic primitives is specified in linear units. The motion of spherical primitives is specified by a dimensionless quaternion.



Telescoping Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Telescoping has an entry line. These lines specify the direction of the axes of action of the DoFs that Telescoping represents.

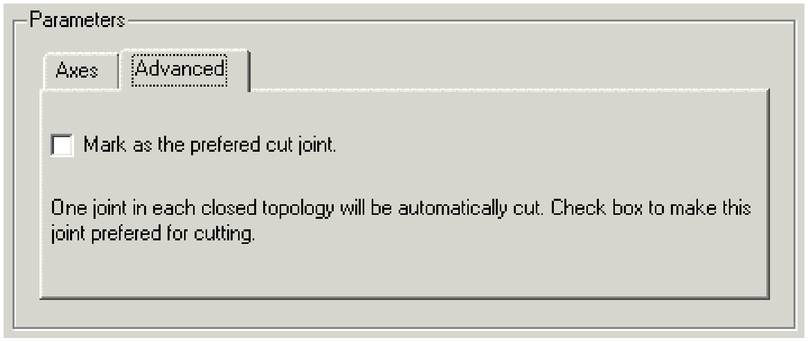
Name - Primitive
The primitive list states the names and types of joint primitives that make up the Telescoping block: spherical primitive `S` and prismatic primitives `P1`.

Axis of Action [x y z]
Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation
- Spherical: field is not active

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

Reference CS
Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Bearing, Prismatic, Six-DoF, Spherical

See “Modeling Joints” on page 4-20 for more on representing DoFs with Joints.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on closed loops and cutting.

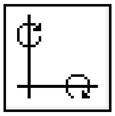
Purpose

Represent composite joint with two rotational DoFs

Library

Joints

Description



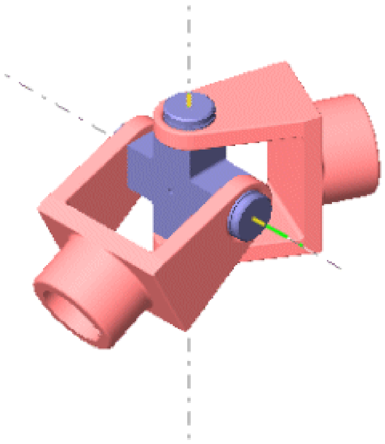
The Universal block represents a composite joint with two rotational degrees of freedom (DoFs) as two revolute primitives. There are no constraints among the primitives.

Caution A joint with two revolute primitives becomes singular if the two rotation axes become parallel (“gimbal lock”). The simulation stops with an error in this case.

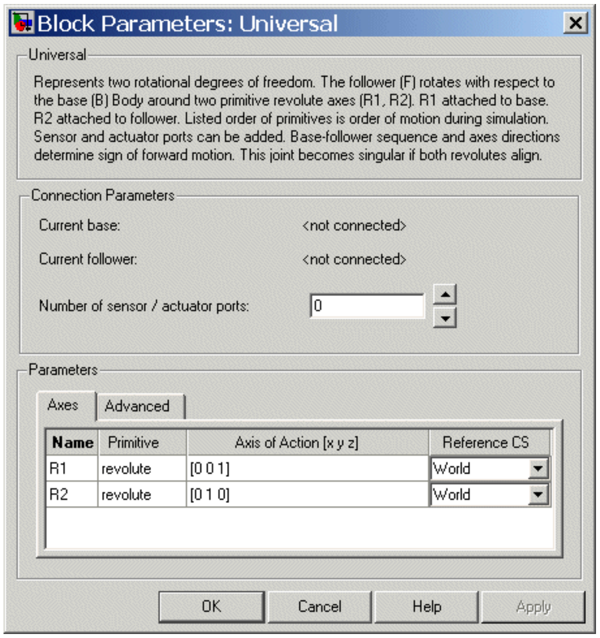
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Universal block is assembled: the origins of these Body CSs must be spatially collocated points, within assembly tolerances

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



Dialog Box and Parameters



Block Parameters: Universal

Universal

Represents two rotational degrees of freedom. The follower (F) rotates with respect to the base (B) Body around two primitive revolute axes (R1, R2). R1 attached to base. R2 attached to follower. Listed order of primitives is order of motion during simulation. Sensor and actuator ports can be added. Base-follower sequence and axes directions determine sign of forward motion. This joint becomes singular if both revolutes align.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Axes | Advanced

Name	Primitive	Axis of Action [x y z]	Reference CS
R1	revolute	[0 0 1]	World
R2	revolute	[0 1 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.

Current base

When you connect the base (B) connector port on the Universal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Universal Base and Follower Body Connector Ports.

The base Body is automatically connected to the first joint primitive `R1` in the primitive list in **Parameters**.

Current follower

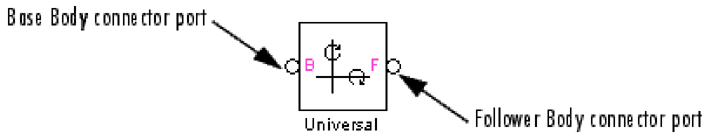
When you connect the follower (F) connector port on the Universal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Universal Base and Follower Body Connector Ports.

The follower Body is automatically connected to the last joint primitive `R2` in the primitive list in **Parameters**.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is `0`.

The motion of revolute primitives is specified in angular units.



Universal Base and Follower Body Connector Ports

Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Universal has an entry line. These lines specify the direction of the axes of action of the DoFs that the Universal represents.

Name - Primitive

The primitive list states the names and types of joint primitives that make up the Universal block: revolute primitives `R1`, `R2`.

Axis of Action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

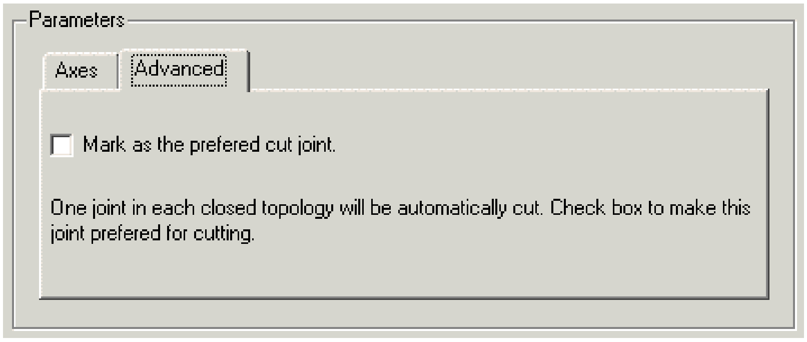
- Revolute: axis of rotation

The default vectors are shown in the dialog above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, the two revolute axes cannot be parallel.

Reference CS

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.


Parameters

☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Gimbal, Revolute

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

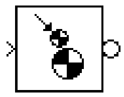
Purpose

Vary mass and inertia on body at specific body coordinate system as function of time (*not* including thrust force or torque)

Library

Sensors & Actuators

Description



The Variable Mass & Inertia Actuator block allows you to vary the mass m and/or inertia tensor $\boldsymbol{I}$ of the Body to which it is connected. The general form of Newton's second law for linear or angular motion is

(*mass* or *inertia*) * *acceleration* = external *force* or *torque*

This block externally varies the leftmost parameter in this law of motion with a Simulink signal.

Caution The Variable Mass & Inertia Actuator does *not* apply any thrust forces or torques associated with the Body's mass loss or gain. Such thrust effects would occur on the left-hand side of the force or torque law as terms proportional to the time derivatives of the mass or inertia tensor, dm/dt or $d\boldsymbol{I}/dt$, multiplied by the related thrust velocities. You must separately apply such thrust forces or torques to the Body with Body Actuators.

How the Actuator Varies a Body's Mass and Inertia Tensor

You connect the Variable Mass & Inertia Actuator block to the original, user-supplied Body at a Body coordinate system (CS). You can connect multiple Variable Mass & Inertia Actuators to a single Body, each Actuator at a separate Body CS port. If Body CS ports are lacking, open the Body dialog and create them as needed.

At each Body CS so connected, the Variable Mass & Inertia Actuator creates an invisible body. The attachment is equivalent to connecting another Body with a Weld, except that the other body's mass properties vary with time. This invisible body has a time-varying mass and/or symmetric inertia tensor supplied by the external Simulink signal. The center of gravity coordinate system (CG CS) of the invisible body is identical to the attached Body CS. The inertia tensor of the invisible body is evaluated at this CS, in this coordinate system's axes.

The Composite Body

SimMechanics creates a combined or *composite* body, made of the invisible, time-varying body created by the Actuator and the original, user-supplied Body. The total mass of the composite body is the sum of the visible Body and the invisible body's masses. The CG of this composite body is recomputed at each time step. The inertia tensor of the composite body is formed at each time step by combining the inertia tensors of the visible Body and the invisible body. The combined inertia tensor is then evaluated at the composite body's new CG.

What The Invisible Body Requires

The time-varying mass and inertia tensor of the invisible body must satisfy these requirements:

- The mass and principal inertial moments can be positive, negative, or zero.

  The only restriction is that the total mass and the principal inertial moments of the composite body be nonnegative.
- The time-varying inertia tensor of the invisible body must be symmetric.

You can mix variable mass and/or variable inertia tensor actuation.

Actuation	Effect on Connected Body
Variable mass alone	Adds a time-varying point mass at the attached Body CS

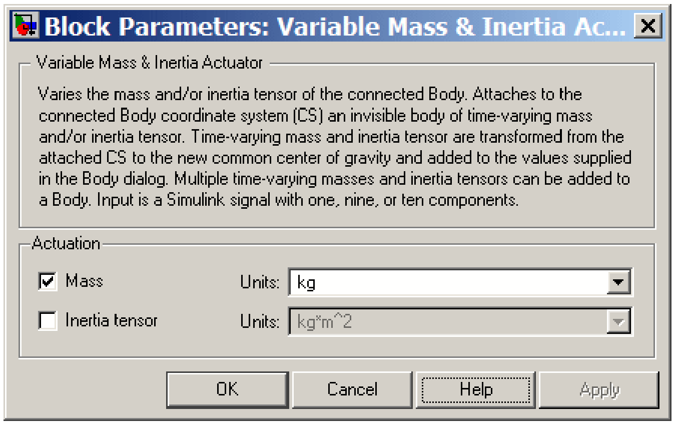
Actuation	Effect on Connected Body
Variable inertia tensor alone	Adds time-varying inertia tensor at the attached Body CS without changing the composite body's total mass
Variable mass and inertia tensor combined	Adds invisible body with time-varying mass and inertia tensor at the attached Body CS

What Does Not Vary in the User-Supplied Body

While the invisible, attached body and the invisible composite body have time-varying mass properties, you do not see any visible changes in the original Body that you are actuating. The mass properties in its dialog do not change.

If you are visualizing the varying-mass/inertia actuated Body as an equivalent ellipsoid, the ellipsoid is rendered using the static data in the Body dialog itself. The ellipsoid rendering ignores the effect of any Variable Mass & Inertia Actuators attached to the Body. See "Rendering Body Shapes in SimMechanics" on page 6-5.

Dialog Box and Parameters


Block Parameters: Variable Mass & Inertia Ac...

Variable Mass & Inertia Actuator

Varies the mass and/or inertia tensor of the connected Body. Attaches to the connected Body coordinate system (CS) an invisible body of time-varying mass and/or inertia tensor. Time-varying mass and inertia tensor are transformed from the attached CS to the new common center of gravity and added to the values supplied in the Body dialog. Multiple time-varying masses and inertia tensors can be added to a Body. Input is a Simulink signal with one, nine, or ten components.

Actuation

[✓] Mass Units: kg

[ ] Inertia tensor Units: kg*m^2

The dialog has one active area, **Actuation**.

