

Aerospace Blockset

For Use with Simulink[®]

Modeling
|

Simulation
|






Implementation
|

User's Guide

Version 1



How to Contact The MathWorks:

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	support@mathworks.com	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	Mail

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

Aerospace Blockset User's Guide

© COPYRIGHT 2002 by The MathWorks, Inc.

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

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: July 2002

Online only

New for Version 1 (Release 13)

Using This Guide

Organization of this Document	vi
Getting Help Online	vii
For Further Help and Feedback	viii
Related Products	ix
Requirements for the Aerospace Blockset	ix
Other Related Products	x
Typographical Conventions	xi
Installation	xii

Introduction

1

What Is the Aerospace Blockset?	1-2
Getting Started with the Aerospace Blockset	1-5
Opening the Aerospace Blockset on Windows Platforms	1-5
Opening the Aerospace Blockset on UNIX Platforms	1-7
Modeling with the Aerospace Blockset	1-9
Model Definition	1-9
Model Simulation	1-19
Learning More About Simulink and Aerospace Blockset	1-20

Case Study: Missile Guidance System

2

Missile Guidance System Model	2-2
Modeling Airframe Dynamics	2-3
ISA Atmosphere Model block	2-5
Aerodynamics & Equations of Motion Subsystem	2-7
Modeling a Classical Three-Loop Autopilot	2-10
Trimming and Linearizing an Airframe Model	2-11
Autopilot Design	2-12
Modeling the Homing Guidance Loop	2-13
Guidance Subsystem	2-14
Seeker/Tracker Subsystem	2-17
Simulating the Missile Guidance System	2-20
Extending the Model	2-22
References	2-23

Block Reference

3

Blocks — By Category	3-2
Actuator Library Blocks	3-3
Animation Library Blocks	3-3
Environment/Atmosphere Library Blocks	3-3
Environment/Gravity Library Blocks	3-3
Environment/Wind Library Blocks	3-3
Equations of Motion/3DoF Library Blocks	3-4
Equations of Motion/6DoF Library Blocks	3-4
GNC Library Blocks	3-4
Propulsion Library Blocks	3-6
Transformation/Axes Library Blocks	3-6

Transformations/Units Library Blocks	3-7
Block Reference Page Description	3-8
Blocks — Alphabetical List	3-9

Using This Guide

“Using This Guide” provides general information about the Aerospace Blockset and the documentation. The following sections are included:

Organization of this Document (p. vi)	Provides overview of the Aerospace Blockset documentation.
Getting Help Online (p. vii)	Describes the options available in accessing help while using the Aerospace Blockset.
Related Products (p. ix)	Describes other MathWorks products that are especially relevant to the kinds of tasks you can perform with the Aerospace Blockset as well as the requirements for the Aerospace Blockset.
Typographical Conventions (p. xi)	One-page table summarizing the typographical conventions used in this document.
Installation (p. xii)	Installation note.

Organization of this Document

This guide contains tutorial sections that are designed to help you become familiar with using the Aerospace Blockset with Simulink®, as well as a reference section for finding detailed information on particular blocks in the blockset:

- “Introduction” on page 1-1 provides an overview of fundamental Aerospace Blockset concepts.
- “Case Study: Missile Guidance System” on page 2-1 provides an overview of an application of the Aerospace Blockset.
- “Block Reference” on page 3-1 provides descriptions of each block’s operation, parameters, and characteristics.

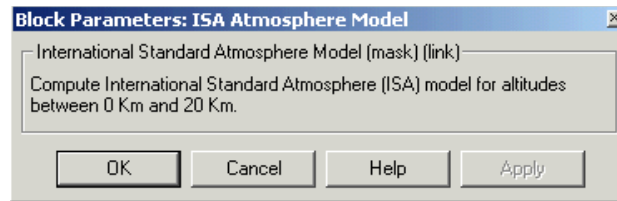
Use this guide in conjunction with the software to learn about the powerful features of the Aerospace Blockset.

Note The User’s Guide documentation for the Aerospace Blockset assumes that you are familiar with Simulink. See the Simulink documentation for more information.

Getting Help Online

There are a number of easy ways to get help on the Aerospace Blockset while you're working at the computer:

- **Block Help** — Click the **Help** button in any block dialog box to view the online reference documentation for that block.



- **Simulink Library Browser** — Right-click a block to access the help for that block.
- **Help browser** — Select **Full Product Family Help** from the **Help** menu, or type `doc` or `helpdesk` at the command line to display the Help browser. Select **Aerospace Blockset** in the **Contents** pane.
- **Command Line** — Type `doc('block name')` at the command line to access the help for a block with the name `block name`. Spaces and capitalization in the block name are ignored.
- **Help Desk (remote)** — Use a Web browser or the Help browser to connect to the MathWorks Web site at `www.mathworks.com`. Follow the **Documentation** link on the **Support** Web page for remote access to the documentation.
- **Release Information** — Select **Full Product Family Help** from the **Help** menu, or type `whatsnew` at the MATLAB[®] command line and select the Aerospace Blockset Release Notes from the **Contents** pane of the Help browser. You can also type `info aeroblks` at the MATLAB command line to view detailed release information related to bug fixes and enhancements.

The Release Notes contain information about new features and recent changes to the version of the Aerospace Blockset that you are using.

For Further Help and Feedback

We hope you enjoy using the Aerospace Blockset and look forward to hearing your comments and suggestions.

support@mathworks.com	Technical support
suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports

Visit the MathWorks Web site at www.mathworks.com for complete contact information.

Related Products

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

Requirements for the Aerospace Blockset

You must have the following products installed to use the Aerospace Blockset:

- MATLAB® 6.5
- Control System Toolbox 5.2
- Simulink 5.0

Virtual Reality-Based Visualization

The optional virtual reality-based visualization with the Aerospace Blockset requires the Virtual Reality Toolbox Version 3.0. The toolbox ships with its own virtual reality viewer.

You can improve the virtual reality 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.

Other Related Products

The toolboxes listed below all include functions that extend the capabilities of MATLAB. The blocksets all include blocks that extend the capabilities of Simulink. These products will enhance your use of the Aerospace Blockset in various applications.

Product	Description
Control System Toolbox	Design and analyze feedback control systems
Real-Time Workshop	Generate C code from Simulink models
Stateflow [®]	Design and simulate event-driven systems
Stateflow Coder	Generate C code from Stateflow charts
Virtual Reality Toolbox	Create and manipulate virtual reality worlds from within MATLAB and Simulink
Real-Time Workshop Embedded Coder	Generate production code for embedded systems

For more information about any of these products, see either

- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at www.mathworks.com; see the “Products” section

Typographical Conventions

This manual uses some or all of these conventions.

Item	Convention	Example
Example code	Monospace font	To assign the value 5 to A, enter <code>A = 5</code>
Function names, syntax, filenames, directory/folder names, and user input	Monospace font	The <code>cos</code> function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code>
Buttons and keys	Boldface with book title caps	Press the Enter key.
Literal strings (in syntax descriptions in reference chapters)	Monospace bold for literals	<code>f = freqspace(n, 'whole')</code>
Mathematical expressions	<i>Italics</i> for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$.
MATLAB output	Monospace font	MATLAB responds with <code>A =</code> <code>5</code>
Menu and dialog box titles	Boldface with book title caps	Choose the File Options menu.
New terms and for emphasis	<i>Italics</i>	An <i>array</i> is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	<code>[c,ia,ib] = union(...)</code>
String variables (from a finite list)	<i>Monospace italics</i>	<code>sysc = d2c(sysd, 'method')</code>

Installation

The Aerospace Blockset follows the same installation procedure as the MATLAB toolboxes. See the MATLAB installation documentation for your platform.

Introduction

Welcome to the Aerospace Blockset, the premier tool for aerospace system simulation and code generation. This section contains the following topics, which will help introduce you to the Aerospace Blockset:

“What Is the Aerospace Blockset?” on page 1-2

This section provides an overview of the Aerospace Blockset.

“Getting Started with the Aerospace Blockset” on page 1-5

This section describes how to open the Aerospace Blockset in Simulink.

“Modeling with the Aerospace Blockset” on page 1-9

This section provides a tutorial on building Simulink models and simulating them.

What Is the Aerospace Blockset?

The Aerospace Blockset brings the full power of Simulink® to aerospace system design, integration, and simulation by providing key aerospace subsystems and components in the adaptable Simulink block format. From environmental models to equations of motion, from gain scheduling to animation, the blockset gives you the core components to rapidly and efficiently assemble a broad range of large aerospace system architectures.

Use the Aerospace Blockset and Simulink to develop your aerospace system concepts, and to efficiently revise and test throughout the life cycle of your design. Use the Aerospace Blockset together with Real-Time Workshop® to automatically generate code for real-time execution in rapid prototyping and for hardware-in-the-loop systems.

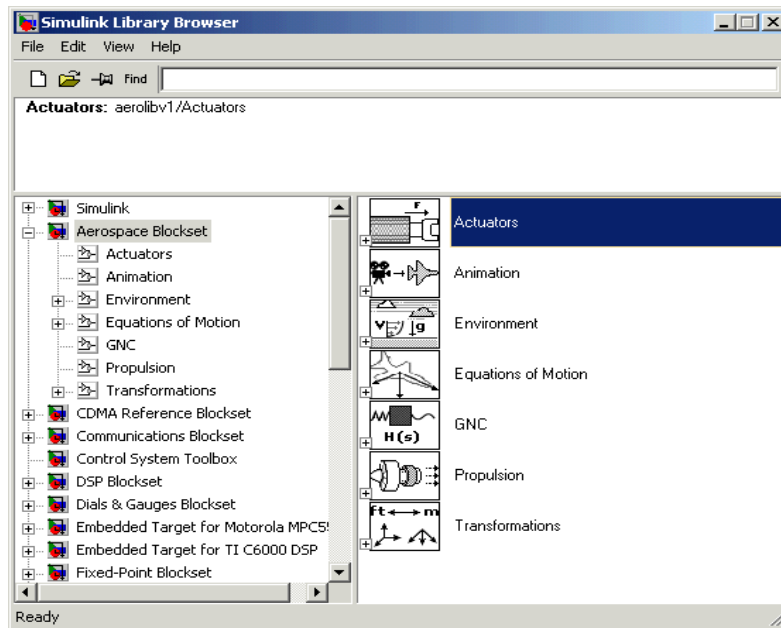
The Aerospace Blockset is a collection of block libraries for use with Simulink. The blockset extends Simulink by providing core components for large aerospace systems. You can use blocks from the Aerospace Blockset in the same way that you would use any other Simulink blocks, combining them with blocks from other libraries to create sophisticated aerospace systems.

The Aerospace Blockset libraries are designed specifically for aerospace applications and include such key operations as environmental modeling, modeling equations of motion, gain scheduling, unit conversion, and more.

You will find that the blockset can be put to work rapidly. The blocks implement mathematical representations from textbooks and references and the experience of the engineers at The MathWorks.

Notice THE MATHWORKS PROGRAMS HAVE NOT BEEN TESTED OR CERTIFIED BY ANY GOVERNMENT AGENCY OR INDUSTRY REGULATORY ORGANIZATION OR ANY OTHER THIRD PARTY. THE PROGRAMS SHOULD NOT BE RELIED ON AS THE SOLE BASIS TO SOLVE A PROBLEM WHOSE INCORRECT SOLUTION COULD RESULT IN INJURY TO PERSON OR PROPERTY. THE PROGRAMS ARE NOT DESIGNED NOR TESTED FOR A LEVEL OF RELIABILITY SUITABLE FOR USE AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT OR OTHER INFORMATION SYSTEMS OR HARDWARE, THE FAILURE OF WHICH CAN REASONABLY BE EXPECTED TO CAUSE DEATH OR PERSONAL INJURY OR PROPERTY OR ENVIRONMENTAL DAMAGE. LICENSEE AGREES THAT PRIOR TO USING, INCORPORATING OR DISTRIBUTING THE PROGRAMS IN ANY PRODUCT, IT WILL THOROUGHLY TEST THE PRODUCT AND THE FUNCTIONALITY OF THE PROGRAMS IN THAT PRODUCT AND BE SOLELY RESPONSIBLE FOR ANY PROBLEMS OR FAILURES.

The Aerospace Blockset contains a collection of blocks organized in a set of nested libraries. The best way to explore the blockset is to expand the **Aerospace Blockset** entry in the Simulink Library Browser. The fully expanded library list is shown here.



See the Simulink documentation for complete information about the Library Browser. To access the blockset through its own window (rather than through the Library Browser), enter

`aerolib`

in the MATLAB Command Window. Double-click any library in the window to display its contents.

For a complete list of all the blocks in the Aerospace Blockset by library, see “Blocks — By Category” on page 3-2.

Getting Started with the Aerospace Blockset

To get started with the Aerospace Blockset, you need to use Simulink. All the blocks in the Aerospace Blockset are designed for use together with the blocks in the Simulink libraries. This section describes how to open the Aerospace Blockset on Windows and on UNIX platforms.

- “Opening the Aerospace Blockset on Windows Platforms” on page 1-5
- “Opening the Aerospace Blockset on UNIX Platforms” on page 1-7

Opening the Aerospace Blockset on Windows Platforms

You can open the Aerospace Blockset from the Simulink Library Browser.

Opening the Simulink Library Browser

To start Simulink, click the  icon in the MATLAB toolbar, or type

```
simulink
```

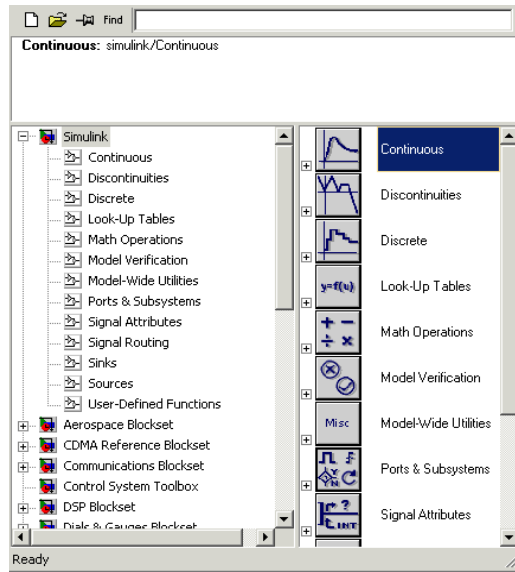
at the command line.

The Simulink Libraries

The libraries in the Simulink Library Browser contain all the basic elements you need to construct a model. Look here for basic math operations, switches, connectors, simulation control elements, and other items that do not have a specific aerospace orientation.

Opening the Aerospace Blockset

On Windows platforms, the Simulink Library Browser opens when you start Simulink. The left pane contains a list of all the blocksets that you currently have installed.



The first item in the list is the Simulink blockset itself, which is already expanded to show the available Simulink libraries. Click the \oplus symbol to the left of any blockset name to expand the hierarchical list and display that blockset's libraries within the browser.

To open the Aerospace Blockset window from the MATLAB command line, enter


```
aerolib
```

See the Simulink documentation for a complete description of the Library Browser.

Opening the Aerospace Blockset on UNIX Platforms

You can open the Aerospace Blockset from the Simulink Library window.

Opening the Simulink Library Window

To start Simulink, click the  icon in the MATLAB toolbar, or type

```
simulink
```

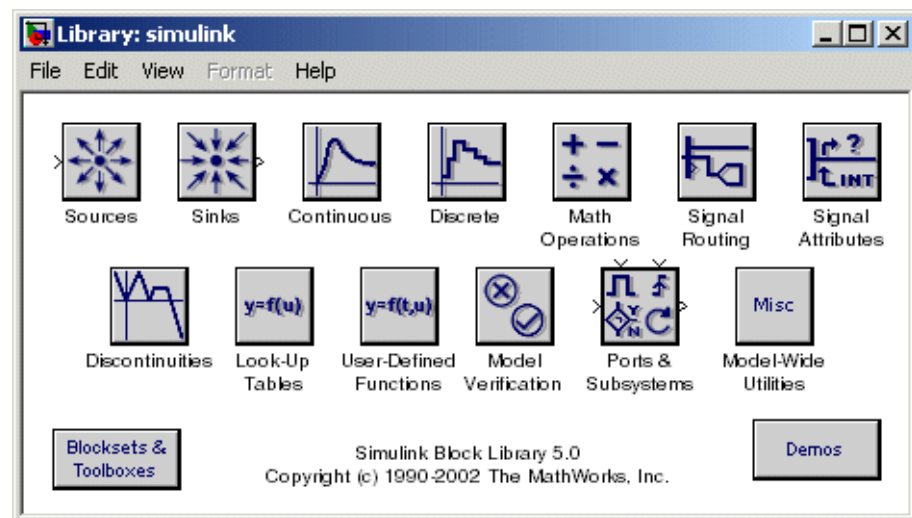
at the command line.

The Simulink Libraries

The libraries in the Simulink Library window contain all the basic elements you need to construct a model. Look here for basic math operations, switches, connectors, simulation control elements, and other items that do not have a specific aerospace orientation.

Opening the Aerospace Blockset

On UNIX platforms, the following Simulink Library window opens when you start Simulink. To view other installed blocksets, double-click the **Blocksets & Toolboxes** button.



Double-click the **Aerospace Blockset** icon to open the Aerospace Blockset.

To open the Aerospace Blockset window from the MATLAB command line,
enter

```
aerolib
```

Modeling with the Aerospace Blockset

If you have never used Simulink before, take some time to get acquainted with its features.

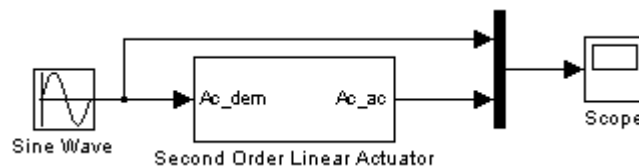
Begin by learning the two basic stages in model construction, discussed in the following sections:

- “Model Definition” on page 1-9
- “Model Simulation” on page 1-19

Model Definition

Simulink is a software package for modeling, simulating, and analyzing dynamic systems. Try building a simple model that drives an actuator with a sine wave and displays the actuator’s position superimposed on the sine wave.

Note If you prefer to open the complete model shown below instead of building it, type `aeroblktutorial` at the MATLAB command line.




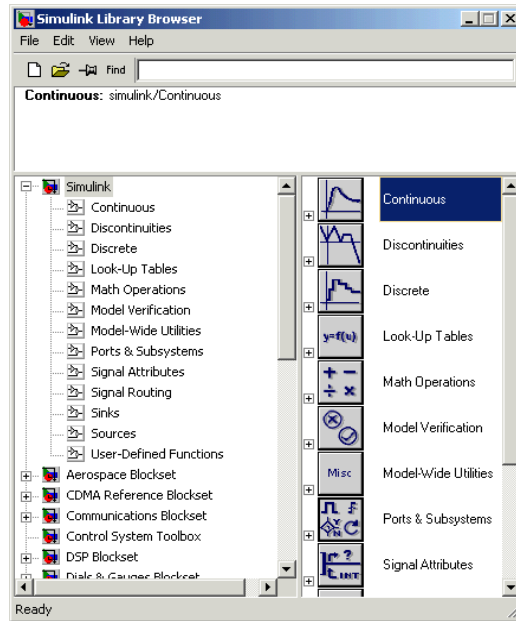
Following are the procedures for defining a model on Windows and UNIX platforms.

- “Defining a Model on Windows Platforms” on page 1-10
- “Defining a Model on UNIX Platforms” on page 1-15

Defining a Model on Windows Platforms

1 Start Simulink.

Click the  button in the MATLAB toolbar or enter `simulink` in the MATLAB Command Window. The Library Browser appears.



2 Open a new model.

Select **New -> Model** from the **File** menu in the Library Browser. A new model window appears on your screen.

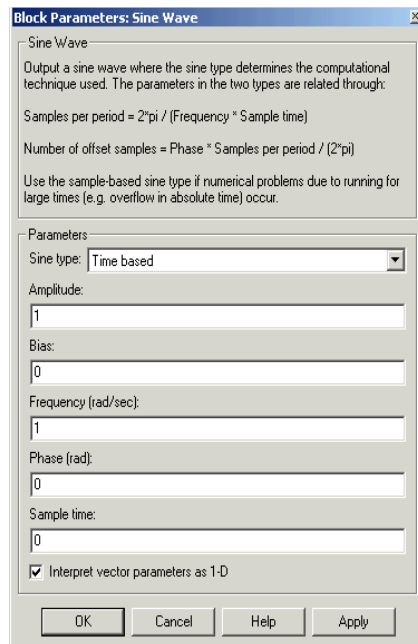
Alternate methods for creating a new model are by selecting **New** from the **File** menu in the Simulink model or by using the key sequence **Ctrl+N**.

- 3 Add a Sine Wave block to the model.
 - a Click **Sources** in the Library Browser to view the blocks in the Simulink Sources library.
 - b Drag the Sine Wave block from the Sources library into the new model window.
- 4 Add a Second Order Linear Actuator block to the model.
 - a Click the \boxplus symbol next to **Aerospace Blockset** in the Library Browser to expand the hierarchical list of the aerospace blocks.
 - b In the expanded list, click **Actuators** to view the blocks in the Actuator library.
 - c Drag the Second Order Linear Actuator block into the model window.
- 5 Add a Mux block to the model.
 - a Click **Signal Routing** in the Library Browser to view the blocks in the Simulink Signals & Systems library.
 - b Drag the Mux block from the Signal Routing library into the model window.
- 6 Add a Scope block to the model.
 - a Click **Sinks** in the Library Browser to view the blocks in the Simulink Sinks library.
 - b Drag the Scope block from the Sinks library into the model window.
- 7 Resize the Mux block in the model.
 - a Click the Mux block to select the block.
 - b Hold down the mouse button and drag a corner of the Mux block to change the size of the block.

- 8** Connect the blocks.
 - a** Position the pointer near the output port of the Sine Wave block. Hold down the mouse button and drag the line that appears until it touches the input port of the Second Order Linear Actuator block. Release the mouse button.
 - b** Using the same technique, connect the output of the Second Order Linear Actuator block to the second input port of the Mux block.
 - c** Using the same technique, connect the output of the Mux block to the input port of the Scope block.
 - d** Position the pointer near the first input port of the Mux block. Hold down the mouse button and drag the line that appears over the line from the output port of the Sine Wave block until double crosshairs appear. Release the mouse button. The lines are connected when a knot is present at their intersection.

- 9** Set the block parameters.
 - a** Double-click the Sine Wave block. The dialog box that appears allows you to set the block's parameters. *Parameters* are defining values that tell the block how to operate.

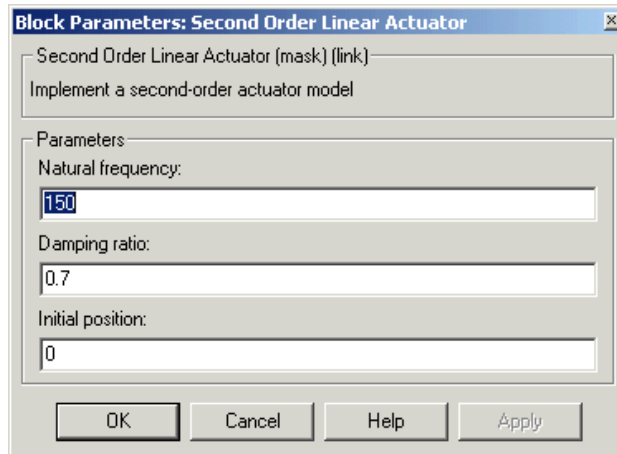
For this example, configure the block to generate a 10 rad/sec sine wave by entering 10 for the **Frequency** parameter. The sinusoid has the default amplitude of 1 and phase of 0 specified by the **Amplitude** and **Phase offset** parameters.



b Click OK.

- c Double-click the Second Order Linear Actuator block.

For this example, the actuator has the default natural frequency of 150 rad/sec, a damping ratio of 0.7, and an initial condition of 0 radians specified by the **Natural frequency**, **Damping ratio**, and **Initial condition** parameters.



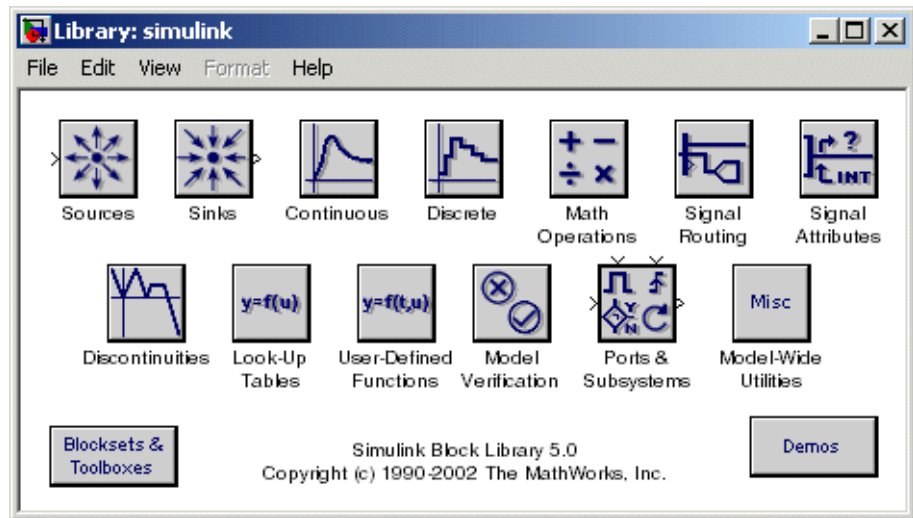
- d Click **OK**.

You can now move on to the model simulation phase. See “Model Simulation” on page 1-19.

Defining a Model on UNIX Platforms

1 Start Simulink.

Enter `simulink` in the MATLAB Command Window. The Simulink Library window appears.



2 Open a new model.

Select **New** -> **Model** from the **File** menu in the Simulink Library window. A new model window appears on your screen.

Alternate methods for creating a new model are by selecting **New** from the **File** menu in the Simulink model or by using the key sequence **Ctrl+N**.