Actuation

You can apply a variable mass, a variable inertia tensor, or both, to a body.

If you apply both, you need to bundle the variable mass and inertia tensor into a 10-component signal, in the order shown in the dialog.

Mass
Select the check box to apply an external time-varying mass from the input Simulink signal. In the pull-down menu to the right, select units for this time-varying mass. The default is `kg` (kilograms).

Inertia tensor
Select the check box to apply an external time-varying inertia tensor from the input Simulink signal. In the pull-down menu to the right, select units for this time-varying inertia tensor. The default is `kg-m`^{2} (kilogram-meters^2).

The Simulink input signal has the following components. For variable mass or inertia tensor actuation alone, omit the missing components.

Time-varying mass (scalar)	Time-varying inertia tensor (9-vector): $(I_{11}, I_{21}, I_{31}, I_{12}, \ldots)$

References

[1] Corbin, H. C., and P. Stehle, *Classical Mechanics*, Second Edition, New York, Dover Publications, 1994 (original edition, 1960), chapters 2, 5, and 9.

[2] Goldstein, H., *Classical Mechanics*, Second Edition, Reading, Massachusetts, Addison-Wesley, 1980, chapters 4 and 5.

[3] Piscane, V. L., and R. C. Moore, eds., *Fundamentals of Space Systems*, Johns Hopkins University/Applied Physics Laboratory Series, New York, Oxford University Press, 1994, chapters 3, 4, and 5.

See Also

Body, Body Actuator, Weld

See “Varying a Body’s Mass and Inertia Tensor” on page 4-49 for more on varying the mass and inertia tensor of a body.

Velocity Driver

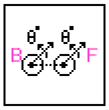
Purpose

Specify linear combination of the linear and angular velocities of two bodies as function of time

Library

Constraints & Drivers

Description



The Velocity Driver block drives a linear combination of the projected translational and angular velocities of two Bodies. The velocities are projected by inner products on to constant vectors you specify.

Let $\boldsymbol{v}_B$, $\boldsymbol{v}_F$ be the two body velocity vectors and ω_B, ω_F be the two body angular velocity vectors. Let $\boldsymbol{c}_B$, $\boldsymbol{c}_F$, $\boldsymbol{d}_B$, $\boldsymbol{d}_F$ be constant vectors. The subscripts 'B' and 'F' refer to base and follower bodies. The Velocity Driver block specifies the linear combination Ω:

$$\Omega = \boldsymbol{c}_B \cdot \boldsymbol{v}_B + \boldsymbol{d}_B \cdot \boldsymbol{\omega}_B - \boldsymbol{c}_F \cdot \boldsymbol{v}_F - \boldsymbol{d}_F \cdot \boldsymbol{\omega}_F = \Omega(t=0) + f(t)$$

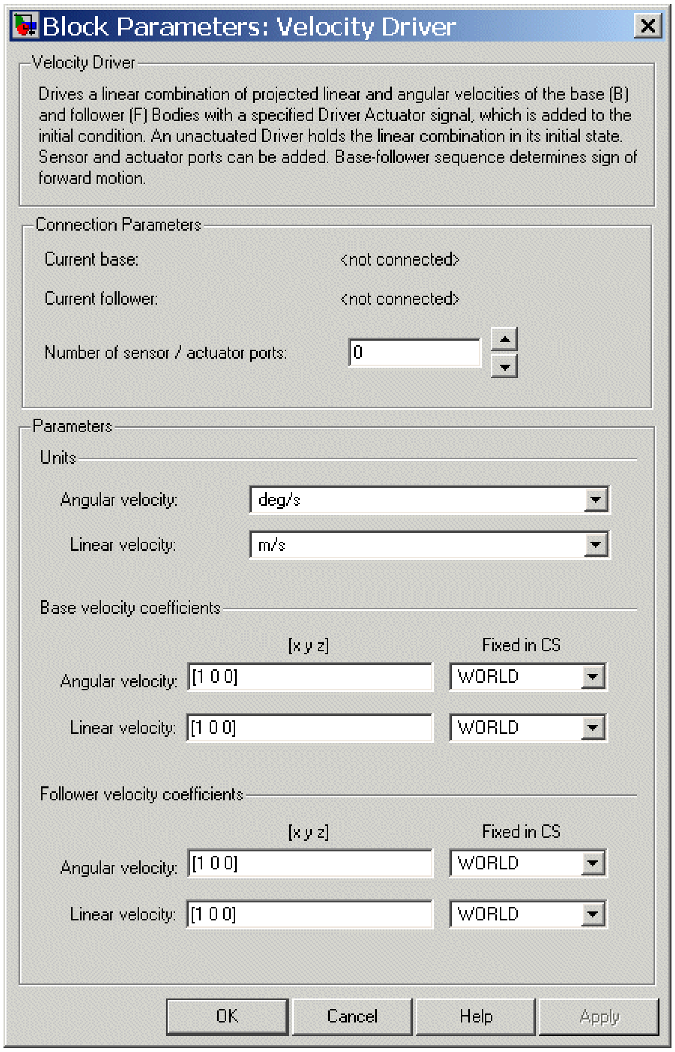
as a function of time *f(t)*. You specify the vectors $\boldsymbol{c}_B$, $\boldsymbol{c}_F$, $\boldsymbol{d}_B$, $\boldsymbol{d}_F$. You also connect a Driver Actuator block to the Velocity Driver.

The Simulink input signal into the Driver Actuator specifies the time-dependent driving function *f(t)* and its first two derivatives, as well as their units. If you do not actuate Velocity Driver, this block acts as a time-independent constraint that freezes the constraint linear combination at its initial value $\Omega(t=0)$ during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loop. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver block and measure the reaction forces/torques between the driven bodies.

Dialog Box and Parameters



Block Parameters: Velocity Driver

Velocity Driver

Drives a linear combination of projected linear and angular velocities of the base (B) and follower (F) Bodies with a specified Driver Actuator signal, which is added to the initial condition. An unactuated Driver holds the linear combination in its initial state. Sensor and actuator ports can be added. Base-follower sequence determines sign of forward motion.

Connection Parameters

Current base: <not connected>

Current follower: <not connected>

Number of sensor / actuator ports: 0

Parameters

Units

Angular velocity: deg/s

Linear velocity: m/s

Base velocity coefficients

	[x y z]	Fixed in CS
Angular velocity:	[1 0 0]	WORLD
Linear velocity:	[1 0 0]	WORLD

Follower velocity coefficients

	[x y z]	Fixed in CS
Angular velocity:	[1 0 0]	WORLD
Linear velocity:	[1 0 0]	WORLD

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.

Current base

When you connect the base (B) connector port on the Velocity Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Velocity Driver Base and Follower Body Connector Ports.

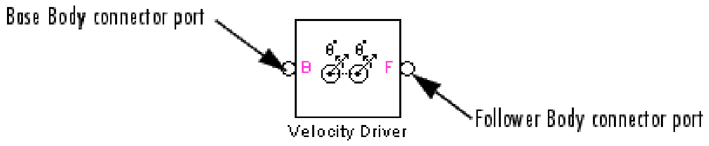
Current follower

When you connect the follower (F) connector port on the Velocity Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Velocity Driver Base and Follower Body Connector Ports.

Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is `0`.

To activate the Driver, connect a Driver Actuator.



Velocity Driver Base and Follower Body Connector Ports

Parameters

The **Parameters** fields are grouped into three sets, **Units**, **Base velocity coefficients**, and **Follower velocity coefficients**.

Units

The vectors $\boldsymbol{c}_{\mathrm{B}}$, $\boldsymbol{c}_{\mathrm{F}}$, $\boldsymbol{d}_{\mathrm{B}}$, $\boldsymbol{d}_{\mathrm{F}}$ carry the implicit units conversion to convert all velocities to the common linear velocity units of *f(t)* that you set in the Driver Actuator connected to the Velocity Driver block.

Angular velocity

From the pull-down menu, choose the common units for all angular velocities. The default is `deg/s` (degrees/second).

The vectors $\boldsymbol{d}_{\mathrm{B}}$ and $\boldsymbol{d}_{\mathrm{F}}$ implicitly carry the units conversion of length/angle. The driving function *f(t)* has the linear velocity units that you set in the Driver Actuator block that you connect to Velocity Driver. If the *f(t)* units differ from the units set in **Linear velocity units** in this dialog, the vectors $\boldsymbol{d}_{\mathrm{B}}$ and $\boldsymbol{d}_{\mathrm{F}}$ implicitly carry the additional units conversion.

Linear velocity

From the pull-down menu, choose the common units for all linear velocities. The default is `m/s` (meters/second).

The driving function *f(t)* has the linear velocity units that you set in the Driver Actuator block that you connect to the Velocity Driver. If the *f(t)* units differ from the units set here, the vectors $\boldsymbol{c}_{\mathrm{B}}$ and $\boldsymbol{c}_{\mathrm{F}}$ implicitly carry the units conversion.

Base Velocity Coefficients

Angular velocity

Under **[x y z]**, enter the **Angular velocity** coefficient vectors for the base Body. These are the components of $\boldsymbol{d}_{\mathrm{B}}$. The default is `[1 0 0]`.

In the **Fixed in CS** pull-down menu, choose which set of coordinates axes, `World` or `Base`, define the vector coefficients of the angular velocity. The default is `WORLD`.

Linear Velocity

Under **[x y z]**, enter the **Linear velocity** coefficient vectors for the base Body. These are the components of $\boldsymbol{c}_{\mathrm{B}}$. The default is `[1 0 0]`.

In the **Fixed in CS** pull-down menu, choose which set of coordinates axes, `World` or `Base`, define the vector coefficients of the linear velocity. The default is `WORLD`.

Follower Velocity Coefficients

Angular velocity

Under **[x y z]**, enter the **Angular velocity** coefficient vector for the follower Body. These are the components of $\boldsymbol{d}_{\mathrm{F}}$. The default is `[1 0 0]`.

In the **Fixed in CS** pull-down menu, choose which set of coordinates axes, `World` or `Follower`, define the vector coefficients of the angular velocity. The default is `WORLD`.

Linear Velocity

Under **[x y z]**, enter the **Linear velocity** coefficient vector for the base Body. These are the components of $\boldsymbol{c}_{\mathrm{F}}$. The default is `[1 0 0]`.

In the **Fixed in CS** pull-down menu, choose which set of coordinates axes, `World` or `Follower`, define the vector coefficients of the linear velocity. The default is `WORLD`.

See Also

Angle Driver, Constraint & Driver Sensor, Driver Actuator, Parallel Constraint

See “Modeling Constraints and Drivers” on page 4-38 for more on restricting DoFs with Drivers.

See “Verifying Machine Topology” on page 4-74 and “How SimMechanics Works” on page 5-15 for more on using drivers in closed loops.

See “Constraints and Drivers” on page 10-5.

Purpose

Represent joint with no DoFs

Library

Joints

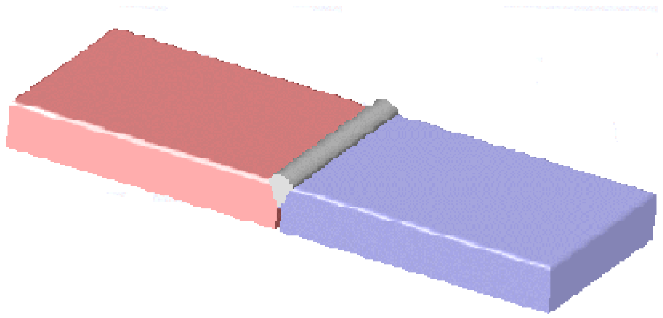
Description

The Weld block represents a joint with no degrees of freedom (DoFs). The two Bodies connected to either side of the Weld block are locked rigidly to one another, with no possible relative motion.

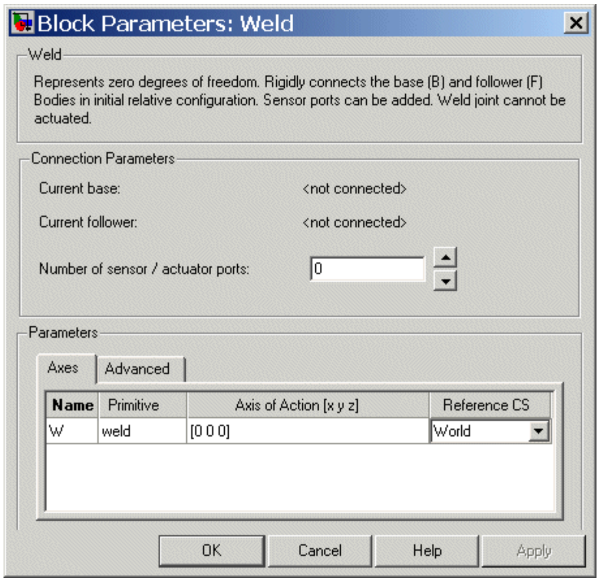
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Weld block is assembled: the origins of these Body CSs must lie along the primitive axes, within assembly tolerancesBut the Body CS origins on either side of the Joint do not have to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves.



Dialog Box and Parameters


Block Parameters: Weld

Weld

Represents zero degrees of freedom. Rigidly connects the base (B) and follower (F) Bodies in initial relative configuration. Sensor ports can be added. Weld joint cannot be actuated.

Connection Parameters

Current base:	<not connected>
Current follower:	<not connected>
Number of sensor / actuator ports:	0

Parameters

Name	Primitive	Axis of Action [x y z]	Reference CS
W	weld	[0 0 0]	World

The dialog has two active areas, **Connection parameters** and **Parameters**.

Connection Parameters

Current base

When you connect the base (B) connector port on the Weld block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Weld Base and Follower Body Connector Ports.

The base Body is automatically connected to the joint primitive `W` in the primitive list in **Parameters**.

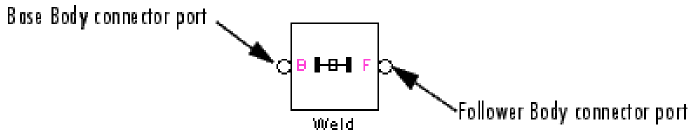
Current follower

When you connect the follower (F) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following figure, Weld Base and Follower Body Connector Ports.

The follower Body is automatically connected to the joint primitive `W` in the primitive list in **Parameters**.

Number of sensor/actuator ports
Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Sensor blocks to this Joint. The default is `0`.

You cannot actuate a Weld joint, and a Weld joint undergoes no motion. A Joint Sensor measures zero motion, but in general nonzero reaction forces, at this joint.



Weld Base and Follower Body Connector Ports

Parameters

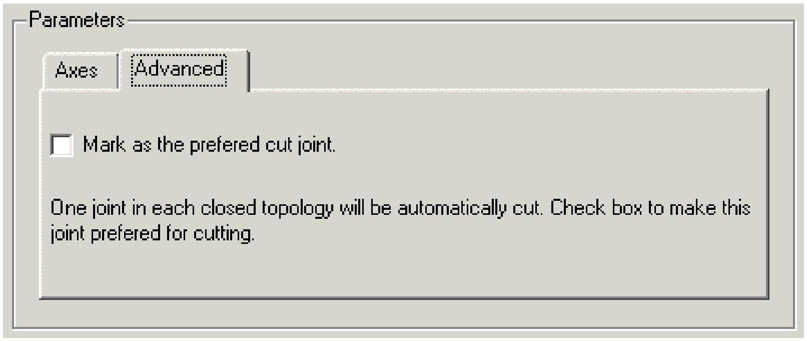
Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are inactive for Weld. This block has no DoF primitives.

Name - Primitive
The primitive list states the names and types of joint primitives that make up the Weld block: a rigid primitive `W` representing no motion.

Axis of Action [x y z]
This field is inactive.

Reference CS
Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is `World`.



☐ Mark as the prefered cut joint.

One joint in each closed topology will be automatically cut. Check box to make this joint prefered for cutting.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is not selected.

See Also

Distance Driver

See "Modeling Joints" on page 4-20 for more on representing DoFs with Joints.

See "Verifying Machine Topology" on page 4-74 and "How SimMechanics Works" on page 5-15 for more on closed loops and cutting.

Commands — Alphabetical List

import_physmod

Purpose

Generate SimMechanics model from Physical Modeling XML file

Syntax

```
import_physmod
import_physmod('-license')
import_physmod('filename.xml')
import_physmod('filename.xml', option1, value1, option2,
   value2, ...)
```

Synopsis

`import_physmod` with no arguments opens the **Physical Modeling XML File Import** dialog. From the dialog, you select the XML file to import and the command options. See "Import Dialog" on page 12-3.

`import_physmod('-license')` displays the third-party license for use of this command. This license information also appears automatically the first time you use the command.

`import_physmod('filename.xml')` generates a SimMechanics model from a Physical Modeling XML file `filename.xml`. The `.xml` extension for the `filename` argument is optional. For a computer-aided design (CAD)-generated XML file, the name of the generated model is the same as the original CAD assembly, regardless of the name of the XML file.

`import_physmod('filename.xml', option1, value1, option2, value2, ...)` generates the SimMechanics model `filename.mdl` from `filename.xml` using the specified option-value pairs when importing. The `.xml` extension for the `filename` argument is optional. See "Input Arguments" on page 12-3 following.

Description

You use this command to generate a dynamic Simulink block diagram model from a Physical Modeling XML file.

The Physical Modeling XML file can be one created by exporting a CAD assembly, for example. You export the Physical Modeling XML file that represents the CAD assembly from your CAD platform through a CAD translator.

The second and final step is to import this Physical Modeling XML file into SimMechanics with the `import_physmod` command. The command creates a SimMechanics model from the information in the Physical Modeling XML file. You can then save, rename, edit, and run the model.

Input Arguments

The input file must be a Physical Modeling XML file. The input filename must have the `.xml` extension, but the `.xml` extension in the `filename` command argument is optional.

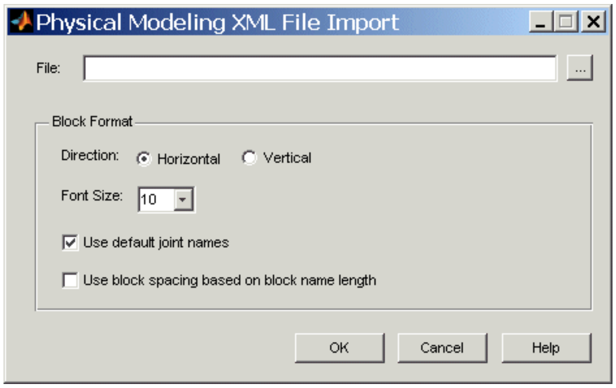
You can execute this command with no input arguments. See “Import Dialog” on page 12-3.

The command options and values follow. At each invocation of `import_physmod`, MATLAB assumes the default options unless you explicitly specify different option values.

Option	Value
`'Direction'`	Diagram growth direction: either `'LR'` (diagram grows from left to right; default) or `'TB'` (diagram grows from top to bottom). Default: `'LR'`
`'FontSize'`	Font size in pixels for the block labels. Default: `10`
`'UseDefaultJointNames'`	Joint naming: either `'on'` (use default joint names like Revolute1, Revolute2, etc., and ignore the full joint names specified in the Physical Modeling XML file) or `'off'` (use the joint names from the Physical Modeling XML file). Default: `'on'`
`'UseBlockNamesForSpacing'`	Block spacing: either `'on'` (blocks are spaced so that their names do not overlap) or `'off'` (use a universal spacing to evenly space all blocks). Default: `'off'`

Import Dialog

If you enter `import_physmod` at the command line with no input arguments, the **Physical Modeling XML File Import** dialog opens. This dialog provides the same control over generating SimMechanics models from Physical Modeling XML files as does the command with input arguments. However, at each invocation of this dialog, MATLAB remembers the last set of import options that you set in this dialog and not the default options.


Physical Modeling XML File Import
File:
...
Block Format
Direction: [✓ Horizontal] Vertical
Font Size: 10
[✓] Use default joint names
Use block spacing based on block name length
OK
Cancel
Help

Click **OK** to complete the command input and start the file import, **Cancel** to stop the file import, and **Help** to open online documentation for this dialog.

The dialog has two active areas, **File** and **Block Format**.

File

Enter the name of a Physical Modeling XML file, including its absolute path on your system. This field replaces the `filename` argument of the `import_physmod` command. There is no default filename.

You can search for files with a file browser by clicking the **...** button. Clicking this button opens the **Import File** browser in the current working directory. Search for and select a Physical Modeling XML file. Close the browser by clicking **Open** or **Cancel**.

If you click **Open** in the browser, the selected filename is copied to the **File** field. The name includes the file's absolute path on your operating system.

Block Format: Direction

Select one of two block diagram growth directions in the generated model, **Horizontal** or **Vertical**. This option replaces the `Direction` option of the `import_physmod` command. The default is **Horizontal**.

Block Format: Font Size

From the pull-down menu, select a font size in pixels for the block labels in the generated model. This option replaces the `FontSize` option of the `import_physmod` command. The default is `10`.

Block Format: Use default joint names

Select this check box to ensure that Joint blocks in the generated model have short, default names that do not include the full model hierarchy. Clearing this check box ensures that the Joint block names are long, containing the full model hierarchy, and take up more space in your model.

The default is selected.

Block Format: Use block spacing based on block name length

Select this check box to enlarge the spacing between blocks in the generated model if their names are long. Clearing this check box leads to block spacing based only on the block sizes, not on the block names.

The default is not selected.

Output Model

The output model name is the same name as the original CAD assembly from which the Physical Modeling XML file was exported and is independent of the name of the XML file itself. You need to save the model once `import_physmod` has generated it. You can change the model's name when you save it.

The `import_physmod` command generates a new SimMechanics model by opening a model window and populating it with blocks.

- The entire model diagram is encapsulated in a subsystem.

- Subsystems below the top level encapsulate subassemblies from the original CAD assembly.
- The machine includes one Ground and one Machine Environment block.
- The other SimMechanics blocks are Bodies and Joints only, corresponding to the parts, degrees of freedom (DoFs), and constraints of the original CAD assembly from which the Physical Modeling XML file was exported.
- A special Root Body occurs at the top of the main hierarchy. A Root Body has no mass and no inertia and does not move.

  Special Root Bodies can occur at the top of each subsystem if such a body is needed to correctly represent DoF constraints in the original CAD assembly.
- The Joints are the appropriate type to represent given combinations of primitives. If no appropriate specific Joint exists, a Custom Joint with the appropriate primitives is used instead. The Joints are Welds if they carry no DoFs.

Example

Create the Physical Modeling XML file for a CAD assembly called `assembly` by exporting the assembly from your CAD platform. Call the file `assembly.xml`.

Then enter

```
import_physmod('assembly.xml')
```

at the command line. The model generation status is indicated by a progress bar. A model window opens, and SimMechanics generates the block diagram in a model file called `assembly.mdl`. You can then save, rename, edit, and run the model.

You can generate the same model, but with point size 14 for the block labels and extra spacing to accommodate the longer block names, by entering

```
import_physmod('assembly.xml','FontSize',14, ...
    'UseBlockNamesForSpacing','on')
```

at the command line.