- 3** Add a Sine Wave block to the model.
 - a** Double-click **Sources** in the Simulink Library window to view the blocks in the Simulink Sources library.
 - b** Drag the Sine Wave block from the Sources library into the new model window.

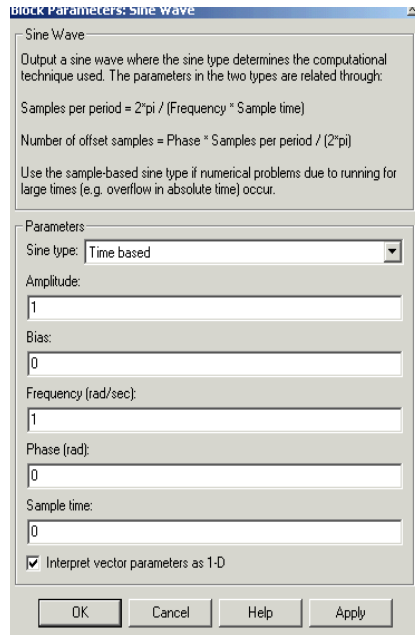
- 4** Add a Second Order Linear Actuator block to the model.
 - a** Double-click **Blocksets & Toolboxes** in the Simulink Library window. This opens the Blocksets and Toolboxes Library window.
 - b** Double-click **Aerospace Blockset** in the Blocksets and Toolboxes Library window. This opens the Aerospace Blockset block libraries.
 - c** In the Aerospace Blockset block libraries, click **Actuators** to view the blocks in the Actuator library.
 - d** Drag the Second Order Linear Actuator block into the model window.

- 5** Add a Mux block to the model.
 - a** Double-click **Signal Routing** in the Simulink Library to view the Signal Routing blocks.
 - b** Drag the Mux block from the Signal Routing library into the model window.

- 6** Add a Scope block to the model.
 - a** Double-click **Sinks** in the Simulink Library window to view the blocks in the Simulink Sinks library.
 - b** Drag the Scope block from the Sinks library into the model window.

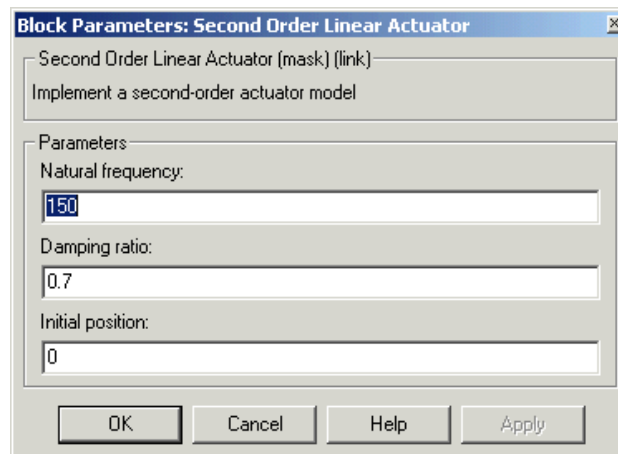
- 7 Resize the Mux block in the model.
 - a Click the Mux block to select the block.
 - b Hold down the mouse button and drag a corner of the Mux block to change the size of the block.
- 8 Connect the blocks.
 - a Position the pointer near the output port of the Sine Wave block. Hold down the mouse button and drag the line that appears until it touches the input port of the Second Order Linear Actuator block. Release the mouse button.
 - b Using the same technique, connect the output of the Second Order Linear Actuator block to the second input port of the Mux block.
 - c Using the same technique, connect the output of the Mux block to the input port of the Scope block.
 - d Position the pointer near the first input port of the Mux block. Hold down the mouse button and drag the line that appears over the line from the output port of the Sine Wave block until double crosshairs appear. Release the mouse button. The lines are connected when a knot is present at their intersection.
- 9 Set the block parameters.
 - a Double-click the Sine Wave block. The dialog box that appears allows you to set the block's parameters. *Parameters* are defining values that tell the block how to operate.

For this example, configure the block to generate a 10 rad/sec sine wave by entering 10 for the **Frequency** parameter. The sinusoid has the default amplitude of 1 and phase of 0 specified by the **Amplitude** and **Phase offset** parameters.



- b** Click **OK**.
- c** Double-click the **Second Order Linear Actuator** block.

For this example, the actuator has the default natural frequency of 150 rad/sec, a damping ratio of 0.7, and an initial condition of 0 radians specified by the **Natural frequency**, **Damping ratio**, and **Initial condition** parameters.




- d Click **OK**.

You can now move on to the model simulation phase. See “Model Simulation”.

Model Simulation

You can run the simulation block diagram that you built to see how the system behaves in time:

- 1 Double-click the Scope block if the Scope window is not already open on your screen. The Scope window appears.
- 2 Select **Start** from the **Simulation** menu in the block diagram window. The signal containing the 10 rad/sec sinusoid and the signal containing the actuator position are plotted on the scope.
- 3 Adjust the Scope block’s display.
 - a While the simulation is running, right-click the y -axis of the scope and select **Autoscale**. The vertical range of the scope is adjusted to better fit the signal.
 - b Click the **Properties** button  on the scope and enter 0.62832 for **Time range**. This resizes the scope’s time axis to display only one cycle of the signal.

- 4 Vary the Sine Wave block parameters.
 - a While the simulation is running, double-click the Sine Wave block to open it.
 - b Change the frequency of the sinusoid. Try entering 1 or 20 in the **Frequency** field. Click **Apply** after entering each new value and observe the changes on the scope.
- 5 Select **Stop** from the **Simulation** menu to stop the simulation.

Many parameters *cannot* be changed while a simulation is running. This is usually the case for parameters that directly or indirectly alter a signal's dimensions or sample rate. There are some parameters, however, like the Sine Wave **Frequency** parameter, that you can *tune* without terminating the simulation.

Running a Simulation from an M-File

You can also modify and run a Simulink simulation from within a MATLAB M-file. By doing this, you can automate the variation of model parameters to explore a large number of simulation conditions rapidly and efficiently. For information on how to do this, see “Running a Simulation Programmatically” in the Simulink documentation.

Learning More About Simulink and Aerospace Blockset

Here are more suggestions to help you get started with Simulink:

- Browse through the Simulink documentation to get complete exposure to all of Simulink's capabilities.
- Open the Simulink library as described in “Opening the Aerospace Blockset on Windows Platforms” on page 1-5. Build a few simple models using blocks from the Simulink and Aerospace Blockset libraries.
- Open some of the models in the Aerospace Blockset Demos library. Most of the advanced demos have blocks that you can double-click to get information about the algorithm or implementation. In each case, just select **Start** from the **Simulation** menu to run the simulation.

Case Study: Missile Guidance System

This chapter illustrates the process of designing and simulating a three-degrees-of-freedom missile guidance system using Simulink and the Aerospace Blockset. The following sections are included.

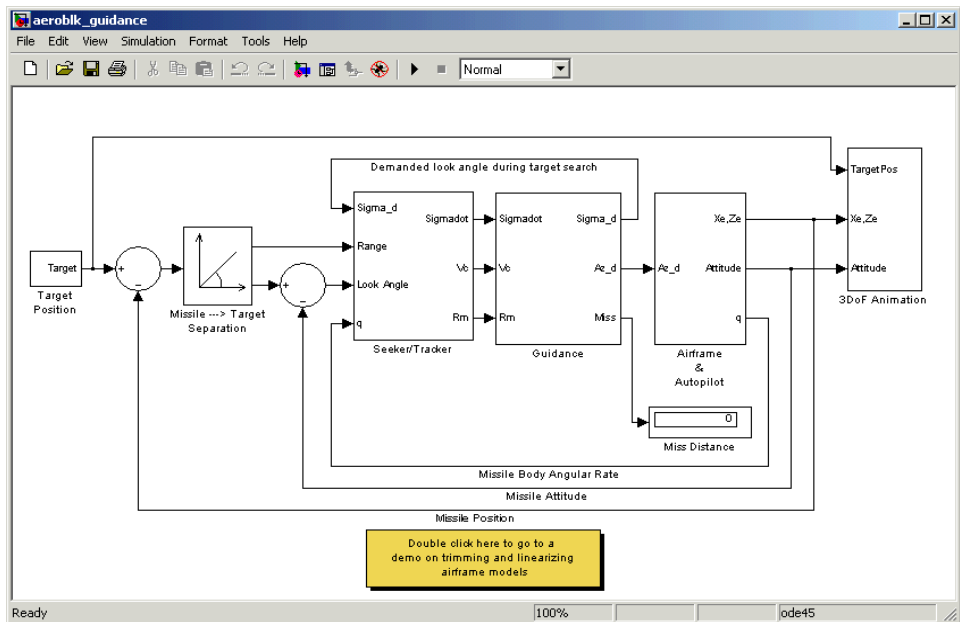
“Missile Guidance System Model” on page 2-2	Open the model that is used in this case study
“Modeling Airframe Dynamics” on page 2-3	Implement atmospheric equations and the equations of motion in the missile airframe.
“Modeling a Classical Three-Loop Autopilot” on page 2-10	Design the missile autopilot to control the acceleration normal to the missile body.
“Modeling the Homing Guidance Loop” on page 2-13	Design the homing guidance loop to track the target and generate the demands that are passed to the autopilot.
“Simulating the Missile Guidance System” on page 2-20	Simulate model and evaluate system performance.
“Extending the Model” on page 2-22	Examine a full six-degrees-of-freedom equations of motion representation.
“References” on page 2-23	Selected Bibliography

Missile Guidance System Model

To view the missile guidance system model, type the following in the MATLAB Command Window:

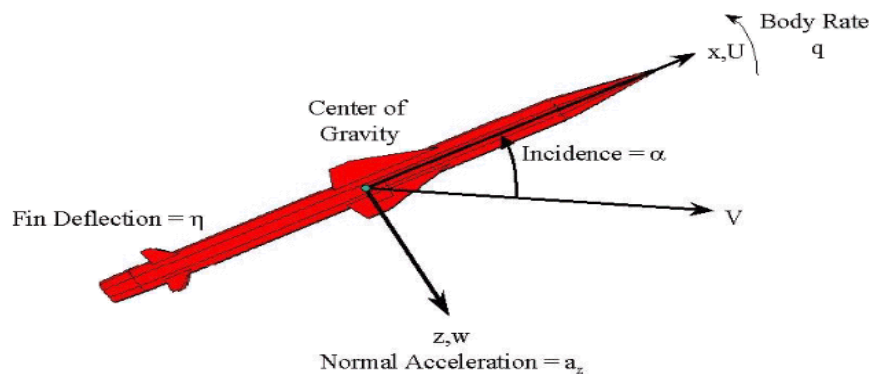
```
aeroblk_guidance
```

The missile airframe and autopilot are contained in the Airframe & Autopilot subsystem. The homing guidance loop consists of the Seeker/Tracker subsystem and the Guidance subsystem.



Modeling Airframe Dynamics

The model of the missile airframe used in this demonstration has been presented in a number of published papers (References [1], [2], and [3]) on the use of advanced control methods applied to missile autopilot design. The model represents a tail-controlled missile traveling between Mach 2 and Mach 4, at altitudes ranging between 3050 meters (10000 feet) and 18290 meters (60000 feet), and with typical angles of attack ranging between ± 20 degrees.



Missile Airframe Model

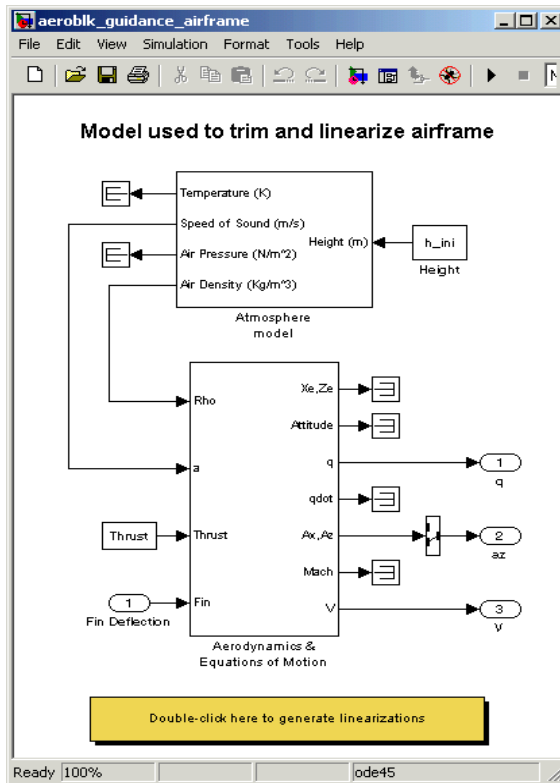
The core element of the model is a nonlinear representation of the rigid body dynamics of the airframe. The aerodynamic forces and moments acting on the missile body are generated from coefficients that are nonlinear functions of both incidence and Mach number. The model can easily be created in the Simulink environment using the Aerospace Blockset.

The model of the missile airframe consists of two main components:

- “ISA Atmosphere Model block” on page 2-5
Calculates the change in atmospheric conditions with changing altitude.
- “Aerodynamics & Equations of Motion Subsystem” on page 2-7
Calculates the magnitude of the forces and moments acting on the missile body, and integrates the equations of motion.

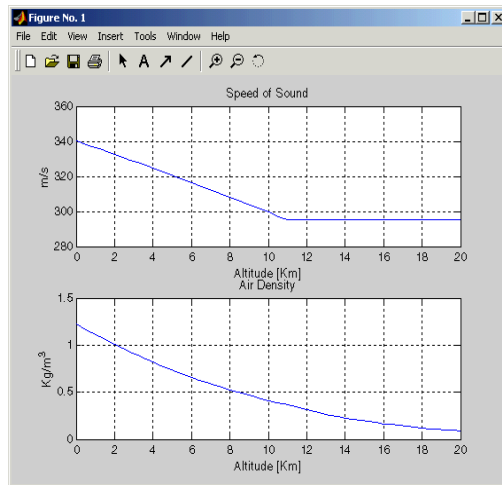
To view the missile airframe model, enter the following in the MATLAB Command Window:

```
aeroblk_guidance_airframe
```



ISA Atmosphere Model block

The ISA Atmosphere Model block is an approximation of the International Standard Atmosphere (ISA). This block consists of two sets of equations: one set of equations models the troposphere region and the other set of equations models the lower stratosphere region. The troposphere region lies between sea level and 11000 meters (36089 feet). It is assumed that there is a linear temperature drop with increasing altitude in the troposphere region. The lower stratosphere region ranges between 11000 meters (36089 feet) and 20000 meters (65617 feet). It is assumed that the temperature remains constant in the lower stratosphere region. The figure below displays how the speed of sound and the air density vary with altitude.



The following equations define the troposphere.

$$T = T_o - Lh$$

$$\rho = \rho_o \cdot \left(\frac{T}{T_o}\right)^{\frac{g}{LR} - 1}$$

$$P = P_o \cdot \left(\frac{T}{T_o}\right)^{\frac{g}{LR}}$$

$$a = \sqrt{\gamma RT}$$

The following equations define the lower stratosphere.

$$T = 216.7^\circ K$$

$$\rho = \rho_o \cdot \left(\frac{T}{T_o}\right)^{\frac{g}{LR}} \cdot e^{\frac{g}{RT}(11000-h)}$$

$$P = P_o \cdot \left(\frac{T}{T_o}\right)^{\frac{g}{LR}-1} \cdot e^{\frac{g}{RT}(11000-h)}$$

$$a = \sqrt{\gamma RT}$$

where

T_0 is the absolute temperature at mean sea level in degrees Kelvin.

ρ_0 is the air density at mean sea level in kg/m^3 .

P_0 is the static pressure at mean sea level in N/m^2 .

h is the altitude in m.

T is the absolute temperature at altitude, h , in degrees Kelvin.

ρ is the air density at altitude h in kg/m^3 .

P is the static pressure at altitude h in N/m^2 .

a is the speed of sound at altitude h in m/s^2 .

L is the lapse rate in degrees Kelvin/m.

R is the characteristic gas constant $\text{J/kg-degrees Kelvin}$.

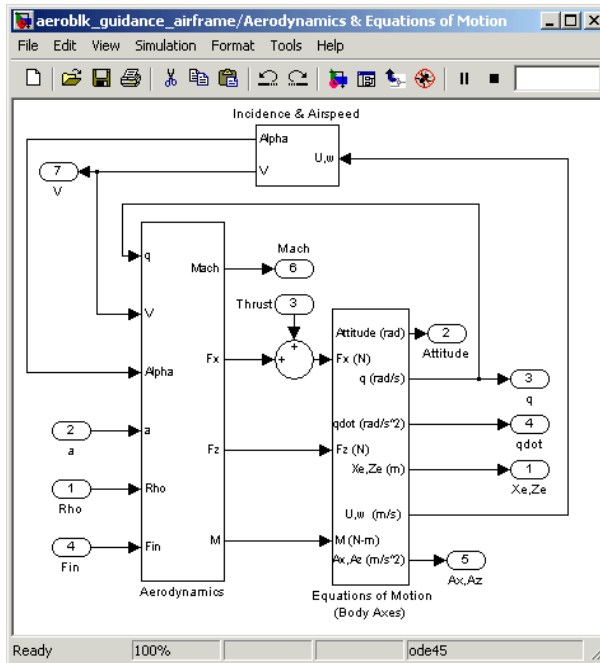
γ is the specific heat ratio.

g is the acceleration due to gravity in m/s^2 .

You can look under the mask of the ISA Atmosphere Model block to see how these equations are implemented in the model.

Aerodynamics & Equations of Motion Subsystem

The Aerodynamics & Equations of Motion subsystem generates the forces and moments applied to the missile in the body axes and integrates the equations of motion that define the linear and angular motion of the airframe. The aerodynamic coefficients are stored in data sets, and, during the simulation, the value at the current operating condition is determined by interpolation using the Interpolation (n-D) using PreLook-Up block.



The following are the three-degrees-of-freedom body axis equations of motion, which are defined in the Equations of Motion (Body Axes) block:

$$\dot{U} = (T + F_x)/m - qW - g \sin \theta$$

$$\dot{W} = F_z/m + qU + g \cos \theta$$

$$\dot{q} = M/I_{yy}$$

$$\dot{\theta} = q$$

The following are the aerodynamic forces and moments equations, which are defined in the Aerodynamics subsystem:

$$F_x = \bar{q}S_{ref}C_x(Mach, \alpha)$$

$$F_z = \bar{q}S_{ref}C_z(Mach, \alpha, \eta)$$

$$M = \bar{q}S_{ref}d_{ref}C_M(Mach, \alpha, \eta, q)$$

$$\bar{q} = \frac{1}{2}\rho V^2$$

The following are the stability axes variables, which are calculated in the Incidence & Airspeed block:

$$V = \sqrt{U^2 + W^2}$$

$$\alpha = \text{atan}(W/U)$$

where

θ is the attitude in radians.

q is the body rotation rate in rad/s.

M is the missile mass in Kg.

g is the acceleration due to gravity in m/s^2 .

I_{yy} is the moment of inertia about the y axis in $\text{Kg}\cdot\text{m}^2$.

W is the acceleration in the Z body axis in m/s^2 .

\dot{q} is the change in body rotation rate in rad/s^2 .

T is the thrust in the X body axis in N.

ρ is the air density in Kg/m^3 .

S_{ref} is the reference area in m^2 .

C_X is the coefficient of aerodynamic force in the X axis.

C_Z is the coefficient of aerodynamic force in the Z axis.

C_M is the coefficient of aerodynamic moment about the Y axis.

d_{ref} is the reference length in meters.

η is the fin angle in radians.

F_X is the aerodynamic force in the X body axis in N.

F_Z is the aerodynamic force in the Z body axis in N.

M is the aerodynamic moment along the Y body axis.

\bar{q} is the dynamic pressure in Pa.

V is the airspeed in m/s.

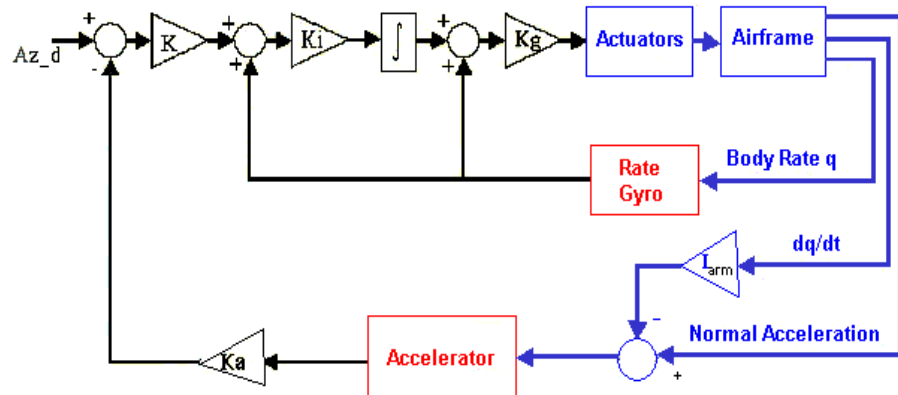
α is the incidence in radians.

U is the velocity in the X body axis in m/s.

W is the velocity in the Z body axis in m/s

Modeling a Classical Three-Loop Autopilot

The missile autopilot controls the acceleration normal to the missile body. In this case study, the autopilot structure is a three-loop design using measurements from an accelerometer placed ahead of the center of gravity and a rate gyro to provide additional damping. The figure below shows the classical configuration of an autopilot. The controller gains are scheduled on incidence and Mach number and they are tuned for robust performance at an altitude of 3050 meters (10000 feet).



Designing an autopilot entails the following:

- “Trimming and Linearizing an Airframe Model” on page 2-11
Models of the airframe pitch dynamics are derived for a number of trimmed flight conditions.
- “Autopilot Design” on page 2-12
Provides overview of the autopilot design process.

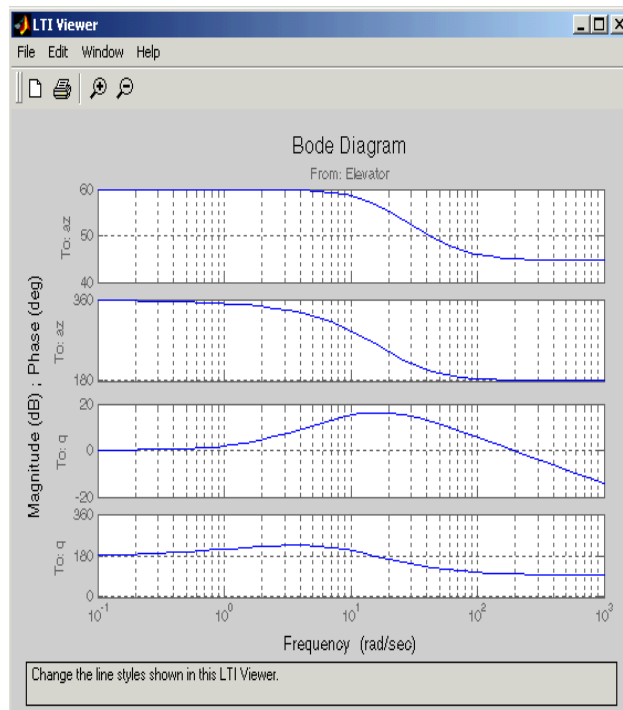
Trimming and Linearizing an Airframe Model

To design the autopilot using classical design techniques requires that linear models of the airframe pitch dynamics be derived for a number of trimmed flight conditions. MATLAB can determine the trim conditions and derive linear state-space models directly from the nonlinear Simulink model. This saves time and helps to validate the model. The functions provided by the Control System Toolbox allow the designer to visualize the behavior of the airframe open loop frequency (or time) responses.

The Airframe trim demo shows how to trim and linearize an airframe model. To run this demo, enter the following in the MATLAB Command Window:

```
aeroblk_lin_aero
```

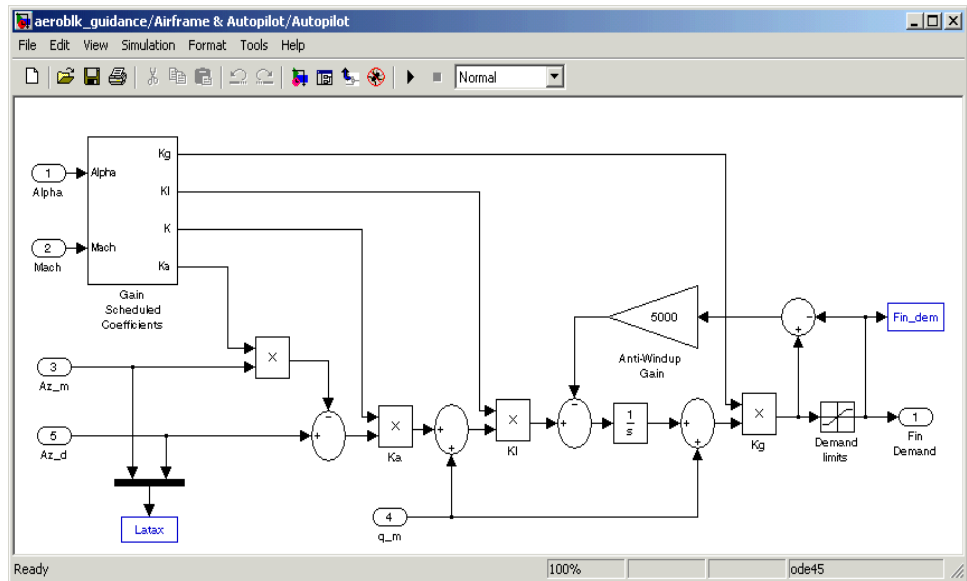
The output from this demo is a Bode diagram in the Control System Toolbox viewer.



Autopilot Design

Autopilot design can begin after the missile airframe has been linearized at a number of flight conditions. Typically, autopilot designs are carried out on a number of linear airframe models derived at varying flight conditions across the expected flight envelope. To implement the autopilot in the nonlinear model involves storing the autopilot gains in two-dimensional lookup tables, and incorporating an anti-windup gain to prevent integrator windup when the fin demands exceed the maximum limits. Testing the autopilot in the nonlinear Simulink model is the best way to demonstrate satisfactory performance in the presence of nonlinearities, such as actuator fin and rate limits and dynamically changing gains.