See Also

See Chapter 7, “Modeling with Computer-Aided Design” for more details and instructions on generating the Physical Modeling XML file for a CAD assembly and importing it into SimMechanics to generate a model.

mech_get_states

Purpose

SimMechanics states from Simulink state vector

Syntax

```
[vector_mgr, mech_states] = mech_get_states(X, block)
[vector_mgr, mech_states] = mech_get_states(X, vectorMgr)
```

Synopsis

`[vector_mgr, mech_states] = mech_get_states(X, block)` returns a state vector manager object whose values reflect the SimMechanics states in the Simulink state vector `X` for the SimMechanics machine containing `block`. The mechanical states `vector_mgr.X` are also returned in `mech_states`.

`[vector_mgr, mech_states] = mech_get_states(X, vectorMgr)` extracts the mechanical states from the Simulink state vector `X` and returns them as an array of states suitable for assignment into `vectorMgr.X`. The state vector manager `vectorMgr` is not modified.

If you call any form of the command with only one output argument, it returns `vector_mgr`. Thus, the three command line entries

```
vm = mech_stateVectorMgr(block);
[vm , mech_states] = mech_get_states(X, vm);
vm.X = mech_states;
```

are equivalent to

```
vm = mech_get_states(X, block);
```

Description

The `mech_get_states` command extracts the mechanical states, with specific values, from the overall Simulink state vector of your SimMechanics model.

The returned vector manager is an instance of the `MECH.StateVectorMgr` class, such as returned by the `mech_stateVectorMgr` command.

Input Arguments

`mech_get_states` accepts two possible combinations of two input arguments. In both cases, the state vector `X` must be compatible with your Simulink model.

The first combination is `mech_get_states(X, block)`, where `X` is the Simulink state vector for your model, and `block` is a string or block handle specifying the absolute path of a block in the machine you want to query.

The second combination is `mech_get_states(X, vectorMgr)`, where `X` is the Simulink state vector for your model, and `vectorMgr` is a mechanical state vector manager object of the `MECH.StateVectorMgr` `class`.

Outputs

`mech_get_states` yields two outputs in the form `[vector_mgr, mech_states]`. You can also call `mech_get_states` with one output.

`vector_mgr` is an instance of the `MECH.StateVectorMgr` object class whose values reflect the mechanical state of the model.

`mech_states` is a vector of the mechanical state values. In the first form of the command, it is identical to `vector_mgr.X`.

Example

First simulate a Stewart platform model for 10 seconds:

```
[t,x,y] = sim('mech_stewart_trajectory',10);
```

Then populate the state vector manager with this model's final state:

```
stewartStateVectorMgr = mech_get_states(x(end,:), ...
   'mech_stewart_trajectory/Plant/Machine Environment')
```

See Also

`mech_runtime_states`, `mech_set_states`, `mech_transfer_states`

To create a state vector manager object, see the command reference for `mech_stateVectorMgr`.

In Simulink, see `sim` and `states`.

mech_runtime_states

Purpose

SimMechanics states from running simulation

Syntax

```
X = mech_runtime_states(vectorMgr)
[X, RTO] = mech_runtime_states(vectorMgr)
[X, RTO] = mech_runtime_states(vectorMgr, RTO)
```

Synopsis

`X = mech_runtime_states(vectorMgr)` returns the vector of instantaneous SimMechanics states in the machine associated with the state vector manager `vectorMgr` in an executing model. The contents of `vectorMgr` are not altered.

`[X, RTO] = mech_runtime_states(vectorMgr)` also returns the Simulink runtime object (RTO) that contains the SimMechanics states. You can use this form to speed up future calls to this command.

`[X, RTO] = mech_runtime_states(vectorMgr, RTO)` uses the Simulink runtime object `RTO` assumed to correspond to the machine associated with `vectorMgr`. This form of the command is faster.

Description

The `mech_runtime_states` command extracts the instantaneous values of the mechanical state of your model while it is running.

The returned value `X` has a format that can be assigned to `vectorMgr.X`, which you can then use to interpret the states.

Input Arguments

`mech_runtime_states` accepts one or two arguments.

The required argument, `vectorMgr`, is a mechanical state vector manager as returned by the `mech_stateVectorMgr` command.

The optional second argument, `RTO`, is a Simulink runtime object that corresponds to the machine referenced by `vectorMgr`.

Outputs

`mech_runtime_states` yields two outputs in the form `[X, RTO]`. You can also call `mech_get_states` with one output.

`X` is the vector of mechanical state values at the instant you query the running model.

`RTO` is a Simulink runtime object.

Example

The following script displays and updates the state of the first Joint in the Stewart platform model, mech_stewart_trajectory. The results appear in the command window. Enter **Ctrl+C** at the command line to halt the simulation.

```
modelName = 'mech_stewart_trajectory';
open(modelName);
groundBlock = find_system(modelName, 'Name', 'Ground1');

vm = mech_stateVectorMgr(groundBlock{1});
set_param(modelName, 'SimulationCommand', 'Start')
[vm.X, rto] = mech_runtime_states(vm)

try
   while (true)
      vm.X = mech_runtime_states(vm, rto);
      clc;
      vm.BlockStates(1) % display
      pause(0.5);
   end
catch
   set_param(modelName, 'SimulationCommand', 'Stop')
   rethrow(lasterr);
end

set_param(modelName, 'SimulationCommand', 'Stop')
```

See Also

`mech_get_states`, `mech_set_states`, `mech_stateVectorMgr`, `mech_transfer_states`

In Simulink, learn about runtime objects and how to access block data during simulation.

In Simulink, see `sim` and `states`.

mech_set_states

Purpose

Populate SimMechanics states in Simulink state vector

Syntax

```
X = mech_set_states(vector_mgr)
X = mech_set_states(vector_mgr, X)
X = mech_set_states(vector_mgr, X, mech_states)
```

Synopsis

`X = mech_set_states(vector_mgr)` returns the Simulink state vector `X` for the model associated with `vector_mgr`. The state vector entries corresponding to the SimMechanics states are filled with the values specified in `vector_mgr`. Entries in `X` that do not correspond to SimMechanics states are set to their initial values, as reported by Simulink.

`X = mech_set_states(vector_mgr, X)` overwrites the entries that correspond to SimMechanics states in the input state vector `X` with the values specified in `vector_mgr`. Entries of `X` that do not correspond to SimMechanics states are left unchanged. Specify `[]` in place of `X` to obtain the initial values for the nonmechanical states, as reported by Simulink.

`X = mech_set_states(vector_mgr, X, mech_states)` overwrites the entries that correspond to SimMechanics states in the input state vector `X` with the values specified in `mech_states` (for example, as reported by `vector_mgr.X`). Entries of `X` that do not correspond to SimMechanics states are left unchanged.

Description

The `mech_set_states` command inserts mechanical state values into a Simulink state vector.

Input Arguments

`mech_set_states` accepts one, two, or three arguments.

The required argument, `vector_mgr`, is an instance of the object class `MECH.StateVectorMgr` corresponding to the machine.

The optional second argument, `X`, is a vector of Simulink state values.

The optional third argument, `mech_states`, requires the second optional argument, `X`, as well. It is a vector of the mechanical state values and is assigned to `vector_mgr.X`.

Outputs

mech_set_states yields one output, X.

X is the Simulink state vector for the model you are querying, with the mechanical state values set to the values specified by the command.

Example

Here you set state values in a simple pendulum model.

```
open mech_spen
vm = mech_stateVectorMgr('mech_spen/Machine Environment');
vm.x = [pi/4 0];
simulinkState = mech_set_states(vm);
IS = simset('InitialState',simulinkState);
[t, xIS] = sim('mech_spen', 5, IS);
```

Compare the initial condition xIS(1,:) with that obtained without setting the initial state in x0(1,:) below:

```
[t, x0] = sim('mech_spen', 5);
```

See Also

mech_get_states, mech_runtime_states, mech_stateVectorMgr, mech_transfer_states

In Simulink, see sim and states.

mech_stateVectorMgr

Purpose

Create machine state vector manager object

Syntax

```
mech_stateVectorMgr
```

Synopsis

`mech_stateVectorMgr` returns a data structure of `MECH.StateVectorMgr` class. It does not return actual state values. You can subsequently assign values to the states in this object, but these values do not propagate to your model. That requires changing the mechanical part of the model's Simulink state vector.

You must call `mech_stateVectorMgr` with one argument, the pathname or handle of any block in the machine whose state you want:

```
MachineState = mech_stateVectorMgr('pathname')
MachineState = mech_stateVectorMgr('handle')
```

Obtain the pathname and handle with the Simulink `gcb` and `gcbh` commands.

You can also call `mech_stateVectorMgr` with an indirect call to the block pathname or handle. Select one of the SimMechanics blocks in the machine and enter one of these commands:

```
MachineState = mech_stateVectorMgr(gcb)
MachineState = mech_stateVectorMgr(gcbh)
```

Note The state manager object includes only the states of a machine made of SimMechanics blocks. Simulink associates the machine to one of the machine's Ground blocks.

Description

Excluding any motion-actuated joint primitives, the total number of mechanical state components is

N = 2*(# of prismatics + # of revolutes) + 8*(# of sphericals)
    + (# of Point-Curve Constraints)

The machine state consists of all the translational and angular positions and velocities of all degrees of freedom (DoFs) in the machine:

- Prismatic and revolute joint primitives each have two state components, a position and a velocity.
- Spherical joint primitives each have eight state components, a quaternion and a quaternion derivative.
- A joint primitive actuated by a Joint Initial Condition Actuator (JICA) is counted like other joint primitives. Because the JICA externally specifies such a primitive's initial position and velocity, the primitive has an active `FixedAtT_0` flag.
- Point-Curve Constraints each have one predictor state component, the arc parameter velocity of the point along the curve.

Joint Primitives Not Counted As Degrees of Freedom

- If the joint primitive is motion actuated with a Joint Actuator block, that joint primitive is not counted in the machine state components.
- The weld primitive contributes no state components.

Input Arguments

There is one input argument, a SimMechanics block's full pathname or handle, or an indirect call to the pathname or handle using the commands `gcb` or `gcbh`. The full path name starts with the model name and continues through any subsystem hierarchy:

```
pathname = modelname/subsystem1/etc.../blockname
```

Obtain the pathname or handle of any block by selecting that block in a window and entering `gcb` or `gcbh` at the command line.

Combine these steps into one with an indirect pathname or handle call. Select a SimMechanics block in the model window and instead enter either command:

```
mech_stateVectorMgr(gcb)
mech_stateVectorMgr(gcbh)
```

Output Arguments

The output of mech_stateVectorMgr is an object of the MECH.StateVectorMgr class.

A machine is a connected set of SimMechanics blocks. Each machine must have at least one Ground block. Simulink chooses one of the Ground blocks as the *machine root*. This root serves as a proxy for the whole machine.

The returned object has four properties. Three are fixed by the machine structure and naming. You can change the values in the fourth, X, but these values do not propagate to the model. Consult “See Also” on page 12-20 following for more about changing a model’s mechanical state vector values.

Property	Variable Type	Content
MachineState.MachineName	string	'modelname/subsystem1 /etc.../rootgroundblock'
MachineState.X	1-by-N real array	[0 0 ... 0]
MachineState.BlockStates	array of N block state managers	Joint primitive and Point-Curve Constraint states
MachineState.StateNames	cell array of N strings	Names of joint primitives and Point-Curve Constraints

Entering the mech_stateVectorMgr command or querying the entire object returns a summary of the object by property.

- The MachineState.X property indicates the number of state components.
- The block state managers of BlockStates are structures, arranged in the array by class: MECH.RPJointStateMgr, MECH.SJointStateMgr, and MECH.PointCurveStateMgr. The components of each manager contain:

- The joint block and prismatic, revolute, and spherical primitive names
- The position and velocity values and units
- The `FixedAtT_0` flag indicating if the primitive is actuated with initial conditions

Using the State Manager Object

Once you create a state vector manager object, you can query the properties individually by entering the full property name:

```
MachineState.MachineName
MachineState.X
```

The state vector manager object is a singleton. If you create the object in `A`, then reassigning `B = A` makes `A` and `B` point to each other, not independent copies. Changing `B` automatically changes `A` and vice versa.

Examples

Some examples illustrate the use of the state vector manager.

Mechanical State Vector with One Primitive

Open the demo model `mech_spen`. Select a SimMechanics block and enter

```
machinestate = mech_stateVectorMgr(gcb)
```

at the command line. The command returns

```
machinestate =
    MECH.StateVectorMgr
    MachineName: 'mech_spen/Ground'
              X: [0 0]
    BlockStates: [1x1 MECH.RPJointStateMgr]
     StateNames: {2x1 cell}
```

The first line in the object is the class and the last four are the properties. The model `mech_spen` contains one Joint block (a Revolute), with two states (angle and angular velocity).

Query individual properties. Entering `machinestate.machinename` returns

```
mech_spen/Ground
```

referring to the one Ground block in the model. Entering `machinestate.X` returns a two-component state vector (N = 2, position and velocity).

```
0      0
```

Entering `machinestate.blockstates` returns

```
MECH.RPJointStateMgr
        BlockName: 'Revolute'
        Primitive: 'R1'
         Position: 0
    PositionUnits: 'rad'
         Velocity: 0
    VelocityUnits: 'rad/s'
       FixedAtT_0: 'off'
```

There are one Joint (a Revolute) and no Point-Curve Constraints, so there is only one state manager of class `MECH.RPJointStateMgr`. This property gives detailed Joint information: block name, primitive name, position and velocity values and units, and the absence of initial condition actuators.

Entering `machinestate.statenames` returns the names of the Joint block, the joint primitive, and the states.

```
'Revolute:R1:Position'
'Revolute:R1:Velocity'
```

Mechanical State Vector with Multiple Primitives

Construct an unnamed model with Ground and Body blocks connected by a Telescoping Joint. Then select one of the blocks and enter `machinestate = mech_stateVectorMgr(gcb)` at the command line. Simulink returns

```
machinestate =
    MECH.StateVectorMgr
    MachineName: 'untitled/Ground'
              X: [0 0 0 0 0 0 0 0 0 0]
    BlockStates: [2x1 MECH.BlockStateMgr]
     StateNames: {10x1 cell}
```

The unnamed model is associated with its Ground block and has a spherical and a prismatic primitive, hence 10 components in the state vector. To see those primitive names, enter `machinestate.statenames` to obtain

```
'Telescoping:S:Quaternion:1'
'Telescoping:S:Quaternion:2'
'Telescoping:S:Quaternion:3'
'Telescoping:S:Quaternion:4'
'Telescoping:P1:Position'
'Telescoping:S:Quaternion_dot:1'
'Telescoping:S:Quaternion_dot:2'
'Telescoping:S:Quaternion_dot:3'
'Telescoping:S:Quaternion_dot:4'
'Telescoping:P1:Velocity'
```

The quaternion and the prismatic position make the first five components, while the quaternion derivative and prismatic velocity make the last five.

See Also

To change actual mechanical state values, consult these commands: `mech_get_states`, `mech_runtime_states`, `mech_set_states`, `mech_transfer_states`

Point-Curve Constraint, Prismatic, Revolute, Spherical

See Chapter 3, “Representing Motion”.

See “Counting Degrees of Freedom” on page 4-77.

See “Trimming Mechanical Models” on page 8-18 and “Linearizing Mechanical Models” on page 8-32.

In Simulink, see `gcb`, `gcbh`, `gcs`.

Purpose

Map states from one machine's state vector manager to another

Syntax

```
dststates = mech_transfer_states(srcMgr, dstMgr,
   transferstruct)
dststates = mech_transfer_states(srcMgr, dstMgr,
   transferstruct, srcstates)
dststates = mech_transfer_states(srcMgr, dstMgr,
   transferstruct, srcstates, dststates)
```

Synopsis

`dststates = mech_transfer_states(srcMgr, dstMgr, transferstruct)` copies selected state entries from the source state vector manager `srcMgr` to the state vector `dststates`, which can be assigned to `dstMgr.X`, where `dstMgr` is the state vector manager for the destination machine. The states to be copied are specified in `transferstruct`. Any destination states not specified in `transferstruct` retain their values as specified initially in `dstMgr`.

`dststates = mech_transfer_states(srcMgr, dstMgr, transferstruct, srcstates)` performs the same transfer, but takes the "source" states in `srcstates` rather than the values in `srcMgr.X`.

`dststates = mech_transfer_states(srcMgr, dstMgr, transferstruct, srcstates, dststates)` performs the same transfer, but also takes the "destination" states in `dststates` rather than the values in `dstMgr.X` for those destination states that are not the target of an assignment.

Either or both of the `dst` and `src` fields in `transferstruct` can also be indices for their respective vector manager's `X` vectors. In the case that they both are, this command is equivalent to:

```
dststates(transferstruct.dst) = srcstates(transferstruct.src)
```

Description

The `mech_transfer_states` command copies specified SimMechanics states between state vector managers.

Input Arguments

`mech_transfer_states` accepts three, four, or five arguments.

The required first argument, `srcMgr`, is a state vector manager for the source system. It is an object of the `MECH.StateVectorMgr` class as returned by the `mech_stateVectorMgr` command.

The required second argument, `dstMgr`, is a state vector manager for the destination system. It is an object of the `MECH.StateVectorMgr` class as returned by the `mech_stateVectorMgr` command.

The required third argument, `transferstruct`, is a structured array whose `src` and `dst` fields contain either the indices or the names of the source and destination states as known to their respective state managers.

The optional fourth argument, `srcstates`, is a source mechanical state vector.

The optional fifth argument, `dststates`, requires the optional fourth argument and is a destination mechanical state vector.

Outputs

`mech_transfer_states` yields one output, `dststates`.

`dststates` is the destination mechanical state vector.

Example

These commands copy the first state in `srcMgr` to the third state in `dstMgr`:

```
transferstruct(1).src=srcMgr.StateNames{1};
transferstruct(1).dst=dstMgr.StateNames{3};
dstMgr.X = mech_transfer_states(srcMgr, dstMgr, transferstruct);
```

See Also

`mech_get_states`, `mech_runtime_states`, `mech_set_states`, `mech_stateVectorMgr`

In Simulink, see `sim` and `states`.

A

Technical Conventions

Mechanical Conventions and Abbreviations

Right-Hand Rule Is Assumed

For rotational motion and vector cross products $\boldsymbol{a} \times \boldsymbol{b}$, the right-hand (RH) rule is always assumed.

Vector Multiplication

Scalar-vector products and matrix-vector multiplication are denoted by $\boldsymbol{a}\cdot\boldsymbol{b}$ and $M\cdot\boldsymbol{v}$, respectively.

Common Abbreviations

These are the abbreviations of mechanical terms most commonly used in this guide.

Abbreviation	Meaning
CG	Center of gravity
CS	Coordinate system
DoF	Degree of freedom
RF	Reference frame

Glossary Terms

Special mechanical or SimMechanics terms are frequently hyperlinked online to entries in the Glossary.

Mechanical Units

SimMechanics accepts any mixture of meters-kilograms-seconds or MKS (SI), cgs, and English units.

Quantity	MKS (SI)	cgs	English
Length	meter (m)	centimeter (cm)	inch (in), foot (ft)
Time	second (s)	second (s)	second (s)
Mass	kilogram (kg)	gram (g)	slug (slug)
Velocity	meter/second (m/s)	centimeter/second (cm/s)	inch/second (in/sec), foot/second (ft/sec)
Acceleration (Gravity)	meter/second^2 (m/s^2)	centimeter/second^2 (cm/s^2)	inch/second^2 (in/sec^2), foot/second^2 (ft/sec^2)
Force	newton (N)	dyne (dyn)	pound (lb)
Angle	radian (rad), degree (deg)	radian (rad), degree (deg)	radian (rad), degree (deg)
Inertia	kilogram-meter^2 (kg-m^2)	gram-centimeter^2 (g-cm^2)	slug-foot^2 (slug-ft^2)
Angular velocity	radian/second (rad/s), degree/second (deg/s)	radian/second (rad/s), degree/second (deg/s)	radian/second (rad/sec), degree/second (deg/sec)
Angular acceleration	radian/second^2 (rad/s^2), degree/second^2 (deg/s^2)	radian/second^2 (rad/s^2), degree/second^2 (deg/s^2)	radian/second^2 (rad/sec^2), degree/second^2 (deg/sec^2)
Torque	newton-meter (N-m)	dyne-centimeter (dyn-cm)	pound-foot (lb-ft)

Bibliography

[1] Gill, P. E., W. Murray, and M. Wright, *Practical Optimization,* San Diego, Academic Press, 1981.

[2] Goldstein, H., *Classical Mechanics,* Second Edition, Reading, Massachusetts, Addison-Wesley, 1980.

[3] Goodman, L. E., and W. H. Warner, *Statics*, Mineola, New York, Dover Publications, 2001 (original edition, 1964).

[4] Goodman, L. E., and W. H. Warner, *Dynamics*, Mineola, New York, Dover Publications, 2001 (original edition, 1963).

[5] Haug, E. J., *Computer-Aided Kinematics and Dynamics of Mechanical Systems, Volume 1: Basic Methods,* Boston, Allyn & Bacon, 1989.

[6] José, J. V., and E. J. Saletan, *Classical Dynamics: A Contemporary Approach,* Cambridge, Cambridge University Press, 1998.

[7] JPL DARTS Web page on spatial operator algebra: http://dshell.jpl.nasa.gov/References/index.html.

[8] Marrin, C., and B. Campbell, *Teach Yourself VRML 2 in 21 Days*, Indianapolis, Indiana, Sams.net, 1997.

[9] Marsden, J. E., and T. S. Ratiu, *Introduction to Mechanics and Symmetry,* Second Edition, New York, Springer-Verlag, 1999.

[10] Meriam, J. L., *Engineering Mechanics,* Fourth Edition, two volumes, New York, John Wiley and Sons, 1997.

[11] Murray, R. M., Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation,* Boca Raton, Florida, CRC Press, 1994.

[12] von Schwerin, R., *MultiBody System SIMulation: Numerical Methods, Algorithms, and Software,* Berlin, Springer-Verlag, 1999.

Glossary

actuator

An *actuator* converts a Simulink signal into SimMechanics force, torque, or motion signals.

- You can configure a *body actuator* to apply forces/torques to a body either as an explicit function of time or through feedback forces/torques.
- You can configure a *joint actuator* to apply forces/torques between the bodies connected on either side of the joint.
- You can configure a *driver actuator* to apply relative motion between the bodies connected on either side of the driver.

SimMechanics also has two specialized actuators, one for setting *joint initial conditions* and one for applying *stiction* to a joint.

Actuator	**What the Actuator Does**
Body Actuator	Applies forces to a body
Driver Actuator	Applies motion to a time-dependent constraint
Joint Actuator	Applies forces or motions to a joint
Joint Initial Condition Actuator	Sets a joint's initial conditions
Joint Stiction Actuator	Applies static and kinetic friction to joint motion

In SimMechanics, an Actuator block has an open round SimMechanics connector port ○ for connecting with a Body, Joint, or Driver block and an angle bracket > Simulink inport for connecting with normal Simulink blocks, such as Source blocks for generating force or torque signals.