The Autopilot subsystem is an implementation of the classical three-loop autopilot design within Simulink.



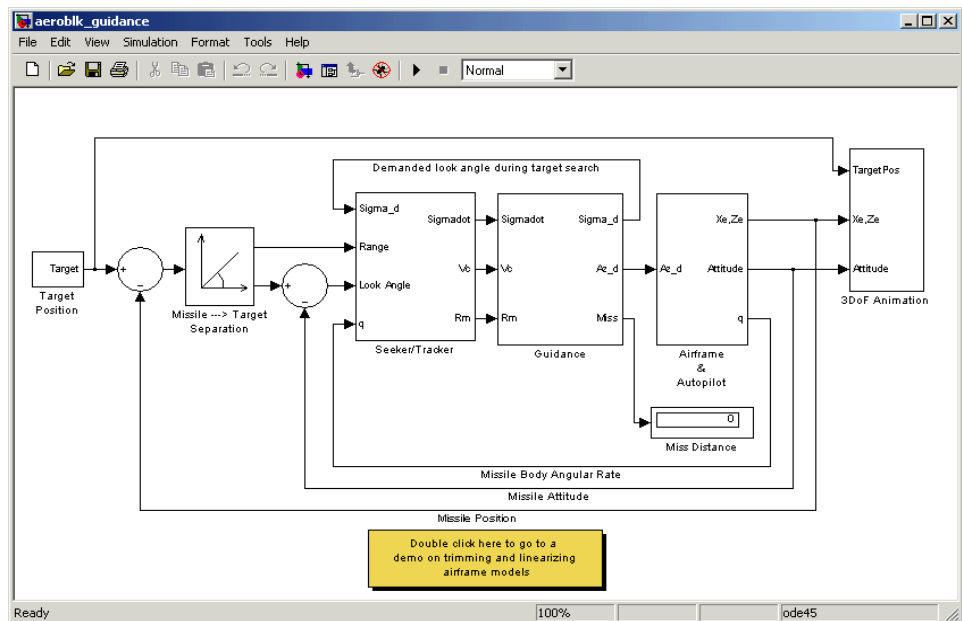
Modeling the Homing Guidance Loop

The complete homing guidance loop consists of these two subsystems:

- “Guidance Subsystem” on page 2-14
Generates the normal acceleration demands that are passed to the autopilot.
- “Seeker/Tracker Subsystem” on page 2-17

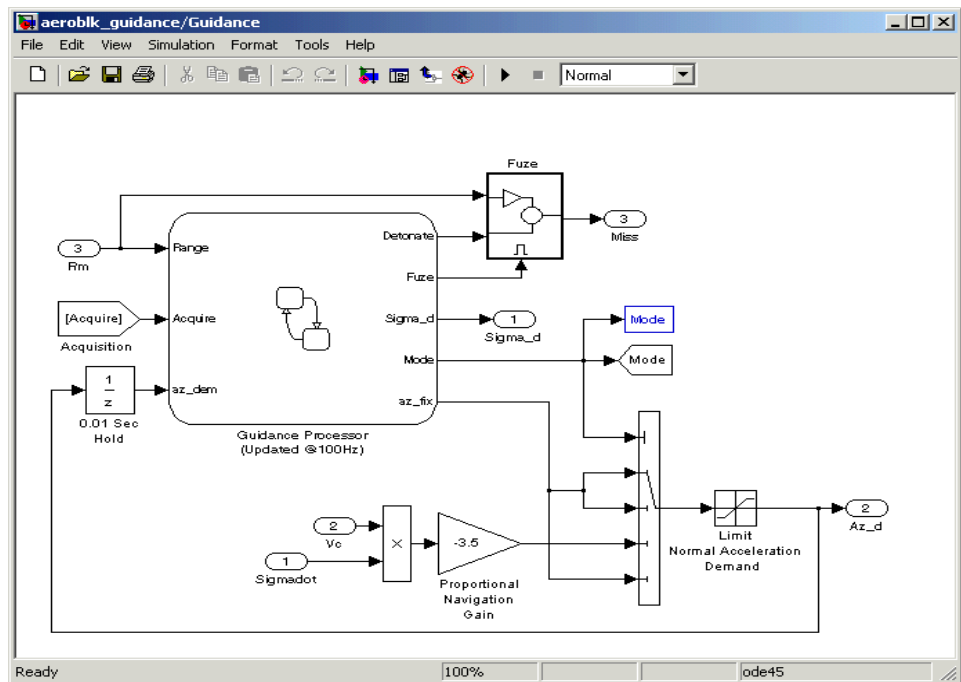
Returns measurements of the relative motion between the missile and the target.

The autopilot is now part of an inner loop within the overall homing guidance system. See Reference [4] for information on different types of guidance systems and on the analysis techniques that are used to quantify guidance loop performance.



Guidance Subsystem

The Guidance subsystem performs an initial search to locate the target's position, and then generates demands during closed-loop tracking. A Stateflow model is used to control the transfer between the different modes of these operations. Stateflow is the ideal tool for rapidly defining all the operational modes, both during normal operation and during unusual situations.

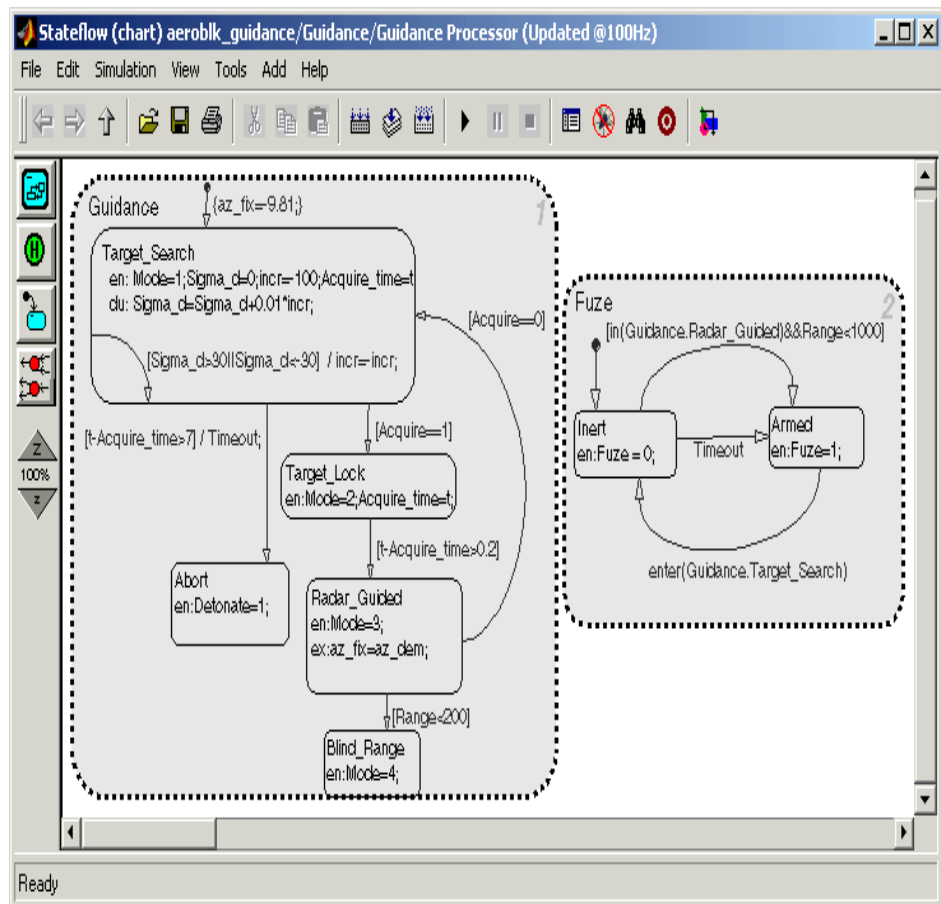


Guidance Processor Statechart

Mode switching is triggered by events generated in Simulink or in the Stateflow chart. The variable **Mode**, which is passed out to Simulink, is used to control the Simulink model's behavior and to determine the response of the Simulink model. For example, the Guidance Processor state chart, which is part of the Guidance subsystem, shows the actions the system takes in

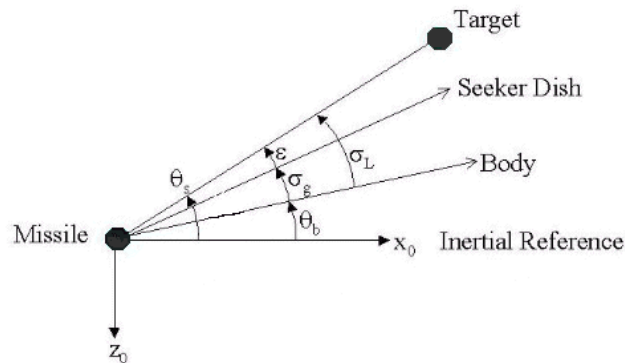
response to either a loss of lock on the target or a failure to acquire the target's position during the target search.

During the target search, this Stateflow state chart controls the tracker directly by sending demands to the seeker gimbals (Sigma_d). Target acquisition is flagged by the tracker once the target lies within the beam width of the seeker (Acquire) and, after a short delay, closed loop guidance starts.



Proportional Navigation Guidance Measurements

Once the seeker has acquired the target a Proportional Navigation Guidance (PNG) law is used to guide the missile until impact. This form of guidance law has been used in guided missiles since the 1950s, and can be applied to radar-, infrared-, or television-guided missiles. The navigation law requires measurements of the closing velocity between the missile and target, which for a radar-guided missile can be obtained using a Doppler tracking device, and an estimate for the rate of change of the inertial sight line angle.



Proportional Navigation Guidance Measurements

where

λ is navigation gain (> 2).

V_c is closing velocity.

θ_b is body attitude.

$\dot{\theta}_s$ is sight line rate.

σ_g is gimbal angle.

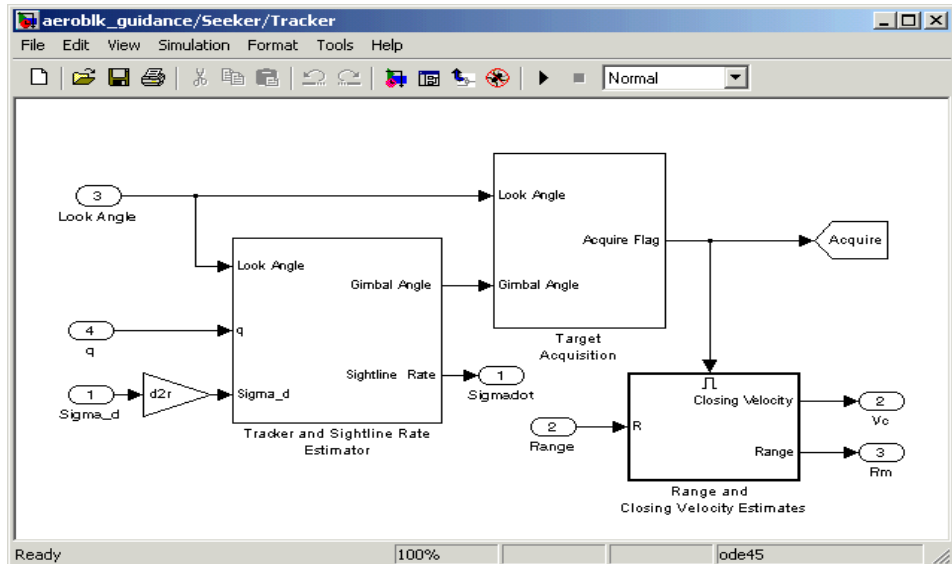
σ_L is look angle.

σ_d is dish angle.

$a_{z_dem} = \lambda V_c \dot{\theta}_s$, the demanded normal acceleration.

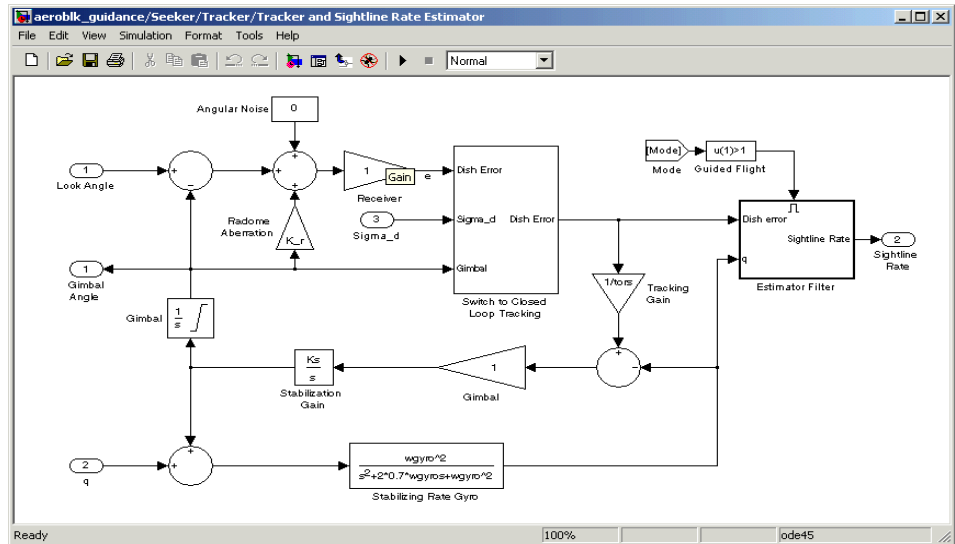
Seeker/Tracker Subsystem

The Seeker/Tracker subsystem controls the seeker gimbals to keep the seeker dish aligned with the target, and it provides the guidance law with an estimate of the sight line rate.



Tracker and Sightline Rate Estimator

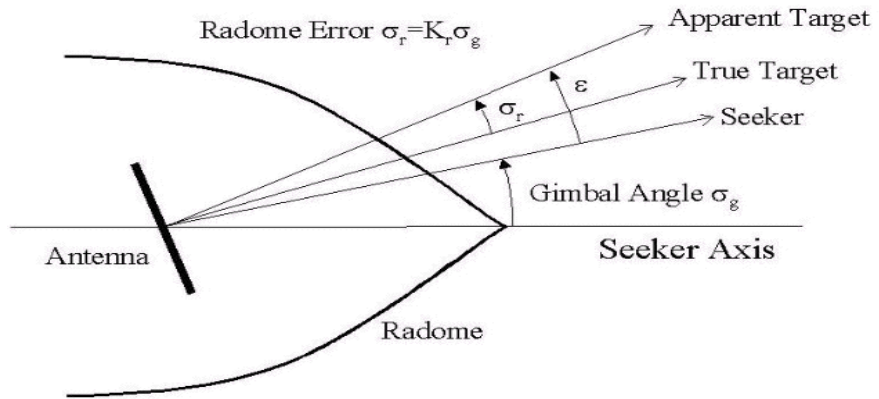
The Tracker and Sightline Rate Estimator, which is the most interesting subsystem of the Seeker/Tracker subsystem because of its complex error modeling, is shown below. It contains a number of feedback loops, estimated parameters, and parasitic effects for the homing guidance. The tracker loop time constant τ_{ors} is set to 0.05 seconds, which is a compromise between maximizing speed of response and keeping the noise transmission to within acceptable levels. The stabilization loop compensates for body rotation rates, and the gain K_s , which is the loop crossover frequency, is set as high as possible subject to the limitations of the bandwidth of the stabilizing rate gyro. The sight line rate estimate is a filtered value of the sum of the rate of change of the dish angle measured by the stabilizing rate gyro and an estimated value for the rate of change of the angular tracking error (e) measured by the receiver. In this demonstration the bandwidth of the estimator filter is set to half that of the bandwidth of the autopilot.



Radome Aberration.

Radome aberration is also modeled by the Tracker and Sightline Rate Estimator subsystem. Radome aberration is a parasitic feedback effect that is commonly modeled in radar-guided missile designs. This effect occurs because the shape of the protective covering over the seeker distorts the returning signal, and then gives a false reading of the look angle to the target. Generally the amount of distortion is a nonlinear function of the current gimbal angle, but a commonly used approximation is to assume a linear relationship between the gimbal angle and the magnitude of the distortion. Other parasitic effects, such

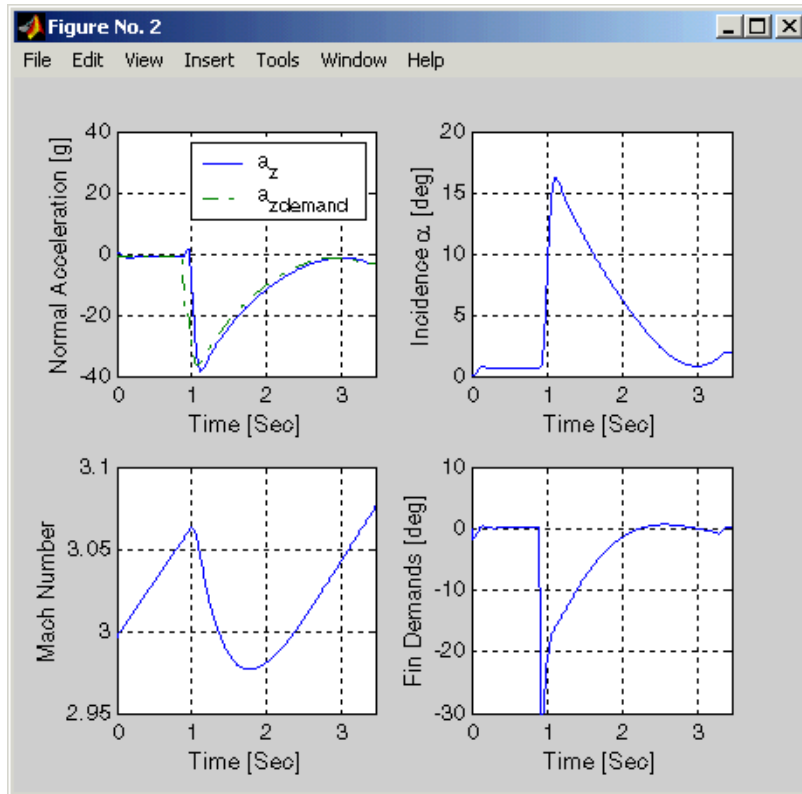
as sensitivity to normal acceleration in the rate gyros, are also often modeled to test the robustness of the target tracker and estimator filters.



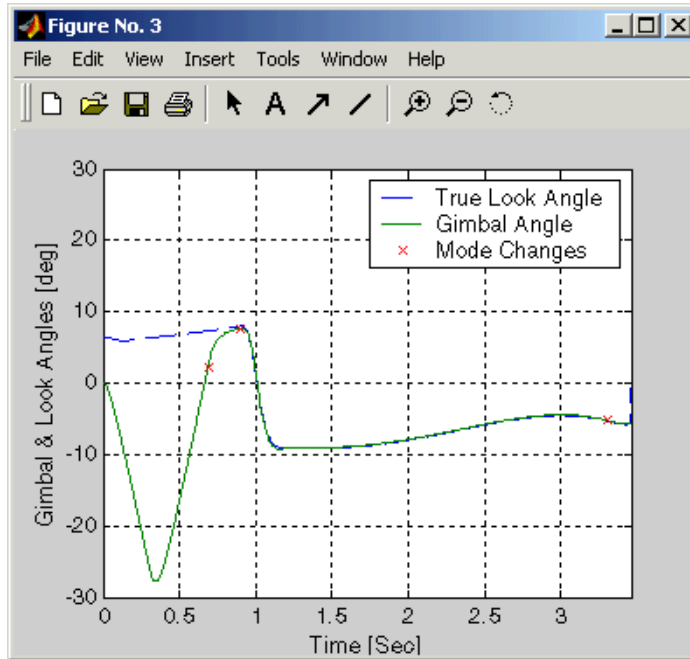
Radome Aberration Angles

Simulating the Missile Guidance System

Running the guidance simulation demonstrates the performance of the overall system. The target is defined to be traveling at a constant speed of 328 m/s on a reciprocal course to the initial missile heading and 500 meters above the initial missile position. The data, shown in the figure below, can be used to determine if the missile can withstand the flight demands and complete the mission.



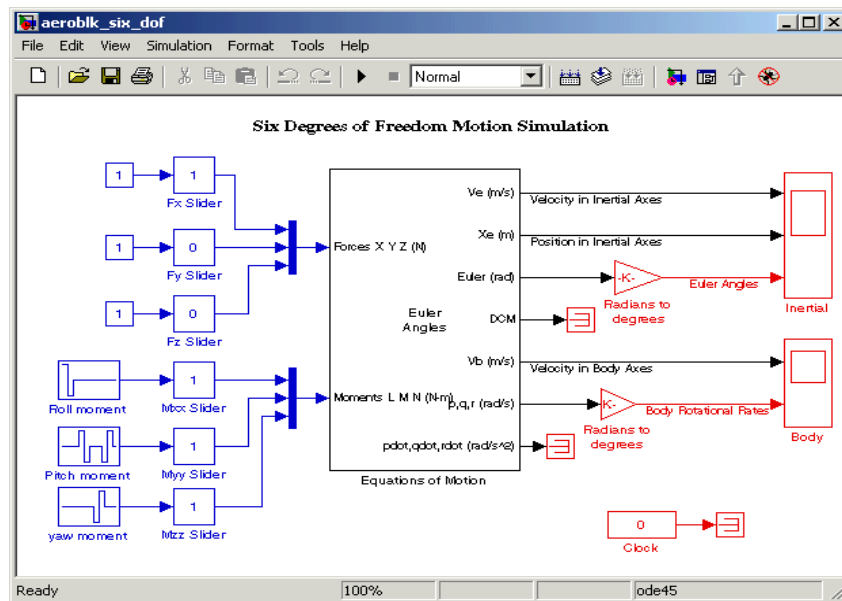
The simulation results show that target acquisition occurred 0.69 seconds into the engagement, with closed loop guidance starting after 0.89 seconds. Impact with the target occurred at 3.46 seconds, and the range to go at the point of closest approach was calculated to be 0.26 meters.



Extending the Model

Modeling the airframe and guidance loop in a single plane is only the start of the design process. Extending the model to a full six-degrees-of-freedom representation requires the implementation of the full equations of motion for a rigid body.

There are two blocks for integrating the equations of motion in the Aerospace Blockset. Both of these blocks use standard Simulink component blocks. The first implementation uses a quaternion to represent the angular orientation of the body in space, which is ideal for simulations where the standard Euler angle definitions become singular as the pitch attitude tends to ± 90 degrees. The second implementation uses the standard Euler angle equations of motion, which is ideal when using the model to obtain trim conditions and linear airframe models. A model containing one of the six-degrees-of-freedom equations of motion blocks is shown below.



References

- 1 Bennani, S., D.M.C. Willemsen, and C.W. Scherer, "Robust LPV control with bounded parameter rates," AIAA-97-3641, August 1997.
- 2 Mracek, C. P and J.R. Cloutier, "Full Envelope Missile Longitudinal Autopilot Design Using the State-Dependent Riccati Equation Method," AIAA-97-3767, August 1997.
- 3 Shamma, J.S. and J.R. Cloutier, "Gain-Scheduled Missile Autopilot Design Using Linear Parameter Varying Transformations," *Journal of Guidance, Control and Dynamics*, Vol. 16, No. 2, March-April 1993.
- 4 Lin, Ching-Fang, "Modern Navigation, Guidance, and Control Processing," Volume 2, ISBN 0-13-596230-7, Prentice Hall, 1991.

Block Reference

Blocks — By Category (p. 3-2)

Provides tables of Aerospace Blockset blocks by category.

Blocks — Alphabetical List (p. 3-9)

Provides an alphabetical list of Aerospace Blockset blocks.

Blocks – By Category

The Aerospace Blockset's main library, **aerolib**, organizes its blocks into libraries according to their behavior. The **aerolib** window displays the block library icons and names:

Actuator Library Blocks	Contain actuator models.
Animation Library Blocks	Contain blocks that provide 3-D animation during simulation.
Environment/Atmosphere Library Blocks	Contain atmosphere models.
Environment/Gravity Library Blocks	Contain gravity models.
Environment/Wind Library Blocks	Contain wind models.
Equations of Motion/3DoF Library Blocks	Contain three-degrees-of-freedom equations of motion blocks.
Equations of Motion/6DoF Library Blocks	Contain six-degrees-of-freedom equations of motion blocks.
GNC Library Blocks	Contain gain scheduling blocks.
Propulsion Library Blocks	Contain simple propulsion system models.
Transformation/Axes Library Blocks	Contain blocks that convert reference axes.
Transformations/Units Library Blocks	Contain blocks that convert unit axes.