See also ***body***, ***connector port***, ***driver***, ***initial condition actuator***, ***joint***, ***primitive joint***, ***sensor***, and ***stiction actuator***.

adjoining CS

The *adjoining CS* of a Body CS is the CS on the neighboring body or ground directly connected to the original Body CS by a Joint, Constraint, or Driver.

See also ***body***, ***Body CS***, ***coordinate system (CS)***, ***grounded CS***, and ***World***.

assembled configuration

A machine is in its *assembled configuration* once it passes through its initial configuration and its disassembled joints are then assembled. The assembly of joints can change body and joint configurations.

See also ***home configuration*** and ***initial configuration***.

assembled joint

Restricts the Body coordinate systems (CSs) on the two bodies at either end of the joint.

- For an *assembled prismatic joint,* the two Body CS origins must lie along the prismatic axis. The two Bodies translate relatively along the same axis.

  For an assembled joint with multiple prismatic primitives, the two Body CS origins must lie in the plane or space defined by the directions of the prismatic axes.

- For an *assembled revolute joint,* the two Body CS origins must be collocated. The two Bodies rotate relatively about the same axis.

  For an assembled joint with multiple revolute primitives, the two Body CS origins must be collocated.

- For an *assembled spherical joint,* the two Body CS origins must be collocated at the spherical primitive's pivot point. The two Bodies pivot relatively about this common origin.

You specify an assembly tolerance for assembled joints, the maximum dislocation distance allowed between all pairs of assembled Body CS origins and the maximum angle of misalignment between all pairs of assembled Body motion axes. If the distance dislocations and/or axis misalignments in an assembled joint grow larger than the assembly tolerance, the simulation stops with an error.

See also ***assembly tolerance, Body CS, collocation***, ***disassembled joint, joint,*** and ***primitive joint***.

assembly

Representation of a machine in computer-aided design. An assembly includes *parts* (bodies with full geometric, mass, and inertia tensor information), as well as *constraints* (sometimes called *mates*) restricting the *degrees of freedom* of the parts.

Every assembly has a *fundamental root*. Assemblies can also have one or more *subassemblies* branching off the main assembly at a single point.

Assembly specifications typically also include design tolerances and how subassemblies move and how they are connected to the main assembly.

See also ***body***, ***computer-aided design (CAD)***, ***constraint***, ***degree of freedom (DoF)***, ***fundamental root***, ***inertia tensor***, ***mass***, ***part***, and ***subassembly***.

assembly tolerance

Determines how closely an ***assembled joint*** must be collocated and aligned. An ***assembled joint*** is connected on either side to Body coordinate systems (CSs) on two Bodies and restricts the relative configurations and motions of those Body CSs.

The assembly tolerances set the *maximum dislocation* of Body CS origins and *maximum misalignment* of motion axes allowed in assembled joints during the simulation.

- For *assembled prismatic primitives,* each pair of Body CS origins must lie in the subspace defined by the prismatic axes. Each pair of Bodies translates along these common axes.
- For *assembled revolute primitives,* each pair of Body CS origins must be collocated and their respective rotational axes aligned. Each pair of Bodies rotates about these common axes.
- For an *assembled spherical primitive,* the pair of Body CS origins must be collocated. The two Bodies pivot about this common origin.

SimMechanics attempts to assemble all joints in your machine at the start of simulation, including initially disassembled joints. If it cannot, the simulation stops with an error.

If the two Body CSs separate or the joint axes misalign in a way that makes their connecting assembled joint primitives no longer respect the assembly tolerances, the simulation stops with an error.

See also ***assembled joint, Body CS, collocation, disassembled joint,*** and ***joint***.

axis-angle rotation

A representation of a three-dimensional spherical rotation as a rotation axis vector $\boldsymbol{n} = (n_x,n_y,n_z)$ of unit length ($\boldsymbol{n}\cdot\boldsymbol{n} = n_x^2 + n_y^2 + n_z^2 = 1$) and a rotation angle θ. Define the rotation axis by the vector $\boldsymbol{n}$; rotate about that axis by θ using the right-hand rule. The $\boldsymbol{n}$ axis is sometimes called the *eigenaxis*.

The rotation axis direction is equivalent to specifying two independent angles; θ is the third independent angle making up the rotation.

In VRML, you represent body rotations by a vector signal $[n_x\ n_y\ n_z\ \theta]$.

See also ***degree of freedom (DoF)***, ***Euler angles***, ***primitive joint***, ***quaternion***, ***right-hand rule***, ***rotation matrix***, and ***VRML***.

base (base body)

The point from which the joint is directed. The joint directionality runs from base to follower body.

Joint directionality sets the direction and the positive sign of all joint motion and force-torque data.

See also ***body, directionality, follower (follower body),*** and ***right-hand rule***.

body

The basic element of a mechanical system or machine. It is characterized by its

- Mass properties (mass and inertia tensor)
- Position and orientation in space

- Attached Body coordinate systems

Bodies are connected to one another by joints, constraints, or drivers. Bodies carry no degrees of freedom.

You can attach to a Body block any number of Body coordinate systems (CSs). All SimMechanics Bodies automatically maintain a minimum of one Body CS at the body's center of gravity (CG). The Body block has special axis triad CS ports ⊞, instead of the open, round connector ports ○, to indicate the attached Body CSs.

See also ***actuator***, ***adjoining CS, Body CS, center of gravity (CG), convex hull, coordinate system (CS), degree of freedom (DoF), equivalent ellipsoid, inertia tensor***, ***joint, local CS, mass,*** and ***sensor***.

Body CS

A local coordinate system (CS) attached to a body, carried along with that body's motion. In general, bodies accelerate as they move, and therefore Body CSs define noninertial reference frames.

You can attach any number of Body CSs to a Body block, and you can choose where to place the Body CS origins and how to orient the Body CS axes. The Body block has special axis triad CS ports ⊞ instead of the open, round connector ports, to give you access to these Body CSs for connecting Joint, Sensor, and Actuator blocks.

Every Body block has an automatic, minimum Body CS at its center of gravity (CG). By default, it also has two other Body CSs for connection to adjacent Joints. The origin and axis orientation of each Body CSs once set by the user during Body configuration, are interpreted as fixed rigidly in that body during the simulation.

See also ***body, center of gravity (CG), convex hull, coordinate system (CS), ground, grounded CS, local CS, reference frame (RF),*** and ***World***.

CAD

See ***computer-aided design (CAD)***.

center of gravity (CG)

The *center of gravity* or center of mass of an extended body is the point in space about which the entire body balances in a uniform gravitational field. For translational dynamics, the body's entire mass can be considered as if concentrated at this point.

Every Body block has an automatic, minimum Body coordinate system (CS) with its origin at the CG — the CG CS. This origin point and the Body CS coordinate axes remain fixed rigidly in the body during the simulation.

See also ***body, Body CS, degree of freedom (DoF), inertia tensor, kinematics,*** and ***primitive joint***.

CG

See ***center of gravity (CG)***.

closed loop system

You can disconnect a *closed loop system* into two separate systems only by cutting more than one joint. The number of closed loops is equal to the minimum number, minus one, of cuttings needed to disconnect the system into two systems.

See also ***open system*** and ***topology***.

collocation

Two points in space are *collocated* if they are coincident, within assembly tolerances.

See also ***assembled joint***, ***assembly tolerance***, and ***disassembled joint***.

composite joint

A joint compounded from more than one joint primitive and thus representing more than one degree of freedom. The joint primitives constituting a composite joint are the *primitives* of that joint.

A spherical primitive represents three rotational degrees of freedom, but is treated as a primitive.

See also ***constrained joint, degree of freedom (DoF), joint,*** and ***primitive joint***.

computer-aided design (CAD)

Computer-aided design systems or platforms provide an environment to design machines, with full geometric information about *parts* (bodies) and their spatial relationships, as well as the *degrees of freedom* and mass properties (*masses* and *inertia tensors*) of the parts.

A CAD representation of a machine is an *assembly*.

See also ***assembly***, ***body***, ***constraint***, ***degree of freedom (DoF)***, ***inertia tensor***, ***mass***, and ***part***.

connection line

You connect each SimMechanics block to another by using *SimMechanics connection lines*. These lines function only with SimMechanics blocks. They do not carry signals, unlike normal Simulink lines, and cannot be branched. You cannot link connection lines directly to Simulink lines.

Connection lines appear red and dashed if they are not anchored at both ends to a connector port ○. Once you anchor them, the lines become black and solid.

See also ***actuator, connector port,*** and ***sensor***.

connector port

An anchor for a **connection line**. Each SimMechanics block has one or more open round SimMechanics connector ports ○ for connecting to other SimMechanics blocks. You must connect these round ports only to other SimMechanics round ports. When an open connector port ○ is attached to a connection line, the Port changes to solid ●.

A *Connection Port block* is provided in the SimMechanics library to create a round SimMechanics connector port for an entire subsystem on that subsystem's boundary.

See also ***actuator, connection line,*** and ***sensor***.

constrained joint

A composite joint with one or more internal constraints restricting the joint's primitives.

An example is the Screw block, which has a prismatic and a revolute primitive with their motions in fixed ratio. Only one of these degrees of freedom is independent.

See also ***degree of freedom (DoF), joint,*** and ***primitive joint***.

constraint

A restriction among degrees of freedom imposed independently of any applied forces/torques. A *constraint* removes one or more independent degrees of freedom, unless that constraint is *redundant* and restricts degrees of freedom that otherwise could not move anyway. Constraints can also create *inconsistencies* with the applied forces/torques that lead to simulation errors.

- Constraints are *kinematic*: they must involve only coordinates and/or velocities. Higher derivatives of coordinates (accelerations, etc.) are determined by the Newtonian force and torque laws and cannot be independently constrained.
- Constraints are *holonomic* (integrable into a form involving only coordinates) or *nonholonomic* (not integrable; that is, irreducibly involving velocities).
- Constraints specify kinematic relationships that are explicit functions of time *(rheonomic)* or not *(scleronomic)*.

  In SimMechanics, scleronomic constraints are called *Constraints,* and rheonomic constraints are called *Drivers*. SimMechanics Constraint and Driver blocks are attached to pairs of Body blocks.
- In computer-aided design (CAD) assemblies, a constraint restricts one or more degrees of freedom of the assembly parts. (CAD constraints are sometimes called *mates*.) When a CAD assembly is converted to a SimMechanics model, such restricted degrees of freedom are translated into specific joints. (In SimMechanics, bodies have no degrees of freedom.)

See also ***assembly***, ***body***, ***computer-aided design (CAD)***, ***degree of freedom (DoF)***, ***directionality***, ***driver***, ***joint***, ***machine precision constraint***, ***part***, ***stabilizing constraint***, and ***tolerancing constraint***.

convex hull

The surface of minimum area with convex (outward-bowing) curvature that passes through all the spatial points in a set. In three dimensions, this set must contain at least four distinct, non-coplanar points to make a closed surface with nonzero enclosed volume.

In SimMechanics, the convex hull is an option for visualizing a body. The set of points is all the Body coordinate system (CS) origins configured in that Body block. The visualization of an entire machine is the set of the convex hulls of all its bodies.

If a Body has fewer than four distinct, non-coplanar Body CSs, its convex hull is a lower-dimensional figure:

- Three distinct Body CSs produce a triangle without volume.
- Two distinct Body CSs produce a line without area.
- One Body CS produces a point without length.

See also ***body***, ***Body CS***, and ***equivalent ellipsoid***.

coordinate system (CS)

A *coordinate system* is defined, in a particular reference frame, by a choice of *origin* and orientation of *coordinate axes*, assumed orthogonal and Cartesian (rectangular). An observer attached to that CS measures distances from that origin and directions relative to those axes.