Actuator Library Blocks

Second Order Linear Actuator	Implement a second-order linear actuator
Second Order Nonlinear Actuator	Implement a second-order actuator with rate and deflection limits

Animation Library Blocks

3DoF Animation	Create a 3-D Handle Graphics animation of a three-degrees-of-freedom object
6DoF Animation	Create a 3-D Handle Graphics animation of a six-degrees-of-freedom object

Environment/Atmosphere Library Blocks

COESA Atmosphere Model	Implement the 1976 COESA lower atmosphere
ISA Atmosphere Model	Implement the International Standard Atmosphere (ISA)

Environment/Gravity Library Blocks

WGS84 Gravity Model	Implement the 1984 World Geodetic System representation of Earth's gravity
---------------------	----------------------------------------------------------------------------

Environment/Wind Library Blocks

Discrete Wind Gust Model	Generate discrete wind gust
Dryden Wind Turbulence Model	Generate wind turbulence with the Dryden velocity spectra
Wind Shear Model	Calculate wind shear conditions

Equations of Motion/3DoF Library Blocks

Equations of Motion	Implement three-degrees-of-freedom equations of motion
Incidence & Airspeed	Calculate incidence and air speed

Equations of Motion/6DoF Library Blocks

6DoF (Euler Angles)	Implement an Euler angle representation of six-degrees-of-freedom equations of motion
6DoF (Quaternion)	Implement a quaternion representation of six-degrees-of-freedom equations of motion

GNC Library Blocks

1D Controller [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller depending on one scheduling parameter
1D Controller Blend $u=(1-L).K1.y+L.K2.y$	Implement a 1-D vector of state-space controllers by linear interpolation of their outputs
1D Observer Form [A(v),B(v),C(v),F(v),H(v)]	Implement a gain-scheduled state-space controller in an observer form depending on one scheduling parameter
1D Self-Conditioned [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller in a self-conditioned form
2D Controller [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller depending on two scheduling parameters
2D Controller Blend	Implement a 2-D vector of state-space controllers by linear interpolation of their outputs

2D Observer Form [A(v),B(v),C(v),F(v),H(v)]	Implement a gain-scheduled state-space controller in an observer form depending on two scheduling parameters
2D Self-Conditioned [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller in a self-conditioned form
3D Controller [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller depending on three scheduling parameters
3D Observer Form [A(v),B(v),C(v),F(v),H(v)]	Implement a gain-scheduled state-space controller in an observer form depending on three scheduling parameters
3D Self-Conditioned [A(v),B(v),C(v),D(v)]	Implement a gain-scheduled state-space controller in a self-conditioned form
Gain Scheduled Lead-Lag	Implement a first-order lead-lag with gain-scheduled coefficients
Interpolate Matrix(x)	Return an interpolated matrix for given input x
Interpolate Matrix(x,y)	Return an interpolated matrix for given inputs x and y
Interpolate Matrix(x,y,z)	Return an interpolated matrix for given inputs x, y, and z
Self-Conditioned [A,B,C,D]	Implement a state-space controller in a self-conditioned form

Propulsion Library Blocks

Turbofan Engine System Implement a first-order representation of a turbofan engine with controller

Transformation/Axes Library Blocks

3x3 Cross Product Calculate the cross product of two 3-by-1 vectors

Direction Cosine Matrix to Euler Angles Convert direction cosine matrix to Euler angles

Direction Cosine Matrix to Quaternions Convert direction cosine matrix to quaternion vector

Euler Angles to Direction Cosine Matrix Convert Euler angles to direction cosine matrix

Euler Angles to Quaternions Convert Euler angles to quaternion vector

Quaternions to Direction Cosine Matrix Convert quaternion vector to direction cosine matrix

Quaternions to Euler Angles Convert quaternion vector to Euler angles

Transformations/Units Library Blocks

Acceleration Conversion	Convert from acceleration units to desired acceleration units
Angle Conversion	Convert from angle units to desired angle units
Angular Acceleration Conversion	Convert from angular acceleration units to desired angular acceleration units
Angular Velocity Conversion	Convert from angular velocity units to desired angular velocity units
Density Conversion	Convert from density units to desired density units
Force Conversion	Convert from force units to desired force units
Length Conversion	Convert from length units to desired length units
Mass Conversion	Convert from mass units to desired mass units
Pressure Conversion	Convert from pressure units to desired pressure units
Temperature Conversion	Convert from temperature units to desired temperature units
Velocity Conversion	Convert from velocity units to desired velocity units

Block Reference Page Description

Blocks appear in alphabetical order and contain the following information:

- The block name, icon, and block library that contains the block
- The purpose of the block
- A description of the block's use
- The block dialog box and parameters
- Additional information, as it applies to the block:
 - Inputs and outputs descriptions
 - Assumptions and limitations to the block's use
 - Examples using the block
 - References to other documents
- A “See Also” of related blocks

Blocks — Alphabetical List

1D Controller $[A(v),B(v),C(v),D(v)]$	3-12
1D Controller Blend $u=(1-L).K1.y+L.K2.y$	3-15
1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$	3-18
1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$	3-21
2D Controller $[A(v),B(v),C(v),D(v)]$	3-25
2D Controller Blend	3-28
2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$	3-32
2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$	3-36
3D Controller $[A(v),B(v),C(v),D(v)]$	3-40
3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$	3-43
3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$	3-47
3DoF Animation	3-51
3x3 Cross Product	3-53
6DoF Animation	3-54
6DoF (Euler Angles)	3-56
6DoF (Quaternion)	3-60
Acceleration Conversion	3-64
Angle Conversion	3-66
Angular Acceleration Conversion	3-68
Angular Velocity Conversion	3-70
COESA Atmosphere Model	3-72
Density Conversion	3-74
Direction Cosine Matrix to Euler Angles	3-76
Direction Cosine Matrix to Quaternions	3-78
Discrete Wind Gust Model	3-80
Dryden Wind Turbulence Model	3-83
Euler Angles to Direction Cosine Matrix	3-91
Euler Angles to Quaternions	3-93
Equations of Motion	3-95
Force Conversion	3-99
Gain Scheduled Lead-Lag	3-101
Incidence & Airspeed	3-102
Interpolate Matrix(x)	3-103
Interpolate Matrix(x,y)	3-105
Interpolate Matrix(x,y,z)	3-107

ISA Atmosphere Model	3-110
Length Conversion	3-111
Mass Conversion	3-113
Pressure Conversion	3-115
Quaternions to Direction Cosine Matrix	3-117
Quaternions to Euler Angles	3-119
Second Order Linear Actuator	3-121
Second Order Nonlinear Actuator	3-123
Self-Conditioned [A,B,C,D]	3-125
Temperature Conversion	3-130
Turbofan Engine System	3-132
Velocity Conversion	3-135
WGS84 Gravity Model	3-137
Wind Shear Model	3-141

1D Controller [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller depending on one scheduling parameter

Library GNC

Description The 1D Controller [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations



$$\begin{aligned} \dot{x} &= A(v)x + B(v)y \\ u &= C(v)x + D(v)y \end{aligned}$$

where v is a parameter over which A , B , C , and D are defined. This type of controller scheduling assumes that the matrices A , B , C , and D vary smoothly as a function of v , which is often the case in aerospace applications.

Dialog Box

Block Parameters: 1D Controller [A(v),B(v),C(v),D(v)]

StateSpaceABCD-1D (mask)
Implement a state-space controller [A,B,C,D] where A, B, C, and D depend on one scheduling parameter, v.

Parameters

A-matrix{v}:
A1

B-matrix{v}:
B1

C-matrix{v}:
C1

D-matrix{v}:
D1

Scheduling variable breakpoints:
v_vec

Initial state, x_initial:
0

OK Cancel Help Apply

1D Controller $[A(v), B(v), C(v), D(v)]$

A-matrix(v)

A-matrix of the state-space implementation. In the case of 1D scheduling, the A-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the A-matrix corresponding to the first entry of v is the identity matrix, then $A(:, :, 1) = [1 \ 0; 0 \ 1];$.

B-matrix(v)

B-matrix of the state-space implementation. In the case of 1D scheduling, the B-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the B-matrix corresponding to the first entry of v is the identity matrix, then $B(:, :, 1) = [1 \ 0; 0 \ 1];$.

C-matrix(v)

C-matrix of the state-space implementation. In the case of 1D scheduling, the C-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the C-matrix corresponding to the first entry of v is the identity matrix, then $C(:, :, 1) = [1 \ 0; 0 \ 1];$.

D-matrix(v)

D-matrix of the state-space implementation. In the case of 1D scheduling, the D-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the D-matrix corresponding to the first entry of v is the identity matrix, then $D(:, :, 1) = [1 \ 0; 0 \ 1];$.

Scheduling variable breakpoints

Vector of the breakpoints for the scheduling variable. The length of v should be same as the size of the third dimension of A, B, C, and D.

Initial state, x_{initial}

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Inputs and Outputs

The first input is the measurements.

The second input is the scheduling variable conforming to the dimensions of the state-space matrices.

1D Controller $[A(v),B(v),C(v),D(v)]$

The output is the actuator demands.

Assumptions and Limitations

If the scheduling parameter input to the block goes out of range, then it is clipped; i.e., the state-space matrices are not interpolated out of range.

Examples

See the autopilot in the `aerob1k_HL20.mdl` demo for an example of this block.

See Also

1D Controller Blend $u=(1-L).K1.y+L.K2.y$
1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
2D Controller $[A(v),B(v),C(v),D(v)]$
3D Controller $[A(v),B(v),C(v),D(v)]$

1D Controller Blend $u=(1-L).K1.y+L.K2.y$

Purpose Implement a 1-D vector of state-space controllers by linear interpolation of their outputs

Library GNC

Description



The 1D Controller Blend $u=(1-L).K1.y+L.K2.y$ block implements an array of state-space controller designs. The controllers are run in parallel, and their outputs interpolated according to the current flight condition or operating point. The advantage of this implementation approach is that the state-space matrices A , B , C , and D for the individual controller designs do not need to vary smoothly from one design point to the next.

For example, suppose two controllers are designed at two operating points $v=v_{min}$ and $v=v_{max}$. The 1D Controller Blend block implements

$$\begin{aligned}\dot{x}_1 &= A_1x_1 + B_1y \\ u_1 &= C_1x_1 + D_1y \\ \dot{x}_2 &= A_2x_2 + B_2y \\ u_2 &= C_2x_2 + D_2y \\ u &= (1-\lambda)u_1 + \lambda u_2\end{aligned}$$

$$\lambda = \begin{cases} 0 & v < v_{min} \\ \frac{v - v_{min}}{v_{max} - v_{min}} & v_{min} \leq v \leq v_{max} \\ 1 & v > v_{max} \end{cases}$$

For longer arrays of design points, the blocks only implement nearest neighbor designs. For the 1D Controller Blend block, at any given instant in time, three controller designs are being updated. This reduces computational requirements.

As the value of the scheduling parameter varies and the index of the controllers that need to be run changes, the states of the oncoming controller are initialized by using the self-conditioned form as defined for the Self-Conditioned [A,B,C,D] block.

1D Controller Blend $u=(1-L).K1.y+L.K2.y$

Dialog Box

Block Parameters: 1D Controller Blend: $u=[1-L].K1.y+L.K2.y$

Blend-1D (mask)
Blend between outputs of a 1-D vector of state-space controllers. All controllers must have the same state dimension.

Parameters

A-matrix(v):
A

B-matrix(v):
B

C-matrix(v):
C

D-matrix(v):
D

Scheduling variable breakpoints:
v_vec

Initial state, x_initial:
0

Poles of $A(v)-H(v)*C(v) = [w1 \dots wn]$:
[-5 -2]

OK Cancel Help Apply

A-matrix(v)

A-matrix of the state-space implementation. In the case of 1D blending, the A-matrix should have three dimensions, the last one corresponding to scheduling variable v . Hence, for example, if the A-matrix corresponding to the first entry of v is the identity matrix, then $A(:, :, 1) = [1 \ 0; 0 \ 1]$;

B-matrix(v)

B-matrix of the state-space implementation.

C-matrix(v)

C-matrix of the state-space implementation.

D-matrix(v)

D-matrix of the state-space implementation.

Scheduling variable breakpoints

Vector of the breakpoints for the scheduling variable. The length of v should be same as the size of the third dimension of A, B, C, and D.

1D Controller Blend $u=(1-L).K1.y+L.K2.y$

Initial state, $x_initial$

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A .

Poles of $A(v)-H(v)*C(v)$

For oncoming controllers, an observer-like structure is used to ensure that the controller output tracks the current block output, u . The poles of the observer are defined in this dialog box as a vector, the number of poles being equal to the dimension of the A -matrix. Poles that are too fast result in sensor noise propagation, and poles that are too slow result in the failure of the controller output to track u .

Inputs and Outputs

The first input is the measurements.

The second input is the scheduling variable conforming to the dimensions of the state-space matrices.

The output is the actuator demands.

Assumptions and Limitations

This block requires the Control Systems Toolbox.

Examples

See the autopilot in the `aeroblk_HL20.mdl` demo for an example of this block in use.

References

Hyde, R.A., "H-infinity Aerospace Control Design - A VSTOL Flight Application," Springer Verlag, *Advances in Industrial Control Series*, 1995. ISBN 3-540-19960-8. See Chapter 5.

See Also

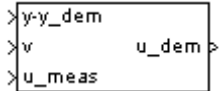
1D Controller $[A(v),B(v),C(v),D(v)]$
1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
2D Controller Blend

1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

Purpose Implement a gain-scheduled state-space controller in an observer form depending on one scheduling parameter

Library GNC

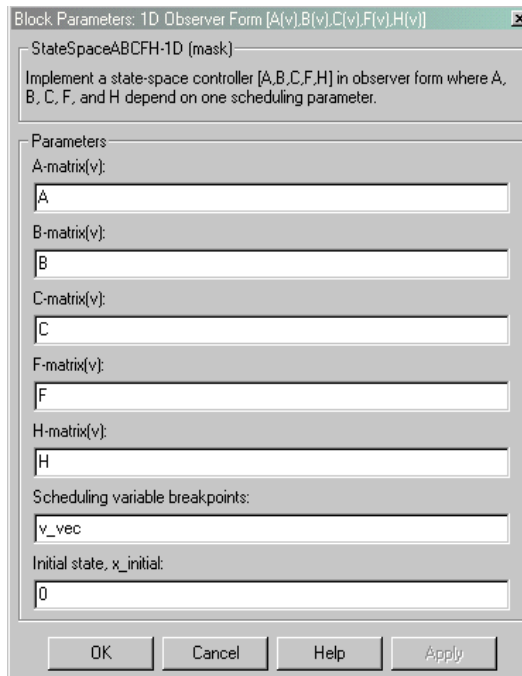
Description The 1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$ block implements a gain-scheduled state-space controller defined in the following observer for:



$$\begin{aligned}\dot{x} &= (A(v) + H(v)C(v))x + B(v)u_{meas} + H(v)(y - y_{dem}) \\ u_{dem} &= F(v)x\end{aligned}$$

The main application of this blocks is to implement a controller designed using H-infinity loop-shaping, one of the design methods supported by the μ -Analysis and Synthesis Toolbox.

Dialog Box



1D Observer Form $[A(v), B(v), C(v), F(v), H(v)]$

A-matrix(v)

A-matrix of the state-space implementation. The A-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the A-matrix corresponding to the first entry of v is the identity matrix, then $A(:, :, 1) = [1 \ 0; 0 \ 1];$

B-matrix(v)

B-matrix of the state-space implementation. The B-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the B-matrix corresponding to the first entry of v is the identity matrix, then $B(:, :, 1) = [1 \ 0; 0 \ 1];$

C-matrix(v)

C-matrix of the state-space implementation. The C-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the C-matrix corresponding to the first entry of v is the identity matrix, then $C(:, :, 1) = [1 \ 0; 0 \ 1];$

F-matrix(v)

State-feedback matrix. The F-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the F-matrix corresponding to the first entry of v is the identity matrix, then $F(:, :, 1) = [1 \ 0; 0 \ 1];$

H-matrix(v)

Observer (output injection) matrix. The H-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the H-matrix corresponding to the first entry of v is the identity matrix, then $H(:, :, 1) = [1 \ 0; 0 \ 1];$

Scheduling variable breakpoints

Vector of the breakpoints for the scheduling variable. The length of v should be same as the size of the third dimension of A, B, C, F, and H.

Initial state, x_{initial}

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Inputs and Outputs

The first input is the set-point error.

1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

The second input is the scheduling variable.

The third input is measured actuator position.

The output is the actuator demands.

Assumptions and Limitations If the scheduling parameter input to the block goes out of range, then it is clipped; i.e., the state-space matrices are not interpolated out of range.

Examples See the observer implementation within the `aerob1k_HL20.mdl` demo.

References Hyde, R.A., “H-infinity Aerospace Control Design - A VSTOL Flight Application,” Springer Verlag, *Advances in Industrial Control Series*, 1995. ISBN 3-540-19960-8. See Chapter 6.

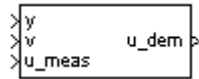
See Also 1D Controller $[A(v),B(v),C(v),D(v)]$
1D Controller Blend $u=(1-L).K1.y+L.K2.y$
1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

1D Self-Conditioned [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller in a self-conditioned form

Library GNC

Description



The 1D Self-Conditioned [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations

$$\dot{x} = A(v)x + B(v)y$$

$$u = C(v)x + D(v)y$$

in the self-conditioned form

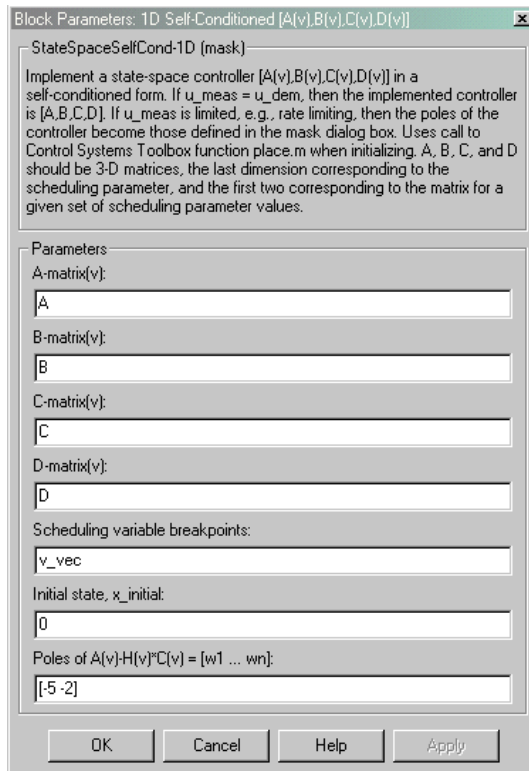
$$\dot{z} = (A(v) - H(v)C(v))z + (B(v) - H(v)D(v))e + H(v)u_{meas}$$

$$u_{dem} = C(v)z + D(v)e$$

For the rationale behind this self-conditioned implementation, refer to the Self-Conditioned [A,B,C,D] block reference. This block implements a gain-scheduled version of the Self-Conditioned [A,B,C,D] block, v being the parameter over which A, B, C, and D are defined. This type of controller scheduling assumes that the matrices A, B, C, and D vary smoothly as a function of v , which is often the case in aerospace applications.

1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

Dialog Box



A-matrix(v)

A-matrix of the state-space implementation. The A-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the A-matrix corresponding to the first entry of v is the identity matrix, then $A(:, :, 1) = [1 \ 0; 0 \ 1]$;

B-matrix(v)

B-matrix of the state-space implementation. The B-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the B-matrix corresponding to the first entry of v is the identity matrix, then $B(:, :, 1) = [1 \ 0; 0 \ 1]$;

1D Self-Conditioned $[A(v), B(v), C(v), D(v)]$

C-matrix(v)

C-matrix of the state-space implementation. The C-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the C-matrix corresponding to the first entry of v is the identity matrix, then $C(:, :, 1) = [1 \ 0; 0 \ 1]$;

D-matrix(v)

D-matrix of the state-space implementation. The D-matrix should have three dimensions, the last one corresponding to the scheduling variable v . Hence, for example, if the D-matrix corresponding to the first entry of v is the identity matrix, then $D(:, :, 1) = [1 \ 0; 0 \ 1]$;

Scheduling variable breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v should be same as the size of the third dimension of A, B, C, and D.

Initial state, x_{initial}

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Poles of $A(v) - H(v) * C(v)$

Vector of the desired poles of $A - HC$. Note that the poles are assigned to the same locations for all values of the scheduling parameter v . Hence the number of pole locations defined should be equal to the length of the first dimension of the A-matrix.

Inputs and Outputs

The first input is the measurements.

The second input is the scheduling variable conforming to the dimensions of the state-space matrices.

The third input is the measured actuator position.

The output is the actuator demands.

Assumptions and Limitations

If the scheduling parameter input to the block goes out of range, then it is clipped; i.e., the state-space matrices are not interpolated out of range.

This block requires the Control Systems Toolbox.

1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

References

The algorithm used to determine the matrix H is defined in Kautsky, Nichols, and Van Dooren, "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, No. 5, pages 1129-1155, 1985.

See Also

1D Controller $[A(v),B(v),C(v),D(v)]$
1D Controller Blend $u=(1-L).K1.y+L.K2.y$
1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

2D Controller [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller depending on two scheduling parameters

Library GNC

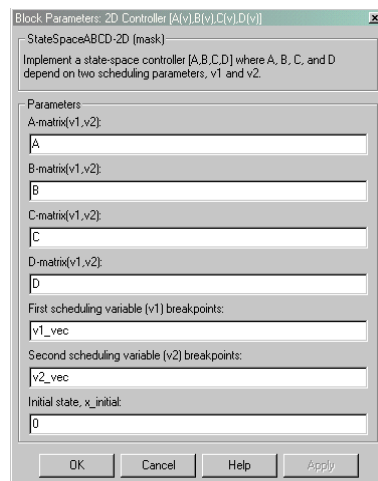
Description The 2D Controller [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations



$$\begin{aligned} \dot{x} &= A(v)x + B(v)y \\ u &= C(v)x + D(v)y \end{aligned}$$

where v is a vector of parameters over which A, B, C, and D are defined. This type of controller scheduling assumes that the matrices A, B, C, and D vary smoothly as a function of v , which is often the case in aerospace applications.

Dialog Box



A-matrix(v1,v2)

A-matrix of the state-space implementation. In the case of 2D scheduling, the A-matrix should have four dimensions, the last two corresponding to scheduling variables $v1$ and $v2$. Hence, for example, if the A-matrix corresponding to the first entry of $v1$ and first entry of $v2$ is the identity matrix, then $A(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

B-matrix(v1,v2)

B-matrix of the state-space implementation. In the case of 2D scheduling, the B-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the B-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $B(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

C-matrix(v1,v2)

C-matrix of the state-space implementation. In the case of 2D scheduling, the C-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the C-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $C(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

D-matrix(v1,v2)

D-matrix of the state-space implementation. In the case of 2D scheduling, the D-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the D-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $D(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

First scheduling variable (v1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v1 should be same as the size of the third dimension of A, B, C, and D.

Second scheduling variable (v2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v2 should be same as the size of the fourth dimension of A, B, C, and D.

Initial state, x_initial

Vector of initial states for the controller, i.e., initial values for the state vector, x. It should have length equal to the size of the first dimension of A.

Inputs and Outputs

The first input is the measurements.

The second and third block inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The output is the actuator demands.

2D Controller $[A(v),B(v),C(v),D(v)]$

Assumptions and Limitations If the scheduling parameter inputs to the block goes out of range, then they are clipped; i.e., the state-space matrices are not interpolated out of range.

Examples See the autopilot in the `aeroblk_HL20.mdl` demo for an example of this block.

See Also

- 1D Controller $[A(v),B(v),C(v),D(v)]$
- 2D Controller Blend
- 2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
- 2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
- 3D Controller $[A(v),B(v),C(v),D(v)]$

Purpose Implement a 2-D vector of state-space controllers by linear interpolation of their outputs

Library GNC

Description



The 2D Controller Blend block implements an array of state-space controller designs. The controllers are run in parallel, and their outputs interpolated according to the current flight condition or operating point. The advantage of this implementation approach is that the state-space matrices A, B, C, and D for the individual controller designs do not need to vary smoothly from one design point to the next.

For the 2D Controller Blend block, at any given instant in time, nine controller designs are updated.

As the value of the scheduling parameter varies and the index of the controllers that need to be run changes, the states of the oncoming controller are initialized by using the self-conditioned form as defined for the Self-Conditioned [A,B,C,D] block.

2D Controller Blend

Dialog Box

Block Parameters: 2D Controller Blend

Blend-2D (mask)
Blend between outputs of a 2-D vector of state-space controllers. All controllers must have the same state dimension.

Parameters

A-matrix(v1,v2):
A

B-matrix(v1,v2):
B

C-matrix(v1,v2):
C

D-matrix(v1,v2):
D

First scheduling variable (v1) breakpoints:
v1_vec

Second scheduling variable (v2) breakpoints:
v2_vec

Initial state, x_initial:
0

Poles of $A(v)H(v)C(v) = [w_1 \dots w_n]$:
[-5 -2]

OK Cancel Help Apply

A-matrix(v1,v2)

A-matrix of the state-space implementation. In the case of 2D blending, the A-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $A(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

B-matrix(v1,v2)

B-matrix of the state-space implementation.

C-matrix(v1,v2)

C-matrix of the state-space implementation.

D-matrix(v1,v2)

D-matrix of the state-space implementation.

First scheduling variable (v1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v1 should be same as the size of the third dimension of A, B, C, and D.

Second scheduling variable (v2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v2 should be same as the size of the fourth dimension of A, B, C, and D.

Initial state, x_{initial}

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Poles of $A(v)-H(v)*C(v)$

For oncoming controllers, an observer-like structure is used to ensure that the controller output tracks the current block output, u . The poles of the observer are defined in this dialog box as a vector, the number of poles being equal to the dimension of the A-matrix. Poles that are too fast result in sensor noise propagation, and poles that are too slow result in the failure of the controller output to track u .

Inputs and Outputs

The first input is the measurements.

The second and third inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The output is the actuator demands.

Assumptions and Limitations

This block requires the Control Systems Toolbox.

Examples

See the autopilot in the `aeroblk_HL20.mdl` demo for an example of this block in use.

References

Hyde, R.A., "H-infinity Aerospace Control Design - A VSTOL Flight Application," Springer Verlag, *Advances in Industrial Control Series*, 1995. ISBN 3-540-19960-8. See Chapter 5.

2D Controller Blend

See Also

1D Controller Blend $u=(1-L).K1.y+L.K2.y$

2D Controller $[A(v),B(v),C(v),D(v)]$

2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

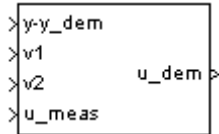
2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

2D Observer Form $[A(v), B(v), C(v), F(v), H(v)]$

Purpose Implement a gain-scheduled state-space controller in an observer form depending on two scheduling parameters

Library GNC

Description The 2D Observer Form $[A(v), B(v), C(v), F(v), H(v)]$ block implements a gain-scheduled state-space controller defined in the following observer form:



$$\begin{aligned} \dot{x} &= (A(v) + H(v)C(v))x + B(v)u_{meas} + H(v)(y - y_{dem}) \\ u_{dem} &= F(v)x \end{aligned}$$

The main application of these blocks is to implement a controller designed using H-infinity loop-shaping, one of the design methods supported by the μ -Analysis and Synthesis Toolbox.