SimMechanics has two CS types:

- World: *global* or *absolute inertial* CS at rest
- Local:
  - Grounded CS
  - Body CS, including the center of gravity (CG) CS

Local coordinate systems are sometimes called *working frames*.

See also ***body, Body CS, center of gravity (CG), convex hull, grounded CS, local CS, reference frame (RF)***, and ***World***.

CS

A ***coordinate system (CS)***.

degree of freedom (DoF)

A single coordinate of relative motion between two bodies. Such a coordinate is free only if it can respond without constraint or imposed motion to externally applied forces or torques. For translational motion, a DoF is a linear coordinate along a single direction. For rotational motion, a DoF is an angular coordinate about a single, fixed axis.

A prismatic joint primitive represents a single translational DoF. A revolute joint primitive represents a single rotational DoF. A spherical joint primitive represents three rotational DoFs in angle-axis form. A weld joint primitive represents zero DoFs.

See also ***body, coordinate system (CS), dynamics, joint,*** and ***kinematics***.

directionality

The *directionality* of a joint, constraint, or driver is its direction of forward motion.

The joint directionality is set by the order of the joint's connected bodies and the direction of the joint axis vector. One body is the *base* body, the other the *follower* body. The joint direction runs from base to follower, up to the sign of the joint axis vector. Reversing the base-follower order or the joint axis vector direction reverses the forward direction of the joint.

Joint directionality sets the direction and the positive sign of all joint motion and force-torque data.

Directionality of constraints and drivers is similar, except there is no joint axis, only the base-follower sequence.

See also ***base (base body)***, ***body, follower (follower body), joint,*** and ***right-hand rule***.

disassembled joint

A *disassembled joint* need not respect the assembly tolerances of your machine.

- For a *disassembled prismatic primitive*, the Body coordinate system (CS) origins do not have to lie on the prismatic axis.
- For a *disassembled revolute primitive*, the Body CS origins do not have to be collocated.
- For a *disassembled spherical primitive*, the Body CS origins do not have to be collocated.

SimMechanics attempts to assemble all disassembled joints in your machine at the start of simulation. If it cannot, the simulation stops with an error.

You can use disassembled joints only in a closed loop, with no more than one per loop.

See also ***assembled joint***, ***assembly tolerance***, ***closed loop system***, ***collocation***, and ***topology***.

DoF

A ***degree of freedom (DoF)***.

driver

A constraint that restricts degrees of freedom as an explicit function of time (a rheonomic constraint) and *independently* of any applied forces/torques. A driver removes one or more independent degrees of freedom, unless that driver is *inconsistent* with the applied forces/torques and forces a simulation error.

In SimMechanics, you specify the driver function of time in a dialog box in terms of an input Simulink signal from a Driver Actuator.

SimMechanics Driver blocks are attached to pairs of Body blocks.

See also ***actuator***, ***body***, ***constraint***, ***directionality***, and ***degree of freedom (DoF)***.

dynamics

A *forward dynamic* analysis of a mechanical system specifies

- The topology of how bodies are connected
- The degrees of freedom (DoFs) and constraints among DoFs

- All the forces/torques applied to the bodies
- The mass properties (masses and inertia tensors) of the bodies
- The initial condition of all DoFs:
  - Initial linear coordinates and velocities
  - Initial angular coordinates and velocities

The analysis then solves Newton's laws to find the system's motion for all later times.

Inverse dynamics is the same, except that the system's motion is specified and the forces/torques necessary to produce this motion are determined.

Dynamics is distinguished from *kinematics* by explicit specification of applied forces/torques and body mass properties.

See also ***constraint***, ***degree of freedom (DoF)***, ***inertia tensor***, ***kinematics***, ***mass***, and ***topology***.

equivalent ellipsoid

The *equivalent ellipsoid* of a body is the homogeneous solid ellipsoid, centered at the body's center of gravity, with the same principal moments of inertia and principal axes as the body. A homogeneous solid ellipsoid is the simplest body with three distinct principal moments.

Every body has a unique equivalent ellipsoid, but a given homogeneous ellipsoid corresponds to an infinite number of other, more complicated, bodies. The rotational dynamics of a body depend only on its equivalent ellipsoid (which determines its principal moments and principal axes), not on its detailed shape.

In SimMechanics, the equivalent ellipsoid is an option for visualizing a body.

See also ***body***, ***convex hull***, ***dynamics***, ***inertia tensor***, ***principal axes***, and ***principal inertial moments***.

Euler angles

A representation of a three-dimensional spherical rotation as a product of three successive independent rotations about three independent axes by three independent (Euler) angles. Follow the Euler angle convention by

1. Rotating about one axis (which rotates the other two).
2. Then rotating about a second axis (rotated from its original direction) not identical to the first.
3. Lastly, rotating about another axis not identical to the second.

There are 3*2*2 = 12 possible Euler angle rotation sequences.

See also ***axis-angle rotation***, ***degree of freedom (DoF)***, ***primitive joint***, ***quaternion***, ***right-hand rule***, and ***rotation matrix***.

fixed part

A *"fixed" part* of a computer-aided design assembly or subassembly is a part that is welded to the assembly or subassembly root. That is, a "fixed" part cannot move relative to the root.

See also ***computer-aided design (CAD)***, ***root body***, ***subassembly***, and ***subassembly root***.

follower (follower body)

The point to which the joint is directed. The joint directionality runs from base to follower body.

Joint directionality sets the direction and the positive sign of all joint motion and force-torque data.

See also ***base (base body), body, directionality,*** and ***right-hand rule***.

fundamental root

A point in a computer-aided assembly that does not move. All translational and rotational motion of parts in the assembly reference this unmoving point.

See also ***assembly***, ***computer-aided design (CAD)***, ***part***, and ***root body***.

ground

A *ground* or *ground point* is a point fixed at rest in the absolute or global inertial World reference frame.

Each ground has an associated grounded coordinate system (CS). The grounded CS's origin is identical to the ground point, and its coordinate axes are always parallel to the coordinate axes of World.

See also ***body***, ***coordinate system (CS)***, ***grounded CS***, ***machine***, and ***World***.

grounded CS

A local CS attached to a ground point. It is at rest in World, but its origin is wherever the ground point is and thus, in general, shifted with respect to the World CS origin. The coordinate axes of a grounded CS are always parallel to the World CS axes.

The World coordinate axes are defined so that:

+*x* points right

+*y* points up (gravity in -*y* direction)

+*z* points out of the screen, in three dimensions

You automatically create a Grounded CS whenever you set up a Ground block.

See also ***adjoining CS***, ***body***, ***Body CS***, ***coordinate system (CS)***, ***ground***, ***local CS***, and ***World***.

home configuration

The bodies of a machine are in their *home configuration* when they are positioned and oriented purely according to the positions and orientations entered into the Body dialogs. This configuration assumes zero body velocities.

See also ***assembled configuration*** and ***initial configuration***.

inertia tensor

The *inertia* or *moment of inertia tensor* of an extended rigid body describes its internal mass distribution and the body's angular acceleration in response to an applied torque.

Let V be the body's volume and $\rho(\boldsymbol{r})$ its mass density, a function of vector position $\boldsymbol{r}$ within the body. Then the components of the inertia tensor $\boldsymbol{I}$ are:

$$I_{ij} = \int_V dV \left[\delta_{ij} |\boldsymbol{r}|^2 - r_i r_j \right] \rho(\boldsymbol{r})$$

The indices i, j range over 1, 2, 3, or x, y, z. This tensor is a real, symmetric 3-by-3 matrix or equivalent MATLAB expression.

SimMechanics always assumes the inertia tensor of a body is evaluated in that body's center of gravity coordinate system (CG CS). That is, the origin is set to the body's CG and the coordinate axes are the CG CS axes.

Because the CG CS of a Body block is fixed rigidly in the body during simulation, the values of the inertia tensor components do not change as the body rotates.

See also ***body, Body CS, equivalent ellipsoid***, ***mass, principal axes,*** and ***principal inertial moments***.

initial condition actuator

An *initial condition actuator* gives you a way to move a system's degrees of freedom nondynamically to prepare a system for dynamical integration, in a way consistent with all constraints.

In SimMechanics, the initial conditions are applied to a joint primitive.

See also ***actuator, dynamics,*** and ***kinematics***.

initial configuration

A machine is in its *initial configuration* once all initial condition actuators have been applied to its joints. This step can change the positions and orientations of the machine's bodies, as well as apply nonzero initial velocities.

See also ***assembled configuration***, ***home configuration***, and ***initial condition actuator***.

joint

Represents one or more mechanical degrees of freedom between two bodies. Joint blocks connect two Body blocks in a SimMechanics schematic. Joints have no mass properties such as a mass or an inertia tensor.

A *joint primitive* represents one translational or rotational degree of freedom or one spherical (three rotational degrees of freedom in angle-axis form). Prismatic and revolute primitives have motion axis vectors. A weld primitive has no degrees of freedom.

A *primitive joint* contains one joint primitive. A *composite joint* contains more than one joint primitive.

Joints have a directionality set by their base-to-follower Body order and the direction of the joint primitive axis. The sign of all motion and force-torque data is determined by this directionality.

See also ***actuator***, ***assembled joint***, ***base (base body)***, ***body***, ***composite joint***, ***constrained joint***, ***constraint***, ***degree of freedom (DoF)***, ***directionality, disassembled joint***, ***follower (follower body)***, ***ground***, ***inertia tensor, massless connector***, ***primitive joint***, and ***sensor***.

kinematics

A *kinematic* analysis of a mechanical system specifies topology, degrees of freedom (DoFs), motions, and constraints, without specification of applied forces/torques or the mass properties of the bodies.

The *machine state* at some time is the set of all

- Instantaneous positions and orientations
- Instantaneous velocities

of all bodies in the system, for both linear (translational) and angular (rotational) DoFs of the bodies.

Specification of applied forces/torques and solution of the system's motion as a function of time are given by the system's dynamics.

See also ***constraint***, ***degree of freedom (DoF)***, ***dynamics***, and ***topology***.

local CS

A local coordinate system (CS) is attached to either a Ground or a Body:

- Grounded CS
- Body CS

You define Body CSs when you configure the properties of a Body. A Grounded CS is automatically defined when you represent a ground point by a Ground block.

A *grounded CS* is always at rest in the World reference frame. The origin of this Grounded CS is the same point as the ground point and thus, in general, not the same as the World CS origin.

A *Body CS* is fixed rigidly in the body and carried along with that body's motion. To indicate an attached coordinate system, a Body block has an axis triad CS port ⊞ in place of the open, round connector port ○.

See also ***body***, ***Body CS***, ***coordinate system (CS)***, ***grounded CS***, ***reference frame (RF)***, and ***World***.

machine

In a SimMechanics model, a *machine* is a complete, connected block diagram representing one mechanical system. It is topologically isolated from any other machine in your model and has at least one ground.

A SimMechanics model has one or more machines.

See also ***ground*** and ***topology***.

machine precision constraint

A *machine precision constraint* is a constraint numerically implemented on the constrained degrees of freedom to the precision of your computer processor's arithmetic. It is the most robust, computationally intensive, and slowest-simulating constraint.

The precision to which the constraint is maintained depends on scale or the physical system of units.

See also ***constraint***, ***stabilizing constraint***, and ***tolerancing constraint***.

mass

The proportionality between a force on a body and the resulting translational acceleration of that body.

Let V be the body's volume and $\rho(\boldsymbol{r})$ its mass density, a function of position $\boldsymbol{r}$ within the body. Then the mass m is:

$$m = \int_V dV\, \rho(\boldsymbol{r})$$

The mass is a real, positive scalar or equivalent MATLAB expression.