2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

Dialog Box

Block Parameters: 2D Observer Form [A(v),B(v),C(v),F(v),H(v)]

StateSpaceABCFH-2D (mask)
Implement a state-space controller [A,B,C,F,H] in observer form where A, B, C, F, and H depend on two scheduling parameters.

Parameters

A-matrix(v1,v2):
A

B-matrix(v1,v2):
B

C-matrix(v1,v2):
C

F-matrix(v1,v2):
F

H-matrix(v1,v2):
H

First scheduling variable (v1) breakpoints:
v1_vec

Second scheduling variable (v2) breakpoints:
v2_vec

Initial state, x_initial:
0

OK Cancel Help Apply

A-matrix(v1,v2)

A-matrix of the state-space implementation. In the case of 2D scheduling, the A-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the A-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $A(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

B-matrix(v1,v2)

B-matrix of the state-space implementation. In the case of 2D scheduling, the B-matrix should have four dimensions, the last two corresponding to scheduling variables v1 and v2. Hence, for example, if the B-matrix corresponding to the first entry of v1 and first entry of v2 is the identity matrix, then $B(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

2D Observer Form $[A(\mathbf{v}),B(\mathbf{v}),C(\mathbf{v}),F(\mathbf{v}),H(\mathbf{v})]$

C-matrix($\mathbf{v}1,\mathbf{v}2$)

C-matrix of the state-space implementation. In the case of 2D scheduling, the C-matrix should have four dimensions, the last two corresponding to scheduling variables $\mathbf{v}1$ and $\mathbf{v}2$. Hence, for example, if the C-matrix corresponding to the first entry of $\mathbf{v}1$ and first entry of $\mathbf{v}2$ is the identity matrix, then $C(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

F-matrix($\mathbf{v}1,\mathbf{v}2$)

State-feedback matrix. In the case of 2D scheduling, the F-matrix should have four dimensions, the last two corresponding to scheduling variables $\mathbf{v}1$ and $\mathbf{v}2$. Hence, for example, if the F-matrix corresponding to the first entry of $\mathbf{v}1$ and first entry of $\mathbf{v}2$ is the identity matrix, then $F(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

H-matrix($\mathbf{v}1,\mathbf{v}2$)

Observer (output injection) matrix. In the case of 2D scheduling, the H-matrix should have four dimensions, the last two corresponding to scheduling variables $\mathbf{v}1$ and $\mathbf{v}2$. Hence, for example, if the H-matrix corresponding to the first entry of $\mathbf{v}1$ and first entry of $\mathbf{v}2$ is the identity matrix, then $H(:, :, 1, 1) = [1 \ 0; 0 \ 1];$.

First scheduling variable ($\mathbf{v}1$) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of $\mathbf{v}1$ should be same as the size of the third dimension of A, B, C, F, and H.

Second scheduling variable ($\mathbf{v}2$) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of $\mathbf{v}2$ should be same as the size of the fourth dimension of A, B, C, F, and H.

Initial state, $\mathbf{x_initial}$

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Inputs and Outputs

The first input is the set-point error.

The second and third inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The fourth input is measured actuator position.

The output is the actuator demands.

2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

Assumptions and Limitations If the scheduling parameter inputs to the block goes out of range, then they are clipped; i.e., the state-space matrices are not interpolated out of range.

Examples See the observer implementation within the aeroblk_HL20.mdl demo.

References Hyde, R.A., "H-infinity Aerospace Control Design - A VSTOL Flight Application," Springer Verlag, *Advances in Industrial Control Series*, 1995. ISBN 3-540-19960-8. See Chapter 6.

See Also

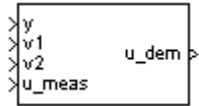
- 1D Controller $[A(v),B(v),C(v),D(v)]$
- 2D Controller $[A(v),B(v),C(v),D(v)]$
- 2D Controller Blend
- 2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$
- 3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

2D Self-Conditioned [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller in a self-conditioned form

Library GNC

Description



The 2D Self-Conditioned [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations

$$\dot{x} = A(v)x + B(v)y$$

$$u = C(v)x + D(v)y$$

in the self-conditioned form

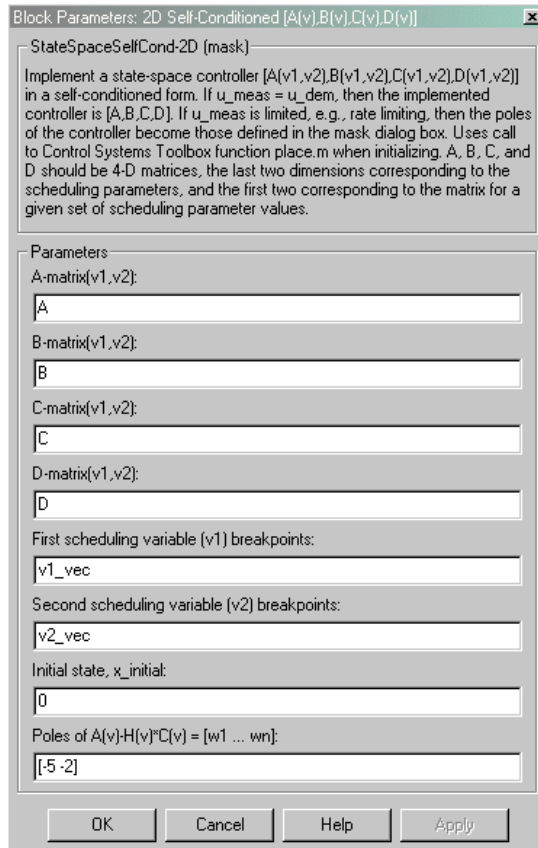
$$\dot{z} = (A(v) - H(v)C(v))z + (B(v) - H(v)D(v))e + H(v)u_{meas}$$

$$u_{dem} = C(v)z + D(v)e$$

For the rationale behind this self-conditioned implementation, refer to the Self-Conditioned [A,B,C,D] block reference. This block implements a gain-scheduled version of the Self-Conditioned [A,B,C,D] block, v being the vector of parameters over which A, B, C, and D are defined. This type of controller scheduling assumes that the matrices A, B, C, and D vary smoothly as a function of v , which is often the case in aerospace applications.

2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

Dialog Box



A-matrix(v1,v2)

A-matrix of the state-space implementation. In the case of 2D scheduling, the A-matrix should have four dimensions, the last two corresponding to scheduling variables $v1$ and $v2$. Hence, for example, if the A-matrix corresponding to the first entry of $v1$ and first entry of $v2$ is the identity matrix, then $A(:, :, 1, 1) = [1 \ 0; 0 \ 1];$

B-matrix(v1,v2)

B-matrix of the state-space implementation. In the case of 2D scheduling, the B-matrix should have four dimensions, the last two corresponding to scheduling variables $v1$ and $v2$. Hence, for example, if the B-matrix

2D Self-Conditioned $[A(v), B(v), C(v), D(v)]$

corresponding to the first entry of v_1 and first entry of v_2 is the identity matrix, then $B(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

C-matrix(v_1, v_2)

C-matrix of the state-space implementation. In the case of 2D scheduling, the C-matrix should have four dimensions, the last two corresponding to scheduling variables v_1 and v_2 . Hence, for example, if the C-matrix corresponding to the first entry of v_1 and first entry of v_2 is the identity matrix, then $C(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

D-matrix(v_1, v_2)

D-matrix of the state-space implementation. In the case of 2D scheduling, the D-matrix should have four dimensions, the last two corresponding to scheduling variables v_1 and v_2 . Hence, for example, if the D-matrix corresponding to the first entry of v_1 and first entry of v_2 is the identity matrix, then $D(:, :, 1, 1) = [1 \ 0; 0 \ 1]$;

First scheduling variable (v_1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v_1 should be same as the size of the third dimension of A, B, C, and D.

Second scheduling variable (v_2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v_2 should be same as the size of the fourth dimension of A, B, C, and D.

Initial state, x_{initial}

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A.

Poles of $A(v)-H(v)*C(v)$

Vector of the desired poles of $A-HC$. Note that the poles are assigned to the same locations for all values of the scheduling parameter, v . Hence the number of pole locations defined should be equal to the length of the first dimension of the A-matrix.

Inputs and Outputs

The first input is the measurements.

The second and third inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The fourth input is the measured actuator position.

2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

The output is the actuator demands.

Assumptions and Limitations If the scheduling parameter inputs to the block goes out of range, then they are clipped; i.e., the state-space matrices are not interpolated out of range.

This block requires the Control Systems Toolbox.

References The algorithm used to determine the matrix H is defined in Kautsky, Nichols, and Van Dooren, "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, No. 5, pages 1129-1155, 1985.

See Also

1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

2D Controller $[A(v),B(v),C(v),D(v)]$

2D Controller Blend

2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

3D Controller [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller depending on three scheduling parameters

Library GNC

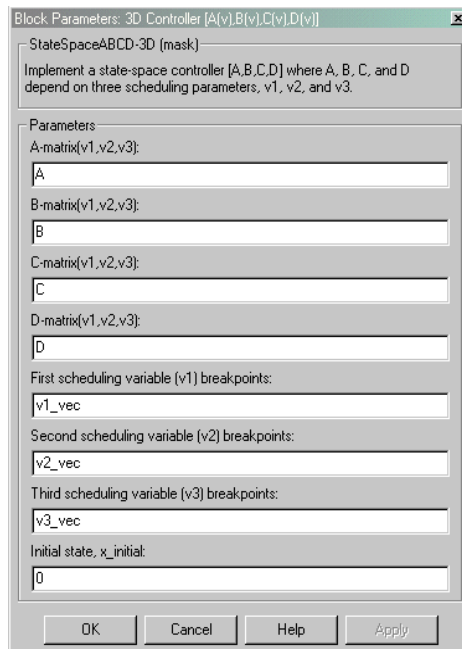
Description The 3D Controller [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations



$$\begin{aligned} \dot{x} &= A(v)x + B(v)u \\ u &= C(v)x + D(v)y \end{aligned}$$

where v is a vector of parameters over which A, B, C, and D are defined. This type of controller scheduling assumes that the matrices A, B, C, and D vary smoothly as a function of v , which is often the case in aerospace applications.

Dialog Box



3D Controller [A(v),B(v),C(v),D(v)]

A-matrix(v1,v2,v3)

A-matrix of the state-space implementation. In the case of 3D scheduling, the A-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the A-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $A(:, :, 1, 1, 1) = [1 \ 0 \ 0; 0 \ 1 \ 0; 0 \ 0 \ 1]$;

B-matrix(v1,v2,v3)

B-matrix of the state-space implementation. In the case of 3D scheduling, the B-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the B-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $B(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1]$;

C-matrix(v1,v2,v3)

C-matrix of the state-space implementation. In the case of 3D scheduling, the C-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the C-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $C(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1]$;

D-matrix(v1,v2,v3)

D-matrix of the state-space implementation. In the case of 3D scheduling, the D-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the D-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $D(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1]$;

First scheduling variable (v1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v1 should be same as the size of the third dimension of A, B, C, and D.

Second scheduling variable (v2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v2 should be same as the size of the fourth dimension of A, B, C, and D.

Third scheduling variable (v3) breakpoints

Vector of the breakpoints for the third scheduling variable. The length of v3 should be same as the size of the fifth dimension of A, B, C, and D.

3D Controller $[A(v),B(v),C(v),D(v)]$

Initial state, $x_{initial}$

Vector of initial states for the controller, i.e., initial values for the state vector, x . It should have length equal to the size of the first dimension of A .

Inputs and Outputs

The first input is the measurements.

The second, third and fourth inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The output is the actuator demands.

Assumptions and Limitations

If the scheduling parameter input to the block goes out of range, then it is clipped; i.e., the state-space matrices are not interpolated out of range.

Examples

See the autopilot in the `aeroblk_HL20.mdl` demo for an example of this block.

See Also

1D Controller $[A(v),B(v),C(v),D(v)]$

2D Controller $[A(v),B(v),C(v),D(v)]$

3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

Purpose

Implement a gain-scheduled state-space controller in an observer form depending on three scheduling parameters

Library

GNC

Description

```
>y-y_dem
>v1
>v2      u_dem >
>v3
>u_meas
```

The 3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$ block implements a gain-scheduled state-space controller defined in the following observer form:

$$\dot{x} = (A(v) + H(v)C(v))x + B(v)u_{meas} + H(v)(y - y_{dem})$$

$$u_{dem} = F(v)x$$

The main application of this block is to implement a controller designed using H-infinity loop-shaping, one of the design methods supported by the μ -Analysis and Synthesis Toolbox.

3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

Dialog Box

Block Parameters: 3D Observer Form [A(v),B(v),C(v),F(v),H(v)]

StateSpaceABCFH-3D (mask)

Implement a state-space controller [A,B,C,F,H] in observer form where A, B, C, F, and H depend on three scheduling parameters.

Parameters

A-matrix(v1,v2,v3):
A

B-matrix(v1,v2,v3):
B

C-matrix(v1,v2,v3):
C

F-matrix(v1,v2,v3):
F

H-matrix(v1,v2,v3):
H

First scheduling variable (v1) breakpoints:
v1_vec

Second scheduling variable (v2) breakpoints:
v2_vec

Third scheduling variable (v3) breakpoints:
v3_vec

Initial state, x_initial:
0

OK Cancel Help Apply

A-matrix(v1,v2,v3)

A-matrix of the state-space implementation. In the case of 3D scheduling, the A-matrix should have five dimensions, the last three corresponding to scheduling variables v_1 , v_2 , and v_3 . Hence, for example, if the A-matrix corresponding to the first entry of v_1 , the first entry of v_2 , and the first entry of v_3 is the identity matrix, then $A(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$

B-matrix(v1,v2,v3)

B-matrix of the state-space implementation. In the case of 3D scheduling, the B-matrix should have five dimensions, the last three corresponding to scheduling variables v_1 , v_2 , and v_3 . Hence, for example, if the B-matrix corresponding to the first entry of v_1 , the first entry of v_2 , and the first entry of v_3 is the identity matrix, then $B(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1];$

3D Observer Form $[A(v), B(v), C(v), F(v), H(v)]$

C-matrix(v1,v2,v3)

C-matrix of the state-space implementation. In the case of 3D scheduling, the C-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the C-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $C(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

F-matrix(v1,v2,v3)

State-feedback matrix. In the case of 3D scheduling, the F-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the F-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $F(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

H-matrix(v1,v2,v3)

observer (output injection) matrix. In the case of 3D scheduling, the H-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the H-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $H(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

First scheduling variable (v1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v1 should be same as the size of the third dimension of A, B, C, F, and H.

Second scheduling variable (v2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v2 should be same as the size of the fourth dimension of A, B, C, F, and H.

Third scheduling variable (v3) breakpoints

Vector of the breakpoints for the third scheduling variable. The length of v3 should be same as the size of the fifth dimension of A, B, C, F, and H.

Initial state, x_initial

Vector of initial states for the controller, i.e., initial values for the state vector, x. It should have length equal to the size of the first dimension of A.

Inputs and Outputs

The first input is the set-point error.

3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

The second, third, and fourth inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The fifth input is measured actuator position.

The output is the actuator demands.

Assumptions and Limitations If the scheduling parameter inputs to the block goes out of range, then they are clipped; i.e., the state-space matrices are not interpolated out of range.

Examples See the observer implementation within the `aerob1k_HL20.mdl` demo.

References Hyde, R.A., “H-infinity Aerospace Control Design - A VSTOL Flight Application,” Springer Verlag, *Advances in Industrial Control Series*, 1995. ISBN 3-540-19960-8. See Chapter 6.

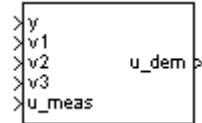
See Also 1D Controller $[A(v),B(v),C(v),D(v)]$
2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$
3D Controller $[A(v),B(v),C(v),D(v)]$
3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

3D Self-Conditioned [A(v),B(v),C(v),D(v)]

Purpose Implement a gain-scheduled state-space controller in a self-conditioned form

Library GNC

Description



The 3D Self-Conditioned [A(v),B(v),C(v),D(v)] block implements a gain-scheduled state-space controller as defined by the equations

$$\dot{x} = A(v)x + B(v)y$$

$$u = C(v)x + D(v)y$$

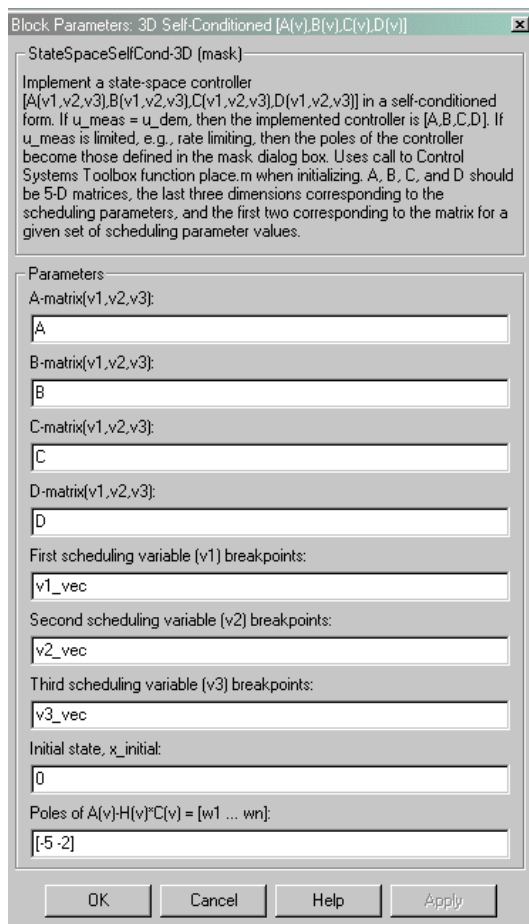
in the self-conditioned form

$$\dot{z} = (A(v) - H(v)C(v))z + (B(v) - H(v)D(v))e + H(v)u_{meas}$$

$$u_{dem} = C(v)z + D(v)e$$

For the rationale behind this self-conditioned implementation, refer to the Self-Conditioned [A,B,C,D] block reference. These blocks implement a gain-scheduled version of the Self-Conditioned [A,B,C,D] block, v being the vector of parameters over which A, B, C, and D are defined. This type of controller scheduling assumes that the matrices A, B, C, and D vary smoothly as a function of v , which is often the case in aerospace applications.

Dialog Box



A-matrix(v1,v2,v3)

A-matrix of the state-space implementation. In the case of 3D scheduling, the A-matrix should have five dimensions, the last three corresponding to scheduling variables v_1 , v_2 , and v_3 . Hence, for example, if the A-matrix corresponding to the first entry of v_1 , the first entry of v_2 , and the first entry of v_3 is the identity matrix, then $A(:, :, 1, 1, 1) = [1 \ 0; 0 \ 1]$;

3D Self-Conditioned $[A(v), B(v), C(v), D(v)]$

B-matrix(v1,v2,v3)

B-matrix of the state-space implementation. In the case of 3D scheduling, the B-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the B-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $B(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

C-matrix(v1,v2,v3)

C-matrix of the state-space implementation. In the case of 3D scheduling, the C-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the C-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $C(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

D-matrix(v1,v2,v3)

D-matrix of the state-space implementation. In the case of 3D scheduling, the D-matrix should have five dimensions, the last three corresponding to scheduling variables v1, v2, and v3. Hence, for example, if the D-matrix corresponding to the first entry of v1, the first entry of v2, and the first entry of v3 is the identity matrix, then $D(:, :, 1, 1, 1) = [1 \ 0 \ 0 \ 1]$;

First scheduling variable (v1) breakpoints

Vector of the breakpoints for the first scheduling variable. The length of v1 should be same as the size of the third dimension of A, B, C, and D.

Second scheduling variable (v2) breakpoints

Vector of the breakpoints for the second scheduling variable. The length of v2 should be same as the size of the fourth dimension of A, B, C, and D.

Third scheduling variable (v3) breakpoints

Vector of the breakpoints for the third scheduling variable. The length of v3 should be same as the size of the fifth dimension of A, B, C, and D.

Initial state, x_initial

Vector of initial states for the controller, i.e., initial values for the state vector, x. It should have length equal to the size of the first dimension of A.

Poles of $A(v)-H(v)*C(v)$

Vector of the desired poles of A-HC. Note that the poles are assigned to the same locations for all values of the scheduling parameter v. Hence the

3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

number of pole locations defined should be equal to the length of the first dimension of the A-matrix.

Inputs and Outputs

The first input is the measurements.

The second, third, and fourth inputs are the scheduling variables ordered conforming to the dimensions of the state-space matrices.

The fifth input is the measured actuator position.

The output is the actuator demands.

Assumptions and Limitations

If the scheduling parameter inputs to the block goes out of range, then they are clipped; i.e., the state-space matrices are not interpolated out of range.

This block requires the Control Systems Toolbox.

References

The algorithm used to determine the matrix H is defined in Kautsky, Nichols, and Van Dooren, "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, No. 5, pages 1129-1155, 1985.

See Also

1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$

3D Controller $[A(v),B(v),C(v),D(v)]$

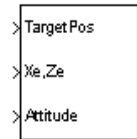
3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$

3DoF Animation

Purpose Create a 3-D Handle Graphics animation of a three-degrees-of-freedom object

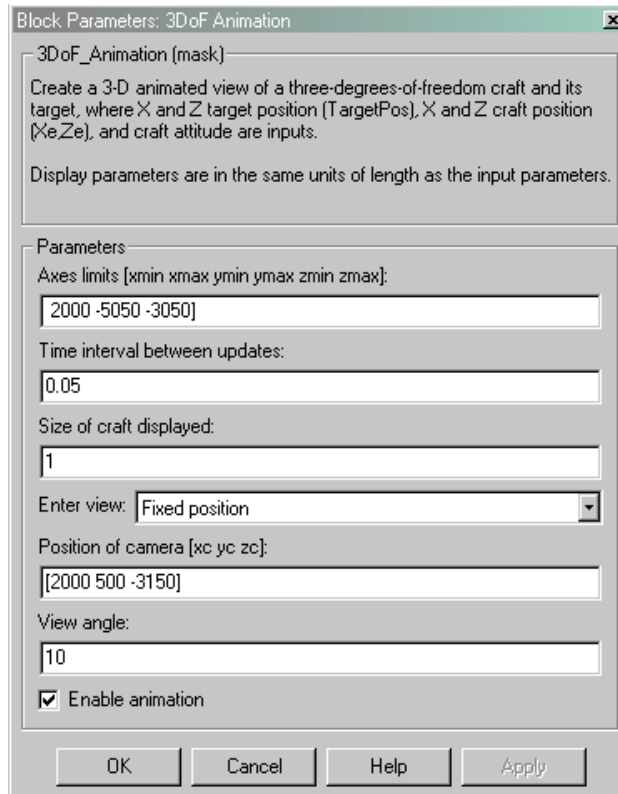
Library Animation

Description The 3DoF Animation block displays a 3-D animated view of a three-degrees-of-freedom (3DoF) craft, its trajectory, and its target using Handle Graphics.



The 3DoF Animation block uses the input values and the dialog parameters to create and display the animation.

Dialog Box



Axes limits [xmin xmax ymin ymax zmin zmax]

Specifies the three-dimensional space to be viewed.

Time interval between updates

Specifies the time interval at which the animation will be redrawn.

Size of craft displayed

Scale factor to adjust the size of the craft and target.

Enter view

Selects preset Handle Graphics parameters **CameraTarget** and **CameraUpVector** for the figure axes. The dialog parameters **Position of camera** and **View angle** are used to customize the position and field of view for the selected view. Possible views are

- Fixed position
- Cockpit
- Fly alongside

Position of camera [xc yc zc]

Specifies the Handle Graphics parameter **CameraPosition** for the figure axes. Used in all cases except for the Cockpit view.

View angle

Specifies the Handle Graphics parameter **CameraViewAngle** for the figure axes in degrees.

Enable animation

When selected, the animation is displayed during the simulation. If not selected, the animation is not displayed.

Inputs

The first input is a vector containing the altitude and the downrange position of the target in Earth coordinates.

The second input is a vector containing the altitude and the downrange position of the craft in Earth coordinates.

The third input is the attitude of the craft.

See Also

6DoF Animation

3x3 Cross Product

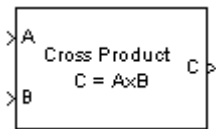
Purpose

Calculate the cross product of two 3-by-1 vectors

Library

Transformations/Axes

Description



The 3x3 Cross Product block computes cross (or vector) product of two vectors, A and B, by generating a third vector, C, in a direction normal to the plane containing A and B, and with magnitude equal to the product of the lengths of A and B multiplied by the sine of the angle between them. The direction of C is that in which a right-handed screw would move in turning from A to B.

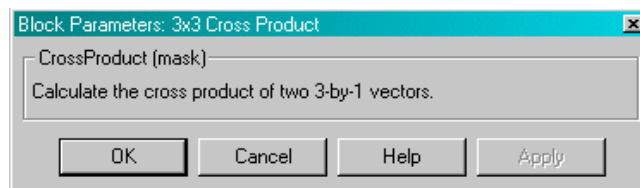
$$A = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$$

$$B = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$$

$$C = A \times B = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

$$= (a_2b_3 - a_3b_2)\mathbf{i} + (a_3b_1 - a_1b_3)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k}$$

Dialog Box



Inputs and Outputs

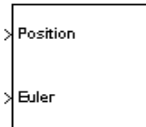
The inputs are two 3-by-1 vectors.

The output is a 3-by-1 vector.

Purpose Create a 3-D Handle Graphics animation of a six-degrees-of-freedom object

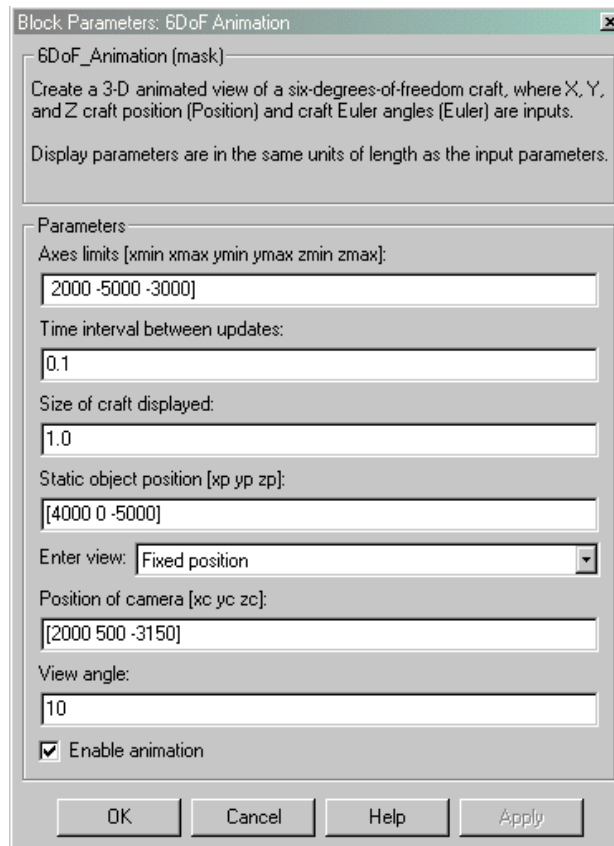
Library Animation

Description The 6DoF Animation block displays a 3-D animated view of a six-degrees-of-freedom (6DoF) craft, its trajectory, and its target using Handle Graphics.



The 6DoF Animation block uses the input values and the dialog parameters to create and display the animation.

Dialog Box



6DoF Animation

Axes limits [xmin xmax ymin ymax zmin zmax]

Specifies the three-dimensional space to be viewed.

Time interval between updates

Specifies the time interval at which the animation will be redrawn.

Size of craft displayed

Scale factor to adjust the size of the craft and target.

Static object position

Specifies the altitude, the cross-range position, and the downrange position of the target.

Enter view

Selects preset Handle Graphics parameters **CameraTarget** and **CameraUpVector** for the figure axes. The dialog parameters **Position of camera** and **View angle** are used to customize the position and field of view for the selected view. Possible views are

- Fixed position
- Cockpit
- Fly alongside

Position of camera [xc yc zc]

Specifies the Handle Graphics parameter **CameraPosition** for the figure axes. Used in all cases except for the Cockpit view.

View angle

Specifies the Handle Graphics parameter **CameraViewAngle** for the figure axes in degrees.

Enable animation

When selected, the animation is displayed during the simulation. If not selected, the animation is not displayed.

Inputs

The first input is a vector containing the altitude, the cross-range position, and the downrange position of the craft in Earth coordinates.

The second input is a vector containing the Euler angles of the craft.

See Also

3DoF Animation

Purpose

Implement an Euler angle representation of six-degrees-of-freedom equations of motion

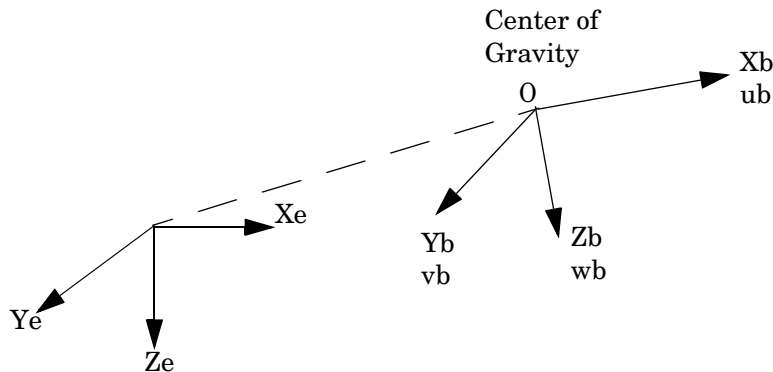
Library

Equations of Motion/6DoF

Description

	\underline{V}_e (m/s) >
> Forces X Y Z (N)	X_e (m) >
	Euler (rad) >
Euler Angles	DCM >
	\underline{V}_b (m/s) >
> Moments L M N (N-m)	p, q, r (rad/s) >
	$p\dot{p}, q\dot{q}, r\dot{r}$ (rad/s ²) >

The 6DoF (Euler Angles) block considers the rotation of a body-fixed coordinate frame (X_b, Y_b, Z_b) about an Earth-fixed reference frame (X_e, Y_e, Z_e) . The origin of the body-fixed coordinate frame is the center of gravity of the body, and the body is assumed to be rigid, an assumption that eliminates the need to consider the forces acting between individual elements of mass. The Earth-fixed reference frame is considered inertial, a simplification that allows the forces due to the Earth's motion relative to a star-fixed reference system to be neglected.



Earth-fixed reference frame

The translational motion of the body-fixed coordinate frame is given below, where the applied forces $[F_x F_y F_z]^T$ are in the body-fixed frame, and the mass of the body m is assumed constant.

$$\underline{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\underline{\dot{V}}_b + \underline{\omega} \times \underline{V}_b)$$

6DoF (Euler Angles)

$$\underline{V}_b = \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}, \underline{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

The rotational dynamics of the body-fixed frame are given below, where the applied moments are $[L \ M \ N]^T$, and the inertia tensor I is with respect to the origin O .

$$\underline{M}_B = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I \underline{\dot{\omega}} + \underline{\omega} \times (I \underline{\omega})$$

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

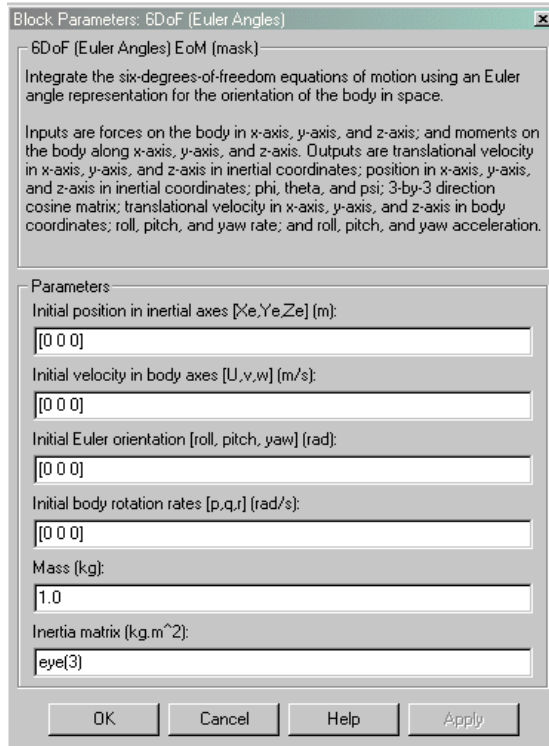
The relationship between the body-fixed angular velocity vector, $[p \ q \ r]^T$, and the rate of change of the Euler angles, $[\dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T$, can be determined by resolving the Euler rates into the body-fixed coordinate frame.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \equiv J^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

Inverting J then gives the required relationship to determine the Euler rate vector.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = J \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & (\sin \phi \tan \theta) & (\cos \phi \tan \theta) \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Dialog Box



Initial position in inertial axes

The three-element vector for the initial location of the body in the Earth-fixed reference frame, in meters.

Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame, in meters per second.

Initial Euler rotation

The three-element vector for the initial Euler rotation angles [roll, pitch, yaw], in radians.

Initial body rotation rates

The three-element vector for the initial body-fixed angular rates, in radians per second.

6DoF (Euler Angles)

Mass

The mass of the rigid body, in kilograms.

Inertia matrix

The 3-by-3 inertia tensor matrix I , in kilograms meters squared.

Inputs and Outputs

The first input to the block is a vector containing the three applied forces, in Newtons, and the second input is a vector containing the three applied moments, in Newton meters.

The first output is a three-element vector containing the velocity in the Earth-fixed reference frame, in meters per second.

The second output is a three-element vector containing the position in the Earth-fixed reference frame, in meters.

The third output is a three-element vector containing the Euler rotation angles [roll, pitch, yaw], in radians.

The fourth output is a 3-by-3 matrix for the coordinate transformation from Earth-fixed axes to body-fixed axes.

The fifth output is a three-element vector containing the velocity in the body-fixed frame, in meters per second.

The sixth output is a three-element vector containing the angular rates in body-fixed axes, in radians per second.

The seventh output is a three-element vector containing the angular accelerations in body-fixed axes, in radians per second.

Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body, and that the mass and inertia are constant.

See Also

6DoF (Quaternion)

Purpose

Implement a quaternion representation of six-degrees-of-freedom equations of motion

Library

Equations of Motion/6DoF

Description

	$\backslash e$ (m/s) >
	$\backslash e$ (m) >
> Forces X Y Z (N)	Euler (rad) >
	Quaternions DCM >
	$\backslash b$ (m/s) >
> Moments L M N (N-m)	p, q, r (rad/s) >
	$pdot, qdot, rdot$ (rad/s ²) >

For a description of the coordinate system employed and the translational dynamics, see the block description for the 6DoF (Euler Angles) block.

The integration of the rate of change of the quaternion vector is given below. The gain K drives the norm of the quaternion state vector to 1.0 should ε become nonzero. You must choose the value of this gain with care, because a large value improves the decay rate of the error in the norm, but also slows the simulation because fast dynamics are introduced. An error in the magnitude in one element of the quaternion vector is spread equally among all the elements, potentially increasing the error in the state vector.

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} q_3 & -q_2 & q_1 \\ q_2 & q_3 & -q_0 \\ -q_1 & q_0 & q_3 \\ -q_0 & -q_1 & -q_2 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + K\varepsilon \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

$$\varepsilon = 1 - (q_0^2 + q_1^2 + q_2^2 + q_3^2)$$

6DoF (Quaternion)

Dialog Box

Block Parameters: 6DoF (Quaternion) ✕

6DoF (Quaternion) EoM (mask)

Integrate the six-degrees-of-freedom equations of motion using a quaternion representation for the orientation of the body in space.

Inputs are forces on the body in x-axis, y-axis, and z-axis; and moments on the body along x-axis, y-axis, and z-axis. Outputs are translational velocity in x-axis, y-axis, and z-axis in inertial coordinates; position in x-axis, y-axis, and z-axis in inertial coordinates; phi, theta, and psi; 3-by-3 direction cosine matrix; translational velocity in x-axis, y-axis, and z-axis in body coordinates; roll, pitch, and yaw rate; and roll, pitch, and yaw acceleration.

Parameters

Initial position in inertial axes [Xe,Ye,Ze] (m):

Initial velocity in body axes [U,v,w] (m/s):

Initial Euler orientation [roll, pitch, yaw] (rad):

Initial body rotation rates [p,q,r] (rad/s):

Mass (kg):

Inertia matrix (kg.m²):

Gain for quaternion normalization:

Initial position in inertial axes

The three-element vector for the initial location of the body in the Earth-fixed reference frame, in meters.

Initial velocity in body axes

The three-element vector for the initial velocity in the body-fixed coordinate frame, in meters per second.

Initial Euler rotation

The three-element vector for the initial Euler rotation angles [roll, pitch, yaw], in radians.

Initial body rotation rates

The three-element vector for the initial body-fixed angular rates, in radians per second.

Mass

The mass of the rigid body, in kilograms.

Inertia matrix

The 3-by-3 inertia tensor matrix I , in kilograms meters squared.

Normalization gain

The gain to maintain the norm of the quaternion vector equal to 1.0.

Inputs and Outputs

The first input to the block is a vector containing the three applied forces, in Newtons, and the second input is a vector containing the three applied moments, in Newton meters.

The first output is a three-element vector containing the velocity in the Earth-fixed reference frame, in meters per second.

The second output is a three-element vector containing the position in the Earth-fixed reference frame, in meters.

The third output is a three-element vector containing the Euler rotation angles [roll, pitch, yaw], in radians.

The fourth output is a 3-by-3 matrix for the coordinate transformation from Earth-fixed axes to body-fixed axes.

The fifth output is a three-element vector containing the velocity in the body-fixed frame, in meters per second.

The sixth output is a three-element vector containing the angular rates in body-fixed axes, in radians per second.

The seventh output is a three-element vector containing the angular accelerations in body-fixed axes, in radians per second.

Assumptions and Limitations

The block assumes that the applied forces are acting at the center of gravity of the body, and that the mass and inertia are constant.

6DoF (Quaternion)

Examples

See the Simulink model `aeroblk_six_dof.mdl` for an example of the use of the 6DoF (Quaternion) block.

See Also

6DoF (Euler Angles)

Purpose Convert from acceleration units to desired acceleration units

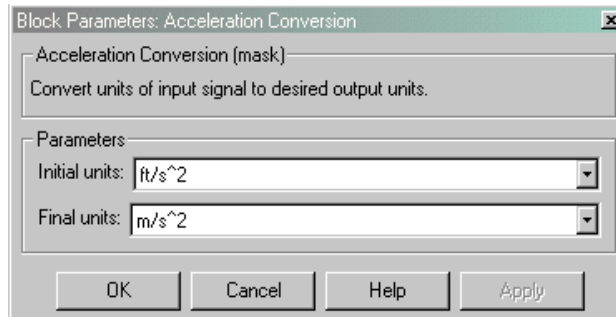
Library Transformations/Units

Description The Acceleration Conversion block computes the conversion factor from specified input acceleration units to specified output acceleration units and applies the conversion factor to the input signal.



The Acceleration Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

m/s^2	Meters per second squared
ft/s^2	Feet per second squared
km/s^2	Kilometers per second squared
in/s^2	Inches per second squared
$km/h-s$	Kilometers per hour per second
$mph-s$	Miles per hour per second

Acceleration Conversion

Inputs and Outputs

The input is acceleration in initial acceleration units.

The output is acceleration in final acceleration units.

See Also

Angle Conversion

Angular Acceleration Conversion

Angular Velocity Conversion

Density Conversion

Force Conversion

Length Conversion

Mass Conversion

Pressure Conversion

Temperature Conversion

Velocity Conversion

Purpose Convert from angle units to desired angle units

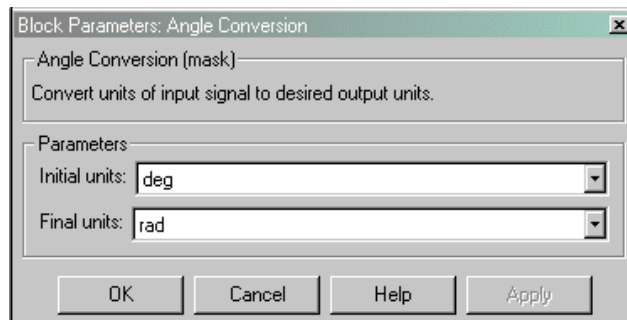
Library Transformations/Units

Description The Angle Conversion block computes the conversion factor from specified input angle units to specified output angle units and applies the conversion factor to the input signal.



The Angle Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

deg	Degrees
rad	Radians
rev	Revolutions

Inputs and Outputs

The input is angle in initial angle units.

The output is angle in final angle units.

Angle Conversion

See Also

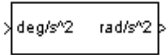
Acceleration Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Angular Acceleration Conversion

Purpose Convert from angular acceleration units to desired angular acceleration units

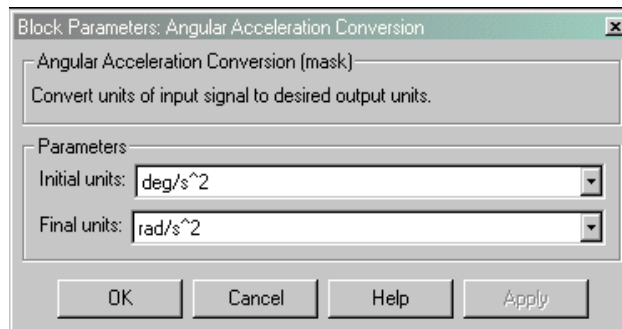
Library Transformations/Units

Description The Angular Acceleration Conversion block computes the conversion factor from specified input angular acceleration units to specified output angular acceleration units and applies the conversion factor to the input signal.



The Angular Acceleration Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

deg/s ²	Degrees per second squared
rad/s ²	Radians per second squared
rpm/s	Revolutions per minute per second

Inputs and Outputs

The input is angular acceleration in initial angular acceleration units.

The output is angular acceleration in final angular acceleration units.

Angular Acceleration Conversion

See Also

Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Density Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Angular Velocity Conversion

Purpose Convert from angular velocity units to desired angular velocity units

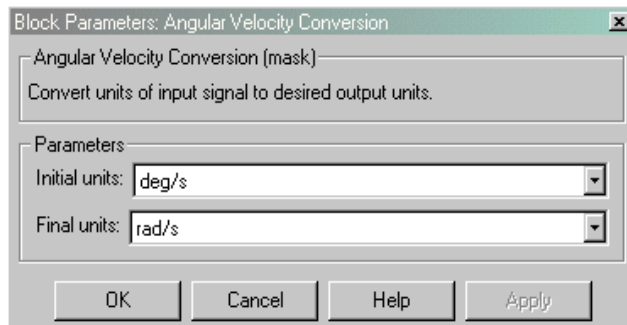
Library Transformations/Units

Description The Angular Velocity Conversion block computes the conversion factor from specified input angular velocity units to specified output angular velocity units and applies the conversion factor to the input signal.



The Angular Velocity Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

deg/s	Degrees per second
rad/s	Radians per second
rpm	Revolutions per minute

Inputs and Outputs

The input is angular velocity in initial angular velocity units.

The output is angular velocity in final angular velocity units.

Angular Velocity Conversion

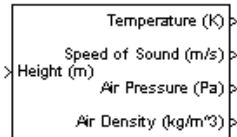
See Also

Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Density Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Purpose Implement the 1976 COESA lower atmosphere

Library Environment/Atmosphere

Description

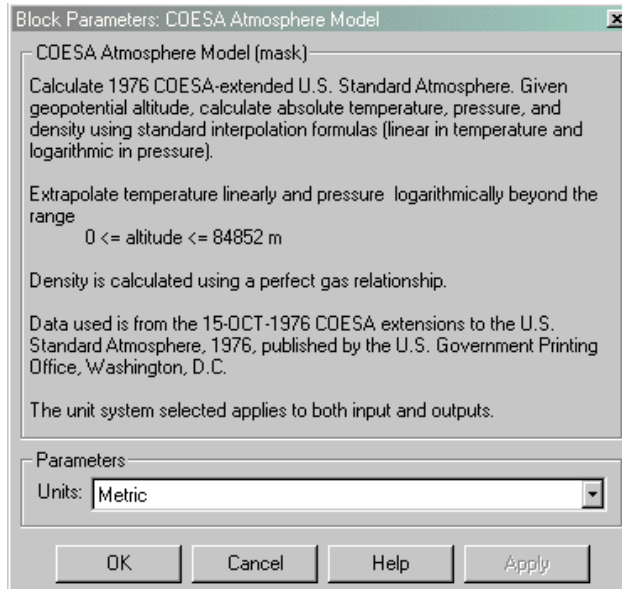


The COESA Atmosphere Model block implements the mathematical representation of the 1976 COESA (Committee on Extension to the Standard Atmosphere) United States (U.S.) standard lower atmospheric values for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.

Below 32000 m (approximately 104987 ft), the U.S. Standard Atmosphere is identical with the Standard Atmosphere of the International Civil Aviation Organization (ICAO).

The COESA Atmosphere Model block icon displays the input and output units selected from the **Units** pop-up menu.

Dialog Box



COESA Atmosphere Model

Units

Specifies the input and output units:

	Height	Temperature	Speed of Sound	Air Density
Metric	Meters	Degrees Kelvin	Meters per second	Kilograms per cubic meter
English	Feet	Degrees Rankine	Feet per second	Pound mass per cubic foot

Inputs and Outputs

The input is geopotential height.

The four outputs are temperature, speed of sound, air pressure, and air density.

Assumptions and Limitations

Below the geopotential altitude of 0 m (0 ft) and above the geopotential altitude of 84852 m (approximately 278386 ft), temperature values are extrapolated linearly and pressure values are extrapolated logarithmically. Density and speed of sound are calculated using a perfect gas relationship.

References

[1] U.S. Standard Atmosphere, 1976, U.S. Government Printing Office, Washington, D.C.

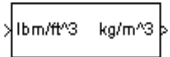
See Also

ISA Atmosphere Model

Purpose Convert from density units to desired density units

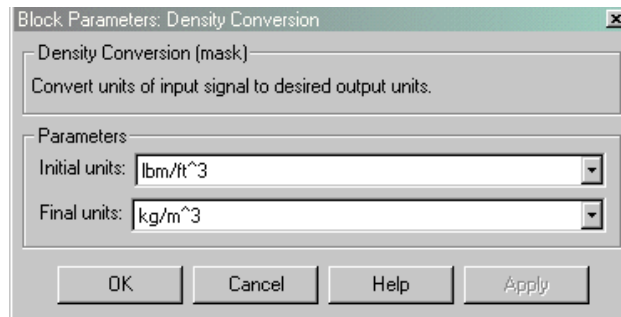
Library Transformations/Units

Description The Density Conversion block computes the conversion factor from specified input density units to specified output density units and applies the conversion factor to the input signal.



The Density Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

lbm/ft ³	Pound mass per cubic foot
kg/m ³	Kilograms per cubic meter
slug/ft ³	Slugs per cubic foot
lbm/in ³	Pound mass per cubic inch

Inputs and Outputs

The input is density in initial density units.

The output is density in final density units.

Density Conversion

See Also

Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Direction Cosine Matrix to Euler Angles

Purpose Convert direction cosine matrix to Euler angles

Library Transformations/Axes

Description



The Direction Cosine Matrix to Euler Angles block converts a 3-by-3 direction cosine matrix (DCM) into three Euler rotation angles. The DCM matrix performs the coordinate transformation of a vector in inertial axes (ox_0, oy_0, oz_0) into a vector in body axes (ox_3, oy_3, oz_3) . The order of the axis rotations required to bring (ox_3, oy_3, oz_3) into coincidence with (ox_0, oy_0, oz_0) is first a rotation about ox_3 through the roll angle (ϕ) to axes (ox_2, oy_2, oz_2) . Second a rotation about oy_2 through the pitch angle (θ) to axes (ox_1, oy_1, oz_1) , and finally a rotation about oz_1 through the yaw angle (ψ) to axes (ox_0, oy_0, oz_0) .

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = DCM \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the three axis transformation matrices defines the following DCM.

$$DCM = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi) & (\sin \phi \sin \theta \sin \psi - \cos \phi \cos \psi) & \sin \phi \cos \theta \\ (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi) & (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi) & \cos \phi \cos \theta \end{bmatrix}$$

To determine Euler angles from the DCM, the following equations are used:

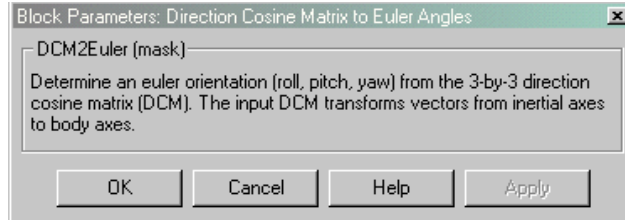
$$\phi = \text{atan}\left(\frac{DCM(2, 3)}{DCM(3, 3)}\right)$$

$$\theta = \text{asin}(-DCM(1, 3))$$

Direction Cosine Matrix to Euler Angles

$$\psi = \operatorname{atan}\left(\frac{DCM(1, 2)}{DCM(1, 1)}\right)$$

Dialog Box



Inputs and Outputs

The input is a 3-by-3 direction cosine matrix.

The output is a 3-by-1 vector of Euler angles.

Assumptions and Limitations

This implementation generates a pitch angle that lies between ± 90 degrees, and roll and yaw angles that lie between ± 180 degrees.

See Also

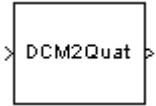
Direction Cosine Matrix to Quaternions
Euler Angles to Direction Cosine Matrix
Euler Angles to Quaternions
Quaternions to Direction Cosine Matrix
Quaternions to Euler Angles

Direction Cosine Matrix to Quaternions

Purpose Convert direction cosine matrix to quaternion vector

Library Transformations/Axes

Description



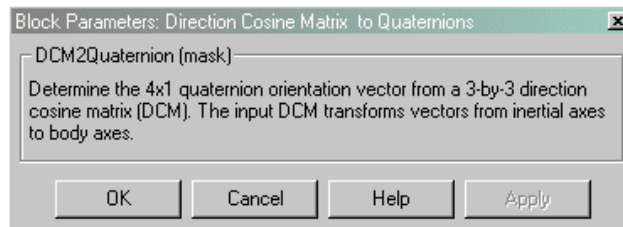
The Direction Cosine Matrix to Quaternions block transforms a 3-by-3 direction cosine matrix (DCM) into a four-element unit quaternion vector (q_0, q_1, q_2, q_3) . The DCM performs the coordinate transformation of a vector in inertial axes to a vector in body axes.

The DCM is defined as a function of a unit quaternion vector by the following:

$$DCM = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix}$$

Using this representation of the DCM, there are a number of calculations to arrive at the correct quaternion. The first of these is to calculate the trace of the DCM to determine which set of algorithms are used. If the trace is greater than zero, the quaternion can be automatically calculated. When the trace is less than or equal to zero, the major diagonal element of the DCM with the greatest value must be identified to determine the final algorithm used to calculate the quaternion. Once the major diagonal element is identified, the quaternion is calculated. For a detailed view of these algorithms, look under the mask of the Direction Cosine Matrix to Quaternions block.

Dialog Box



Inputs and Outputs

The input is a 3-by-3 direction cosine matrix.

The output is a 4-by-1 quaternion vector.