A body's mass is insensitive to choice of reference frame, coordinate system origin, or coordinate axes orientation.

See also ***body*** and ***inertia tensor***.

massless connector

A *massless connector* is equivalent to two joints whose respective primitive axes are spatially separated by a fixed distance. You can specify the gap distance and the axis of separation. The space between the degrees of freedom is filled by a rigid connector of zero mass.

You cannot actuate or sense a massless connector.

See also ***disassembled joint*** and ***joint***.

open system

You can disconnect an *open system* into two separate systems by cutting no more than one joint.

Such systems can be divided into two types:

- An *open chain* is a series of bodies connected by joints and topologically equivalent to a line.
- An *open tree* is a series of bodies connected by joints in which at least one body has more than two joints connected to it. Bodies with more than two connected joints define *branch points* in the tree. A tree can be disconnected into multiple chains by cutting the branch points.

The end body of a chain is a body with only one connected joint.

See also ***closed loop system*** and ***topology***.

part

A *part* represents a body in a computer-aided design (CAD) assembly. In CAD representations, a part typically includes full geometric, as well as mass and inertia tensor, information about a body.

Body degrees of freedom are restricted in CAD by constraints (sometimes called *mates*).

After translation into a SimMechanics model, parts are represented by bodies.

See also ***assembly***, ***body***, ***computer-aided design (CAD)***, ***constraint***, ***degree of freedom (DoF)***, ***inertia tensor***, and ***mass***.

physical tree

You obtain the *physical tree* representation of a machine topology from the full machine topology by removing actuators and sensors and cutting each closed loop once. The physical tree retains bodies, joints, constraints, and drivers.

See also ***closed loop system, open system, spanning tree,*** and ***topology***.

primitive joint

A *primitive joint* expresses one degree of freedom (DoF) or coordinate of motion, if this DoF is a translation along one direction (*prismatic joint*) or a rotation about one fixed axis (*revolute joint*).

In SimMechanics, a *spherical joint* (three DoFs: two rotations to specify directional axis, one rotation about that axis) is also treated as a primitive joint.

These three types of primitive joints are the *joint primitives* from which composite joints are built.

A weld primitive has no degrees of freedom.

See also ***composite joint*** and ***joint***.

principal axes

The inertia tensor of a body is real and symmetric and thus can be diagonalized, with three real eigenvalues and three orthogonal eigenvectors. The *principal axes* of a body are these eigenvectors.

See also ***equivalent ellipsoid***, ***inertia tensor***, and ***principal inertial moments***.

principal inertial moments

The inertia tensor of a body is real, symmetric, and diagonalizable, with three real eigenvalues and three orthogonal eigenvectors. The *principal inertial moments* or *principal moments of inertia* of a body are these eigenvalues, the diagonal values when the tensor is diagonalized.

The principal moments of a real body satisfy the *triangle inequalities:* the sum of any two moments is greater than or equal to the third moment.

If two of the three principal moments are equal, the body has some symmetry and is dynamically equivalent to a *symmetric top*. If all three principal moments are equal, the body is dynamically equivalent to a *sphere*.

See also ***equivalent ellipsoid***, ***inertia tensor***, and ***principal axes***.

quaternion

A *quaternion* represents a three-dimensional spherical rotation as a four-component row vector of unit length:

$$q = \left[n_x \sin(\theta/2),\ n_y \sin(\theta/2),\ n_z \sin(\theta/2),\ \cos(\theta/2)\right] = [\boldsymbol{q}_\mathrm{v}, q_\mathrm{s}]$$

with $q{*}q = 1$. The vector $\boldsymbol{n} = (n_\mathrm{x},n_\mathrm{y},n_\mathrm{z})$ is a three-component vector of unit length: $\boldsymbol{n}\cdot\boldsymbol{n} = 1$. The unit vector $\boldsymbol{n}$ specifies the axis of rotation. The rotation angle about that axis is θ and follows the right-hand rule.

The axis-angle representation of the rotation is just $[\ \boldsymbol{n}\ \theta\ ]$.

See also ***axis-angle rotation***, ***degree of freedom (DoF)***, ***Euler angles***, ***primitive joint***, ***right-hand rule***, and ***rotation matrix***.

reference frame (RF)

The state of motion of an observer.

An *inertial* RF is a member of a set of all RFs moving uniformly with respect to one another, without relative acceleration. This set defines inertial space.

An RF is necessary but not sufficient to define a coordinate system (CS). A CS requires an origin point and a oriented set of three orthogonal axes.

See also ***coordinate system (CS)***, ***local CS***, and ***World***.

RF

A ***reference frame (RF)***.

right-hand rule

The *right-hand rule* is the standard convention for determining the sign of a rotation: point your right thumb into the positive rotation axis and curl your fingers into the forward rotational direction.

See also ***degree of freedom (DoF), directionality,*** and ***joint***.

root body

After a computer-aided design (CAD) assembly is translated into a SimMechanics model, a block sequence Ground — Root Weld — Root Body or Root Body — Root Weld — Fixed Body represents the CAD assembly's fundamental or subassembly root, respectively.

In CAD assemblies, the fundamental or subassembly root represents a fixed point relative to which all part motion or subassembly part motion is measured.

See also ***assembly***, ***body***, ***computer-aided design (CAD)***, ***fixed part***, ***fundamental root***, ***ground***, ***subassembly***, and ***subassembly root***.

rotation matrix

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix R: $R^{\mathrm{T}}R = RR^{\mathrm{T}} = I$, where I is the 3-by-3 identity and R^{T} is the transpose of R.

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} = \begin{pmatrix} R_{\mathrm{xx}} & R_{\mathrm{xy}} & R_{\mathrm{xz}} \\ R_{\mathrm{yx}} & R_{\mathrm{yy}} & R_{\mathrm{yz}} \\ R_{\mathrm{zx}} & R_{\mathrm{zy}} & R_{\mathrm{zz}} \end{pmatrix}$$

In general, R requires three independent angles to specify the rotation fully. There are many ways to represent the three independent angles. Here are two:

- You can form three independent rotation matrices R_1, R_2, R_3, each representing a single independent rotation. Then compose the full rotation matrix R with respect to fixed coordinate axes (like World) as a product of these three: $R = R_3 * R_2 * R_1$. The three angles are *Euler angles*.
- You can represent R in terms of an *axis-angle rotation* $\boldsymbol{n} = (n_x, n_y, n_z)$ and θ with $\boldsymbol{n} \cdot \boldsymbol{n} = 1$. The three independent angles are θ and the two needed to orient $\boldsymbol{n}$. Form the antisymmetric matrix:

$$\hat{J} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

Then *Rodrigues' formula* simplifies R:

$$R = \exp(\theta J) \;=\; I \;+\; J\sin\theta \;+\; J^2(1-\cos\theta)$$

See also ***axis-angle rotation***, ***degree of freedom (DoF)***, ***Euler angles***, ***primitive joint***, ***quaternion***, and ***right-hand rule***.

sensor

Measures the motion of, or forces/torques acting on, a body or joint. A sensor can also measure the reaction forces in a constraint or driver constraining a pair of bodies.

In SimMechanics, a Sensor block has an open round SimMechanics connector port ○ for connecting with a Body or Joint block and an angle bracket > Simulink outport for connecting with normal Simulink blocks, such as a Sinks block like Scope.

See also ***actuator***, ***body***, ***connector port***, ***constraint***, ***driver***, ***joint***, and ***primitive joint***.

spanning tree

You obtain the spanning tree representation of a machine topology from the full machine topology by removing everything except bodies and joints and cutting each closed loop once.

See also ***closed loop system***, ***open system***, ***physical tree***, and ***topology***.

stabilizing constraint

Numerically implements a constraint by modifying the dynamics of a system so that the constraint manifold is attractive, without changing the constrained solution. This constraint solver type is computationally the least intensive, the least robust, and the fastest-simulating.

The precision to which the constraint is maintained depends on scale or the physical system of units.

See also ***constraint***, ***machine precision constraint***, and ***tolerancing constraint***.

stiction actuator

Applies discontinuous friction forces to a joint primitive according to the relative velocity of one body with the other body.

If this relative velocity drops below a specified threshold, the relative motion ceases and the bodies or joints become locked rigidly to one another by static friction.

Above that threshold, the bodies or joints move relative to one another with kinetic friction.

See also ***actuator***, ***composite joint***, ***dynamics***, ***joint***, and ***primitive joint***.

subassembly

Representation of a subset of machine parts and constraints (mates) in computer-aided design (CAD).

A subassembly is attached to its parent CAD assembly at a single branching point.

A subassembly is either *flexible* or *rigid*. That is, its parts either can move with respect to one another or they cannot.

See also ***assembly***, ***computer-aided design (CAD)***, ***constraint***, ***part***, and ***subassembly root***.

subassembly root

A point in a computer-aided design (CAD) subassembly that does not move relative to the assembly point off of which it branches. All translational and rotational motion of parts in the subassembly reference this unmoving point.

See also ***assembly***, ***computer-aided design (CAD)***, ***fundamental root***, ***part***, ***root body***, and ***subassembly***.

tolerancing constraint

A *tolerancing constraint* is numerically implemented on constrained degrees of freedom only up to a specified accuracy and/or precision.

This accuracy/precision is independent of any accuracy/precision limits on the solver used to integrate the system's motion, although constraints cannot be maintained to greater accuracy than the accuracy of the solver.

The precision to which the constraint is maintained depends on scale or the physical system of units.

Tolerancing constraints are moderately robust and moderately intensive and execute at moderate speed. They are less intensive than machine precision constraints, but computationally more intensive than stabilizing constraints.

Tolerancing constraints are most useful in realistic simulation of constraint slippage ("slop" or "play").

See also ***constraint***, ***machine precision constraint***, and ***stabilizing constraint***.

topology

The global connectivity of the elements of a machine.

For mechanical models, the *elements* are bodies and the *connections* are joints, constraints, and drivers. Two topologies are *equivalent* if you can transform one system into another by continuous deformations and without cutting connections or joining elements.

An *open system* has no closed loops.

- An *open chain* is topologically equivalent to a line; and each body is connected to only two other bodies, if the body is internal, or one other body if it is at an end.
- An *open tree* has one or more *branch points*. A branch point is where an internal body is connected to more than two other bodies. A tree can be disconnected into multiple chains by cutting at the branch points.

A *closed loop system* has one or more closed loops. The number of closed loops is equal to the minimum number of joints, minus one, that must be cut to dissociate a system into two disconnected systems.

An actual system can have one of these primitive topologies or can be built from multiple primitive topologies.

See also ***body***, ***closed loop system***, ***joint***, ***machine***, and ***open system***.

VRML

Virtual Reality Modeling Language, an open, Web-oriented ISO standard for defining three-dimensional virtual worlds in multimedia and the Internet. Virtual Reality Toolbox uses VRML to create and populate virtual worlds with user-defined bodies.

In VRML, body rotations are represented in the axis-angle form. The SimMechanics RotationMatrix2VR block converts rotation matrices to the equivalent axis-angle forms.

See also ***axis-angle rotation*** and the Web3D Consortium at `www.web3d.org`.

World

In SimMechanics, *World* is both the absolute inertial ***reference frame (RF)*** and absolute ***coordinate system (CS)*** in that RF. World has a fixed origin and fixed coordinate axes that cannot be changed.

The World coordinate axes are defined so that:

+*x* points right

+*y* points up (gravity in -*y* direction)

+*z* points out of the screen, in three dimensions

See also ***adjoining CS***, ***coordinate system (CS)***, ***ground***, ***grounded CS***, and ***reference frame (RF)***.

Index