Direction Cosine Matrix to Quaternions

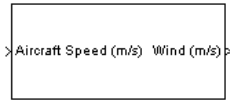
See Also

[Direction Cosine Matrix to Euler Angles](#)
[Euler Angles to Direction Cosine Matrix](#)
[Euler Angles to Quaternions](#)
[Quaternions to Direction Cosine Matrix](#)
[Quaternions to Euler Angles](#)

Purpose Generate discrete wind gust

Library Environment/Wind

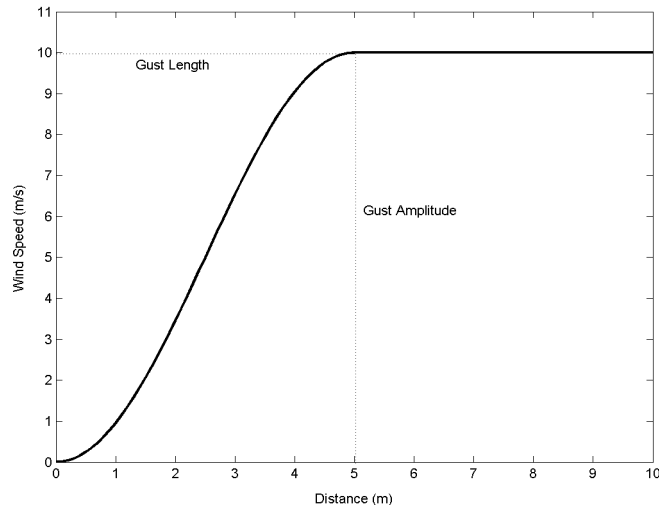
Description



The Discrete Wind Gust Model block implements a wind gust of the standard “1-cosine” shape. This block implements the mathematical representation in the Military Specification MIL-F-8785C [1]. The gust is applied to each axis individually, or to all three axes at once. The user specifies the gust amplitude (the increase in wind speed generated by the gust), the gust length (length, in meters, over which the gust builds up) and the gust start time.

The Discrete Wind Gust Model block can represent the wind speed in units of feet per second, meters per second, or knots.

The following figure shows the shape of the gust with a start time of zero. The parameters that govern the gust shape are indicated on the diagram.



The discrete gust can be used singly or in multiples to assess airplane response to large wind disturbances.

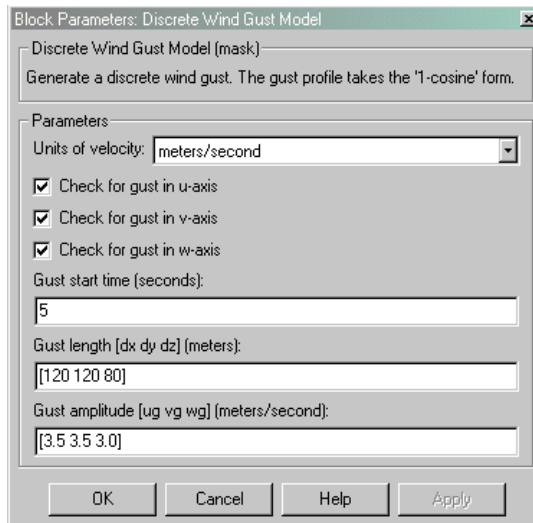
Discrete Wind Gust Model

The mathematical representation of the discrete gust is

$$V_{wind} = \begin{cases} 0 & x < 0 \\ \frac{V_m}{2} \left(1 - \cos\left(\frac{\pi x}{d_m}\right) \right) & 0 \leq x \leq d_m \\ V_m & x > d_m \end{cases}$$

where V_m is the gust amplitude, d_m is the gust length, x is the distance traveled, and V_{wind} is the resultant wind velocity in the body axis frame.

Dialog Box



Units of velocity

Define the units of wind gust.

	Wind	Altitude
Meters/second	Meters/second	Meters
Feet/second	Feet/second	Feet
Knots	Knots	Feet

Check for gust in u-axis

Select to apply the wind gust to the u-axis in the body frame.

Check for gust in v-axis

Select to apply the wind gust to the v-axis in the body frame.

Check for gust in w-axis

Select to apply the wind gust to the w-axis in the body frame.

Gust start time (seconds)

The model time, in seconds, at which the gust begins.

Gust length (meters or feet)

The length, in meters or feet (depending on the choice of units), over which the gust builds up in each axis.

Gust amplitude (meters/second, feet/second, or knots)

The magnitude of the increase in wind speed caused by the gust in each axis.

Inputs and Outputs

The input is altitude in units selected.

The output is wind speed in units selected.

Examples

See `aerob1k_HL20.mdl` example included with the blockset.

References

Military Specification MIL-F-8785C, 5th November 1980.

See Also

Dryden Wind Turbulence Model
Wind Shear Model

Dryden Wind Turbulence Model

Purpose Generate wind turbulence with the Dryden velocity spectra

Library Environment/Wind

Description

>Altitude (m)	Wind velocity (m/s)>
>Airspeed (m/s)	Angular rates (rad/sec)>

The Dryden Wind Turbulence Model block uses the Dryden spectral representation to add turbulence to the aerospace model by passing band-limited white noise through appropriate forming filters. This block implements the mathematical representation in the Military Specification MIL-F-8785C [1].

Turbulence can be considered as a stochastic process defined by velocity spectra. For an aircraft flying at a speed V through a “frozen” turbulence field with a spatial frequency of Ω radians per meter, the circular frequency ω is calculated by multiplying V by Ω . The appropriate component spectra for the Dryden model of turbulence are shown here.

Longitudinal:

$$\Phi_u(\omega) = \frac{2\sigma_u^2 L_u}{\pi V} \cdot \frac{1}{1 + (L_u \frac{\omega}{V})^2}$$

$$\Phi_{p_g}(\omega) = \frac{\sigma_w^2}{VL_w} \cdot \frac{0.8 \left(\frac{\pi L_w}{4b}\right)^{\frac{1}{3}}}{1 + \left(\frac{4b}{\pi} \frac{\omega}{V}\right)}$$

Lateral:

$$\Phi_v(\omega) = \frac{\sigma_v^2 L_v}{\pi V} \cdot \frac{1 + 3(L_v \frac{\omega}{V})^2}{[1 + (L_v \frac{\omega}{V})^2]^2}$$

$$\Phi_r(\omega) = \frac{\left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{3b\omega}{\pi V}\right)^2} \cdot \Phi_v(\omega)$$

Vertical:

$$\Phi_w(\omega) = \frac{\sigma_w^2 L_w}{\pi V} \cdot \frac{1 + 3(L_w \frac{\omega}{V})^2}{[1 + (L_w \frac{\omega}{V})^2]^2}$$

$$\Phi_q(\omega) = \frac{\left(\frac{\omega}{V}\right)^2}{1 + \left(\frac{4b\omega}{\pi V}\right)^2} \cdot \Phi_w(\omega)$$

b is the aircraft wingspan, L_u, L_v, L_w are the turbulence scale lengths, and $\sigma_u, \sigma_v, \sigma_w$ are the turbulence intensities.

To generate a signal with the correct characteristics a unit variance band-limited white noise signal is passed through appropriate forming filters that are derived by taking the spectral square roots of the spectrum equations. The resulting transfer functions are shown here.

Longitudinal:

$$H_u(s) = \sigma_u \sqrt{\frac{2L_u}{\pi V}} \frac{1}{1 + \frac{L_u}{V}s}$$

$$H_p(s) = \sigma_w \sqrt{\frac{0.8}{V}} \frac{(\pi/(4b))^{1/6}}{L_w^{1/3} \left(1 + \left(\frac{4b}{\pi}\right)s\right)}$$

Lateral:

Dryden Wind Turbulence Model

$$H_v(s) = \sigma_v \sqrt{\frac{L_v}{\pi V}} \frac{1}{\left(1 + \frac{L_v}{V} s\right)^2}$$

$$H_r(s) = \frac{s/V}{\left(1 + \left(\frac{3b}{\pi V}\right)s\right)} \cdot H_v(s)$$

Vertical:

$$H_w(s) = \sigma_w \sqrt{\frac{L_w}{\pi V}} \frac{1}{\left(1 + \frac{L_w}{V} s\right)^2}$$

$$H_q(s) = \frac{s/V}{\left(1 + \left(\frac{4b}{\pi V}\right)s\right)} \cdot H_w(s)$$

The turbulence scale lengths and intensities are functions of altitude, and there are two distinct regions.

Low-Altitude Model (Altitude < 1000 feet)

In [1] the turbulence scale lengths at low altitudes are as given below, where h is the altitude in feet.

$$L_w = h$$

$$L_u = L_v = \frac{h}{(0.177 + 0.000823h)^{1.2}}$$

The turbulence intensities are given below, where W_{20} is the wind speed at 20 feet (6 m). Typically for “light” turbulence the wind speed at 20 feet is 15 knots, for “moderate” turbulence the wind speed is 30 knots, and for “severe” turbulence the wind speed is 45 knots.

$$\sigma_w = 0.1W_{20}$$

$$\frac{\sigma_u}{\sigma_w} = \frac{\sigma_v}{\sigma_w} = \frac{1}{(0.177 + 0.000823h)^{0.4}}$$

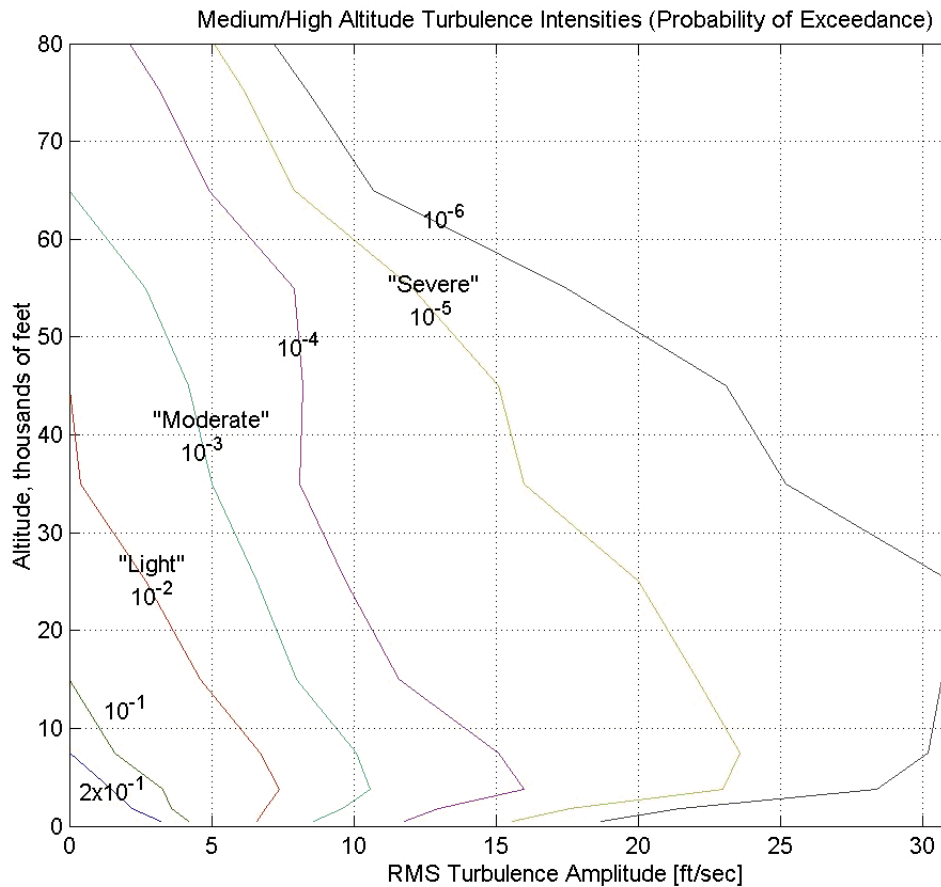
Dryden Wind Turbulence Model

Medium/High Altitudes (Altitude > 2000 feet)

For medium to high altitudes the turbulence scale lengths and intensities are based on the assumption that the turbulence is isotropic. In reference [1] the scale lengths are as given below:

$$L_u = L_v = L_w = 1750 \text{ ft}$$

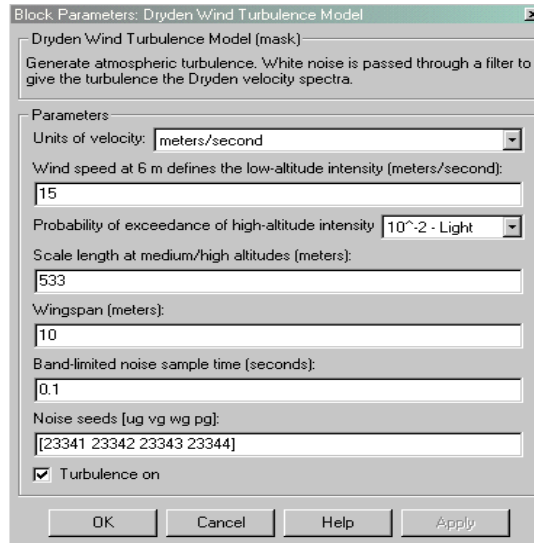
The turbulence intensities are determined from a lookup table that gives the turbulence intensity as a function of altitude and the probability of the turbulence intensity's being exceeded.



Dryden Wind Turbulence Model

At altitudes between 1000 feet and 2000 feet, the turbulence scale lengths and intensities are determined by interpolating between the value from the low-altitude model at 1000 feet and the value from the high-altitude model at 2000 feet.

Dialog Box



Units of Velocity

Define the units of wind speed due to the turbulence.

	Wind Velocity	Altitude	Air Speed
Meters/second	Meters/second	Meters	Meters/second
Feet/second	Feet/second	Feet	Feet/second
Knots	Knots	Feet	Knots

Wind speed at 20 ft (6 m) defines the low altitude intensity

The measured wind speed at a height of 20 feet provides the intensity for the low-altitude turbulence model.

Dryden Wind Turbulence Model

Probability of exceedance of high altitude intensity

Above 2000 feet, the turbulence intensity is determined from a lookup table that gives the turbulence intensity as a function of altitude and the probability of the turbulence intensity's being exceeded.

Scale length at medium/high altitudes

The turbulence scale length above 2000 feet is assumed constant, and from reference [1] a figure of 1750 feet is recommended.

Wingspan

The wingspan is required in the calculation of the turbulence on the angular rates.

Band-limited noise sample time (seconds)

The sample time at which the unit variance white noise signal is generated.

Noise seeds

There are four random numbers required to generate the turbulence signals, one for each of the three velocity components and one for the roll rate. The turbulences on the pitch and yaw angular rates are based on further shaping of the outputs from the shaping filters for the vertical and lateral velocities.

Turbulence on

Selecting the check box generates the turbulence signals.

Inputs and Outputs

The first input is altitude, in units selected.

The second input is aircraft speed, in units selected.

The first output is a three-element signal containing the turbulence velocities, in the selected units.

The second output is a three-element signal containing the turbulence angular rates, in radians per second.

Examples

See the `aeroblk_HL20.mdl` example included with the blockset.

References

Military Specification MIL-F-8785C, 5th November, 1980.

See Also

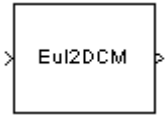
Discrete Wind Gust Model
Wind Shear Model

Euler Angles to Direction Cosine Matrix

Purpose Convert Euler angles to direction cosine matrix

Library Transformations/Axes

Description



The Euler Angles to Direction Cosine Matrix block converts the three Euler rotation angles into a 3-by-3 direction cosine matrix (DCM). The DCM matrix performs the coordinate transformation of a vector in inertial axes (ox_0, oy_0, oz_0) into a vector in body axes (ox_3, oy_3, oz_3) . The order of the axis rotations required to bring (ox_3, oy_3, oz_3) into coincidence with (ox_0, oy_0, oz_0) is first a rotation about ox_3 through the roll angle (ϕ) to axes (ox_2, oy_2, oz_2) . Second a rotation about oy_2 through the pitch angle (θ) to axes (ox_1, oy_1, oz_1) , and finally a rotation about oz_1 through the yaw angle (ψ) to axes (ox_0, oy_0, oz_0) .

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = DCM \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

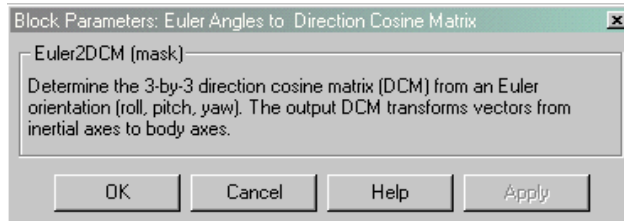
$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

Combining the three axis transformation matrices defines the following DCM.

$$DCM = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi) & (\sin \phi \sin \theta \sin \psi - \cos \phi \cos \psi) & \sin \phi \cos \theta \\ (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi) & (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi) & \cos \phi \cos \theta \end{bmatrix}$$

Euler Angles to Direction Cosine Matrix

Dialog Box



Inputs and Outputs

The input is a 3-by-1 vector of Euler angles.

The output is a 3-by-3 direction cosine matrix.

Examples

See `aeroblk_six_dof.mdl` to see the use of the Euler Angles to Direction Cosine Matrix block in the implementation of the equations of motion for a rigid body.

See Also

Direction Cosine Matrix to Euler Angles
Direction Cosine Matrix to Quaternions
Euler Angles to Quaternions
Quaternions to Direction Cosine Matrix
Quaternions to Euler Angles

Euler Angles to Quaternions

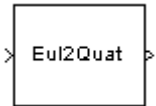
Purpose

Convert Euler angles to a quaternion vector

Library

Transformations/Axes

Description



The Euler Angles to Quaternions block converts the rotation described by the three Euler angles (roll, pitch, yaw) into the four-element quaternion vector (q_0, q_1, q_2, q_3) .

A quaternion vector represents a rotation about a unit vector (μ_x, μ_y, μ_z) through an angle θ . A unit quaternion itself has unit magnitude, and can be written in the following vector format.

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\mu_x \\ \sin(\theta/2)\mu_y \\ \sin(\theta/2)\mu_z \end{bmatrix}$$

An alternative representation of a quaternion is as a complex number,

$$q = q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3$$

where, for the purposes of multiplication,

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

$$\mathbf{i}\mathbf{j} = -\mathbf{j}\mathbf{i} = \mathbf{k}$$

$$\mathbf{j}\mathbf{k} = -\mathbf{k}\mathbf{j} = \mathbf{i},$$

$$\mathbf{k}\mathbf{i} = -\mathbf{i}\mathbf{k} = \mathbf{j}$$

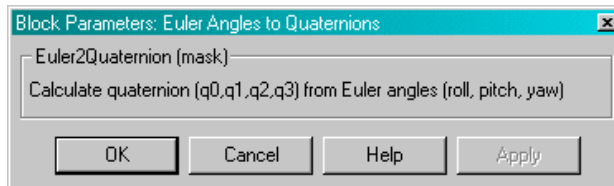
The benefit of representing the quaternion in this way is the ease with which the quaternion product can represent the resulting transformation after two or more rotations. The quaternion to represent the rotation through the three Euler angles is given below.

$$q = q_\phi q_\theta q_\psi = \left(\cos\left(\frac{\phi}{2}\right) - \mathbf{i} \sin\left(\frac{\phi}{2}\right) \right) \left(\cos\left(\frac{\theta}{2}\right) - \mathbf{j} \sin\left(\frac{\theta}{2}\right) \right) \left(\cos\left(\frac{\psi}{2}\right) - \mathbf{k} \sin\left(\frac{\psi}{2}\right) \right)$$

Expanding the preceding representation gives the four quaternion elements following.

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right) \cos\left(\frac{\theta}{2}\right) \cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right) \sin\left(\frac{\theta}{2}\right) \sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right) \cos\left(\frac{\theta}{2}\right) \cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right) \sin\left(\frac{\theta}{2}\right) \sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right) \sin\left(\frac{\theta}{2}\right) \cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right) \cos\left(\frac{\theta}{2}\right) \sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right) \cos\left(\frac{\theta}{2}\right) \sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right) \sin\left(\frac{\theta}{2}\right) \sin\left(\frac{\psi}{2}\right) \end{bmatrix}$$

Dialog Box



Inputs and Outputs

The input is a 3-by-1 vector of Euler angles.

The output is a 4-by-1 quaternion vector.

See Also

Direction Cosine Matrix to Euler Angles
Direction Cosine Matrix to Quaternions
Euler Angles to Direction Cosine Matrix
Quaternions to Direction Cosine Matrix
Quaternions to Euler Angles

Equations of Motion

Purpose

Implement three-degrees-of-freedom equations of motion

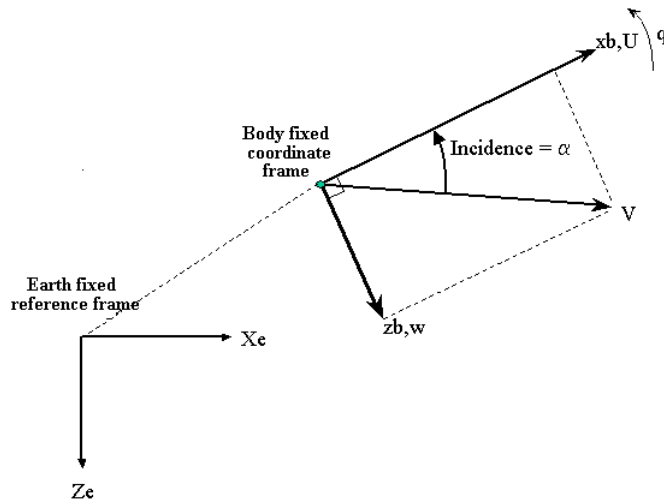
Library

Equations of Motion/3DoF

Description

	Attitude (rad)	>
> Fx (N)	q (rad/s)	>
	qdot (rad/s ²)	>
> Fz (N)	x _e , z _e (m)	>
	U, w (m/s)	>
> M (N-m)	A _x , A _z (m/s ²)	>

The 3DoF Equations of Motion block considers the rotation in the vertical plane of a body-fixed coordinate frame about an Earth-fixed reference frame.



The equations of motion are given below:

$$\dot{u} = \frac{F_x}{m} - qw - g \sin \theta$$

$$\dot{w} = \frac{F_z}{m} + qu + g \cos \theta$$

$$\dot{q} = \frac{M}{I_{yy}}$$

$$\dot{\theta} = q$$

where the applied forces are assumed to act at the center of gravity of the body.

Dialog Box

Block Parameters: Equations of Motion (Body Axes)

3DoF EoM (mask)

Integrate the three-degrees-of-freedom equations of motion to determine body position, velocity, attitude, and related values.

Inputs are forces on the body in x-axis, forces on the body in z-axis, and moments on the body along y-axis. Outputs are pitch angle, pitch rate, pitch acceleration, position in x-axis and z-axis in Earth coordinates, translational velocity in x-axis and z-axis, and acceleration in x-axis and z-axis due to body forces.

Parameters

Initial velocity [m/s]:
100

Initial body attitude [rad]:
0

Initial incidence [rad]:
0

Initial body rotation rate [rad/sec]:
0

Initial position [x z] [m]:
[0 0]

Mass [Kg]:
1

Inertia [Kg.m²):
1

Acceleration due to gravity [m/s/s]:
9.81

OK Cancel Help Apply

Initial velocity [m/s]

A scalar value for the initial velocity of the body, (V_0).

Initial body attitude [rad]

A scalar value for the initial pitch attitude of the body, (θ_0).

Initial incidence [rad]

A scalar value for the initial angle between the velocity vector and the body, (α_0).

Initial body rotation rate [rad/sec]

A scalar value for the initial body rotation rate, (q_0).

Equations of Motion

Initial position (x,z) [m]

A two-element vector containing the initial location of the body in the Earth-fixed reference frame.

Mass [kg]

A scalar value for the mass of the body.

Inertia [Kg.m²]

A scalar value for the inertia of the body.

Acceleration due to gravity [m/s²]

A scalar value for the acceleration due to gravity. If gravity is to be neglected in the simulation, this value can be set to 0.

Inputs and Outputs

The first input to the block is the force acting along the body x-axis, in Newtons (F_x).

The second input to the block is the force acting along the body z-axis, in Newtons (F_z).

The third input to the block is the applied pitch moment, in Newton.meters (M).

The first output from the block is the pitch attitude, in radians (θ).

The second output is the pitch angular rate, in radians per second (q).

The third output is the pitch angular acceleration, in radians per second squared (\dot{q}).

The fourth output is a two-element vector containing the location of the body, in the Earth-fixed reference frame, in meters (X_e, Z_e).

The fifth output is a two-element vector containing the velocity of the body resolved into the body-fixed coordinate frame, in meters per second (u, w).

The sixth output is a two-element vector containing the acceleration of the body resolved into the body-fixed coordinate frame, in meters per second squared (A_x, A_z).

Examples

See the Simulink model `aeroblk_guidance.mdl` for an example of the use of the 3DoF Equations of Motion block.

See Also

Incidence & Airspeed

Force Conversion

Purpose Convert from force units to desired force units

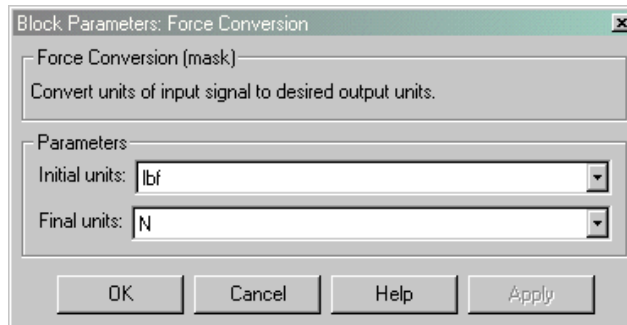
Library Transformations/Units

Description The Force Conversion block computes the conversion factor from specified input force units to specified output force units and applies the conversion factor to the input signal.



The Force Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

lbf	Pound force
N	Newtons

Inputs and Outputs

The input is force in initial force units.

The output is force in final force units.

See Also

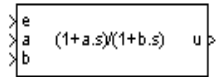
Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Gain Scheduled Lead-Lag

Purpose Implement a first-order lead-lag with gain-scheduled coefficients

Library GNC

Description The Gain Scheduled Lead-Lag block implements a first-order lag of the form

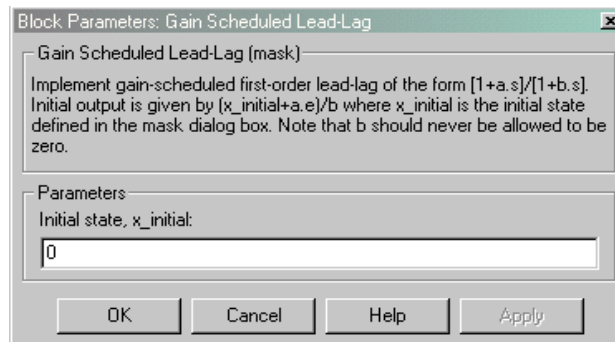


$$u = \frac{1 + as}{1 + bs}e$$

where e is the filter input, and u the filter output.

The coefficients a and b are inputs to the block, and hence can be made dependent on flight condition or operating point. For example, they could be produced from the Look-Up Table (n-D) block.

Dialog Box



Initial state, $x_{initial}$

The initial internal state for the filter $x_{initial}$. Given this initial state, the initial output is given by

$$u|_{t=0} = \frac{x_{initial} + ae}{b}$$

Inputs and Outputs

The first input is the filter input.

The second input is the numerator coefficient.

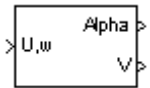
The third input is the denominator coefficient.

The output is the filter output

Purpose Calculate incidence and air speed

Library Equations of Motion/3DoF

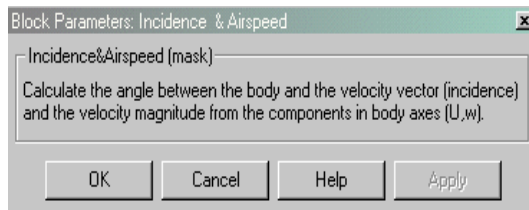
Description



The Incidence & Airspeed block supports the 3DoF equations of motion model by calculating the angle between the velocity vector and the body, and also the total air speed from the velocity components in the body-fixed coordinate frame.

$$\alpha = \operatorname{atan}\left(\frac{w}{u}\right)$$
$$V = \sqrt{u^2 + w^2}$$

Dialog Box



Inputs and Outputs

The input to the block is the two-element vector containing the velocity of the body resolved into the body-fixed coordinate frame.

The first output of the block is the incidence angle, in radians.

The second output is the velocity of the body.

Examples

See the Simulink demo `aeroblk_guidance.mdl` for an example of the use of this block.

See Also

Equations of Motion

Interpolate Matrix(x)

Purpose Return an interpolated matrix for given input x

Library GNC

Description The Interpolate Matrix(x) block interpolates a one-dimensional array of matrices.

>x Matrix(x)>

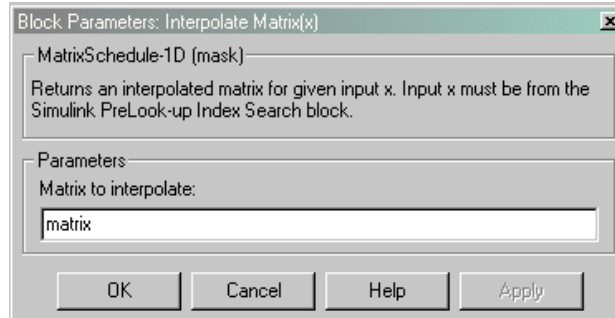
This one-dimensional case assumes a matrix M is defined at a discrete number of values of an independent variable $\mathbf{x} = [x_1 \ x_2 \ x_3 \ \dots \ x_i \ x_{i+1} \ \dots \ x_n]$. Then for $x_i < x < x_{i+1}$, the block output is given by

$$(1 - \lambda)M(x_i) + \lambda M(x_{i+1})$$

where the interpolation fraction is defined as

$$\lambda = (x - x_i) / (x_{i+1} - x_i)$$

Dialog Box and Parameters



Matrix to interpolate

Matrix to be interpolated. It should be three dimensional, the first two dimensions corresponding to the matrix at each value of x . For example, if you have three matrices A, B, and C defined at $x = 0$, $x = 0.5$, and $x = 1.0$, then the input matrix is given by

```
matrix(:,:,1) = A;  
matrix(:,:,2) = B;  
matrix(:,:,3) = C;
```


Inputs and Outputs

The first input is the first independent variable.

The output is the interpolated matrix.

Assumptions and Limitations

This block must be driven from the Simulink PreLookup Index Search block.

Examples

See the following Aerospace Blockset blocks: 1D Controller

[A(v),B(v),C(v),D(v)], 1D Observer Form [A(v),B(v),C(v),F(v),H(v)], and 1D

Self-Conditioned [A(v),B(v),C(v),D(v)].

See Also

Interpolate Matrix(x,y)

Interpolate Matrix(x,y,z)

Interpolate Matrix(x,y)

Purpose Return an interpolated matrix for given inputs x and y

Library GNC

Description The Interpolate Matrix(x,y) block interpolates a two-dimensional array of matrices.



This two-dimensional case assumes the matrix is defined as a function of two independent variables, $\mathbf{x} = [x_1 x_2 x_3 \dots x_i x_{i+1} \dots x_n]$ and $\mathbf{y} = [y_1 y_2 y_3 \dots y_j y_{j+1} \dots y_m]$. For given values of x and y , four matrices are interpolated. Then for $x_i < x < x_{i+1}$ and $y_j < y < y_{j+1}$, the output matrix is given by

$$(1 - \lambda_y)[(1 - \lambda_x)M(x_i, y_j) + \lambda_x M(x_{i+1}, y_j)] + \lambda_y [(1 - \lambda_x)M(x_i, y_{j+1}) + \lambda_x M(x_{i+1}, y_{j+1})]$$

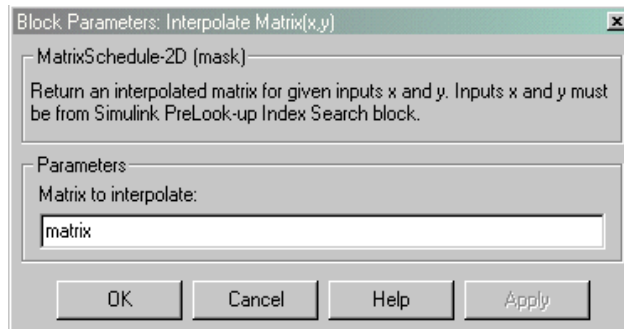
where the two interpolation fractions are denoted by

$$\lambda_x = (x - x_i) / (x_{i+1} - x_i)$$

and

$$\lambda_y = (y - y_j) / (y_{j+1} - y_j)$$

Dialog Box and Parameters



Matrix to interpolate

Matrix to be interpolated. It should be four dimensional, the first two dimensions corresponding to the matrix at each value of x and y . For example, if you have four matrices A, B, C, and D defined at

(x = 0.0,y = 1.0), (x = 0.0,y = 3.0), (x = 1.0,y = 1.0) and (x = 1.0,y = 3.0), then the input matrix is given by

matrix(:,:,1,1) = A;

matrix(:,:,1,2) = B;

matrix(:,:,2,1) = C;

matrix(:,:,2,2) = D;

Inputs and Outputs

The first input is the first independent variable.

The second input is the second independent variable.

The output is the interpolated matrix.

Assumptions and Limitations

This block must be driven from the Simulink PreLookup Index Search block.

Examples

See the following Aerospace Blockset blocks: 2D Controller [A(v),B(v),C(v),D(v)], 2D Observer Form [A(v),B(v),C(v),F(v),H(v)], and 2D Self-Conditioned [A(v),B(v),C(v),D(v)].

See Also

Interpolate Matrix(x)
Interpolate Matrix(x,y,z)

Interpolate Matrix(x,y,z)

Purpose

Return an interpolated matrix for given inputs x, y, and z

Library

GNC

Description

```
>x  
>y Matrix(x,y,z) >  
>z
```

The Interpolate Matrix(x,y,z) block interpolates a three-dimensional array of matrices.

This three-dimensional case assumes the matrix is defined as a function of three independent variables, $\mathbf{x} = [x_1 x_2 x_3 \dots x_i x_{i+1} \dots x_n]$, $\mathbf{y} = [y_1 y_2 y_3 \dots y_j y_{j+1} \dots y_m]$, and $\mathbf{z} = [z_1 z_2 z_3 \dots z_k z_{k+1} \dots z_p]$. For given values of x, y, and z, eight matrices are interpolated. Then for $x_i < x < x_{i+1}$, $y_j < y < y_{j+1}$ and $z_k < z < z_{k+1}$, the output matrix is given by

$$\begin{aligned} & (1-\lambda_z) \{ (1-\lambda_y) [(1-\lambda_x)M(x_i, y_j, z_k) + \lambda_x M(x_{i+1}, y_j, z_k)] + \\ & \quad \lambda_y [(1-\lambda_x)M(x_i, y_{j+1}, z_k) + \lambda_x M(x_{i+1}, y_{j+1}, z_k)] \} \\ & + \lambda_z \{ (1-\lambda_y) [(1-\lambda_x)M(x_i, y_j, z_{k+1}) + \lambda_x M(x_{i+1}, y_j, z_{k+1})] + \\ & \quad \lambda_y [(1-\lambda_x)M(x_i, y_{j+1}, z_{k+1}) + \lambda_x M(x_{i+1}, y_{j+1}, z_{k+1})] \} \end{aligned}$$

where the three interpolation fractions are denoted by

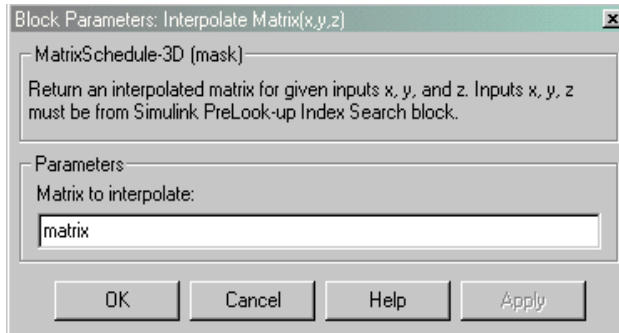
$$\lambda_x = (x - x_i) / (x_{i+1} - x_i)$$

$$\lambda_y = (y - y_j) / (y_{j+1} - y_j)$$

$$\lambda_z = (z - z_k) / (z_{k+1} - z_k)$$

In the three-dimensional case, the interpolation is carried out first on x, then y, and finally z.

Dialog Box and Parameters



Matrix to interpolate

Matrix to be interpolated. It should be five dimensional, the first two dimensions corresponding to the matrix at each value of x, y, and z. For example, if you have eight matrices A, B, C, D, E, F, G, and H defined at the following values of x, y, and z, then the corresponding input matrix is given by

(x = 0.0, y = 1.0, z = 0.1)	matrix(:, :, 1, 1, 1) = A;
(x = 0.0, y = 1.0, z = 0.5)	matrix(:, :, 1, 1, 2) = B;
(x = 0.0, y = 3.0, z = 0.1)	matrix(:, :, 1, 2, 1) = C;
(x = 0.0, y = 3.0, z = 0.5)	matrix(:, :, 1, 2, 2) = D;
(x = 1.0, y = 1.0, z = 0.1)	matrix(:, :, 2, 1, 1) = E;
(x = 1.0, y = 1.0, z = 0.5)	matrix(:, :, 2, 1, 2) = F;
(x = 1.0, y = 3.0, z = 0.1)	matrix(:, :, 2, 2, 1) = G;
(x = 1.0, y = 3.0, z = 0.5)	matrix(:, :, 2, 2, 2) = H;

Inputs and Outputs

The first input is the first independent variable.

The second input is the second independent variable.

The third input is the third independent variable.

The output is the interpolated matrix.

Interpolate Matrix(x,y,z)

Assumptions and Limitations

This block must be driven from the Simulink PreLookup Index Search block.

Examples

See the following Aerospace Blockset blocks: 3D Controller [A(v),B(v),C(v),D(v)], 3D Observer Form [A(v),B(v),C(v),F(v),H(v)], and 3D Self-Conditioned [A(v),B(v),C(v),D(v)].

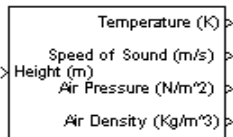
See Also

Interpolate Matrix(x)
Interpolate Matrix(x,y)

Purpose Implement the International Standard Atmosphere (ISA)

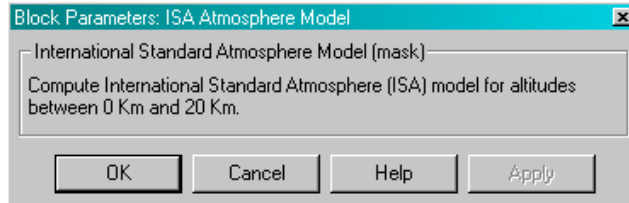
Library Environment/Atmosphere

Description The ISA Atmosphere Model block implements the mathematical representation of the international standard atmosphere values for absolute temperature, pressure, density, and speed of sound for the input geopotential altitude.



The ISA Atmosphere Model block icon displays the input and output metric units.

Dialog Box



Inputs and Outputs The input is geopotential height.

The four outputs are temperature, speed of sound, air pressure, and air density.

Assumptions and Limitations Below the geopotential altitude of 0 Km and above the geopotential altitude of 20 Km, temperature and pressure values are held. Density and speed of sound are calculated using a perfect gas relationship.

References [1] U.S. Standard Atmosphere, 1976, U.S. Government Printing Office, Washington, D.C.

See Also COESA Atmosphere Model

Length Conversion

Purpose Convert from length units to desired length units

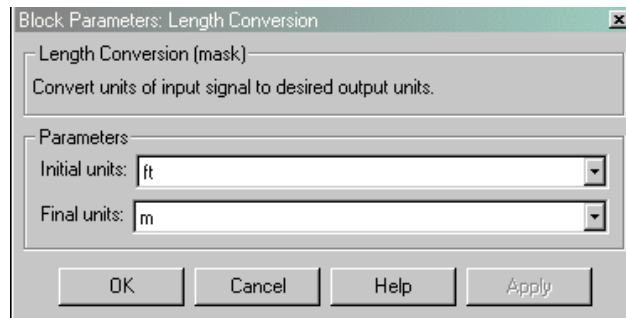
Library Transformations/Units

Description The Length Conversion block computes the conversion factor from specified input length units to specified output length units and applies the conversion factor to the input signal.



The Length Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

m	Meters
ft	Feet
km	Kilometers
in	Inches
mi	Miles
naut mi	Nautical miles

Inputs and Outputs

The input is length in initial length units.

The output is length in final length units.

See Also

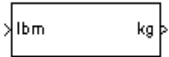
Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Force Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Mass Conversion

Purpose Convert from mass units to desired mass units

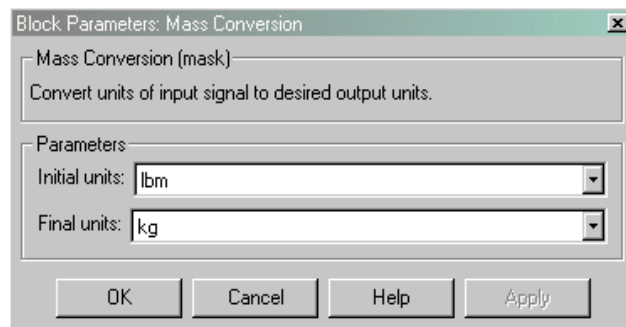
Library Transformations/Units

Description The Mass Conversion block computes the conversion factor from specified input mass units to specified output mass units and applies the conversion factor to the input signal.



The Mass Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

lbm Pound mass

kg Kilograms

slug

Inputs and Outputs

The input is mass in initial mass units.

The output is mass in final mass units.

See Also

Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Length Conversion
Force Conversion
Pressure Conversion
Temperature Conversion
Velocity Conversion

Pressure Conversion

Purpose Convert from pressure units to desired pressure units

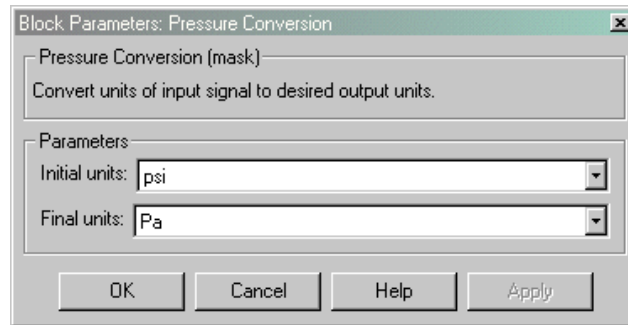
Library Transformations/Units

Description The Pressure Conversion block computes the conversion factor from specified input pressure units to specified output pressure units and applies the conversion factor to the input signal.



The Pressure Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units
Specifies the input units.

Final units
Specifies the output units.

The following conversion units are available:

psi	Pound mass per square inch
Pa	Pascals
psf	Pound mass per square foot
atm	Atmospheres

Inputs and Outputs

The input is pressure in initial pressure units.

The output is pressure in final pressure units.

See Also

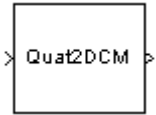
- Acceleration Conversion
- Angle Conversion
- Angular Acceleration Conversion
- Angular Velocity Conversion
- Density Conversion
- Force Conversion
- Length Conversion
- Mass Conversion
- Temperature Conversion
- Velocity Conversion

Quaternions to Direction Cosine Matrix

Purpose Convert quaternion vector to direction cosine matrix

Library Transformations/Axes

Description



The Quaternions to Direction Cosine Matrix block transforms the four-element unit quaternion vector (q_0, q_1, q_2, q_3) into a 3-by-3 direction cosine matrix (DCM). The outputted DCM performs the coordinate transformation of a vector in inertial axes to a vector in body axes.

Using quaternion algebra, if a point P is subject to the rotation described by a quaternion q, it changes to P' given by the following relationship:

$$\begin{aligned}P' &= qPq^c \\q &= q_0 + \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 \\q^c &= q_0 - \mathbf{i}q_1 - \mathbf{j}q_2 - \mathbf{k}q_3 \\P &= 0 + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z\end{aligned}$$

Expanding P' and collecting terms in x, y, and z gives the following for P' in terms of P in the vector quaternion format.

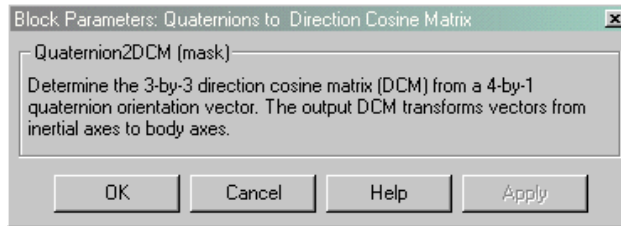
$$P' = \begin{bmatrix} 0 \\ x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 0 \\ (q_0^2 + q_1^2 - q_2^2 - q_3^2)x + 2(q_1q_2 - q_0q_3)y + 2(q_1q_3 + q_0q_2)z \\ 2(q_0q_3 + q_1q_2)x + (q_0^2 - q_1^2 + q_2^2 - q_3^2)y + 2(q_2q_3 - q_0q_1)z \\ 2(q_1q_3 - q_0q_2)x + 2(q_0q_1 + q_2q_3)y + (q_0^2 - q_1^2 - q_2^2 + q_3^2)z \end{bmatrix}$$

Since individual terms in P' are linear combinations of terms in x, y, and z, a matrix relationship to rotate the vector (x, y, z) to (x', y', z') can be extracted from the preceding. This matrix rotates a vector in inertial axes, and hence is transposed to generate the DCM that performs the coordinate transformation of a vector in inertial axes into body axes.

Quaternions to Direction Cosine Matrix

$$DCM = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix}$$

Dialog Box



Inputs and Outputs

The input is a 4-by-1 quaternion vector.

The output is a 3-by-3 direction cosine matrix.

Examples

See `aeroblk_six_dof.mdl` for an example of the use of the Quaternions to Direction Cosine Matrix block in an implementation of the equations of motion of a rigid body.

See Also

Direction Cosine Matrix to Euler Angles
Direction Cosine Matrix to Quaternions
Euler Angles to Direction Cosine Matrix
Euler Angles to Quaternions
Quaternions to Euler Angles

Quaternions to Euler Angles

Purpose Convert quaternion vector to Euler angles

Library Transformations/Axes

Description



The Quaternions to Euler Angles block converts the four-element unit quaternion (q_0, q_1, q_2, q_3) into the equivalent three Euler angle rotations (roll, pitch, yaw).

The conversion is generated by comparing elements in the direction cosine matrix (DCM), as functions of the Euler rotation angles, with elements in the DCM, as functions of a unit quaternion vector.

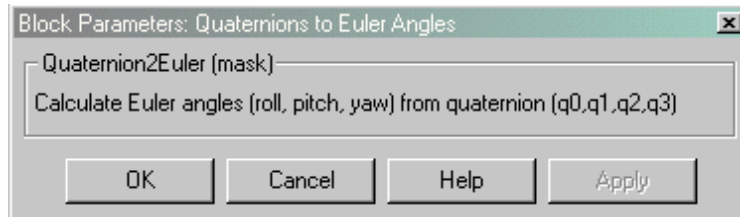
$$DCM = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi) & (\sin \phi \sin \theta \sin \psi - \cos \phi \cos \psi) & \sin \phi \cos \theta \\ (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi) & (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi) & \cos \phi \cos \theta \end{bmatrix}$$

$$DCM = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix}$$

From the preceding, you can derive the following relationships between DCM elements and individual Euler angles:

$$\begin{aligned} \phi &= \text{atan}(DCM(2, 3), DCM(3, 3)) \\ &= \text{atan}(2(q_2q_3 + q_0q_1), (q_0^2 - q_1^2 - q_2^2 + q_3^2)) \\ \theta &= \text{asin}(-DCM(1, 3)) \\ &= \text{asin}(-2(q_1q_3 - q_0q_2)) \\ \psi &= \text{atan}(DCM(1, 2), DCM(1, 1)) \\ &= \text{atan}(2(q_1q_2 + q_0q_3), (q_0^2 + q_1^2 - q_2^2 - q_3^2)) \end{aligned}$$

Dialog Box



Inputs and Outputs

The input is a 4-by-1 quaternion vector.

The output is a 3-by-1 vector of Euler angles.

Assumptions and Limitations

This implementation generates a pitch angle that lies between ± 90 degrees, and roll and yaw angles that lie between ± 180 degrees.

Examples

See `aero_six_dof.mdl` for an example of the use of the Quaternions to Euler Angles block in an implementation of the equations of motion of a rigid body.

See Also

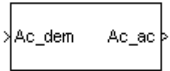
Direction Cosine Matrix to Euler Angles
Direction Cosine Matrix to Quaternions
Euler Angles to Direction Cosine Matrix
Euler Angles to Quaternions
Quaternions to Direction Cosine Matrix

Second Order Linear Actuator

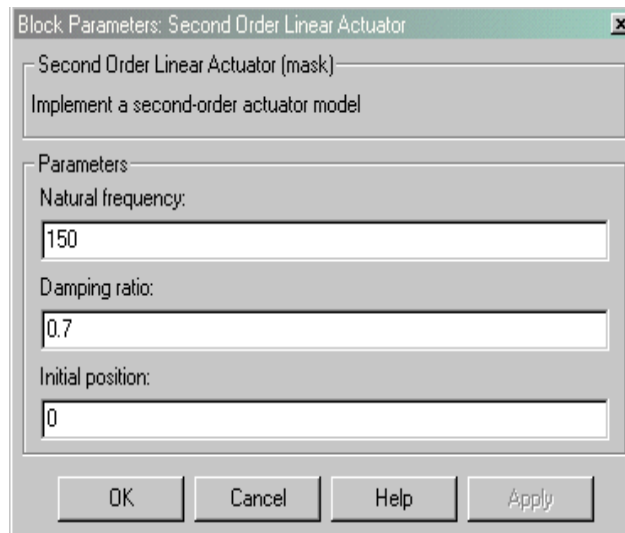
Purpose Implement a second-order linear actuator

Library Actuators

Description The Second Order Linear Actuator block outputs the actual actuator position using the input demanded actuator position and other dialog parameters that define the system.



Dialog Box



Natural frequency

The natural frequency of the actuator. The units of natural frequency are radians per second.

Damping ratio

The damping ratio of the actuator. A dimensionless parameter.

Initial position

The initial position of the actuator. The units of initial position should be the same as the units of demanded actuator position.

Second Order Linear Actuator

Inputs and Outputs

The input is the demanded actuator position.

The output is the actual actuator position.

See Also

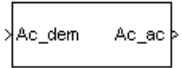
Second Order Nonlinear Actuator

Second Order Nonlinear Actuator

Purpose Implement a second-order actuator with rate and deflection limits

Library Actuators

Description The Second Order Nonlinear Actuator block outputs the actual actuator position using the input demanded actuator position and other dialog parameters that define the system.



Dialog Box

The dialog box is titled "Block Parameters: Second Order Nonlinear Actuator". It contains the following parameters:

- Natural frequency: 150
- Damping ratio: 0.7
- Maximum deflection: $20 \cdot \pi / 180$
- Minimum deflection: $-20 \cdot \pi / 180$
- Maximum rate: $500 \cdot \pi / 180$
- Initial position: 0

Buttons: OK, Cancel, Help, Apply

Second Order Nonlinear Actuator

Natural frequency

The natural frequency of the actuator. The units of natural frequency are radians per second.

Damping ratio

The damping ratio of the actuator. A dimensionless parameter.

Maximum deflection

The largest actuator position allowable. The units of maximum deflection should be the same as the units of demanded actuator position.

Minimum deflection

The smallest actuator position allowable. The units of minimum deflection should be the same as the units of demanded actuator position.

Maximum rate

The fastest speed allowable for actuator motion. The units of maximum rate should be the units of demanded actuator position per second.

Initial position

The initial position of the actuator. The units of initial position should be the same as the units of demanded actuator position.

Inputs and Outputs

The input is the demanded actuator position.

The output is the actual actuator position.

See Also

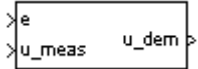
Second Order Linear Actuator

Self-Conditioned [A,B,C,D]

Purpose Implement a state-space controller in a self-conditioned form

Library GNC

Description The Self-Conditioned [A,B,C,D] block can be used to implement the state-space controller defined by



$$\begin{bmatrix} \dot{x} = Ax + Be \\ u = Cx + De \end{bmatrix}$$

in the self-conditioned form

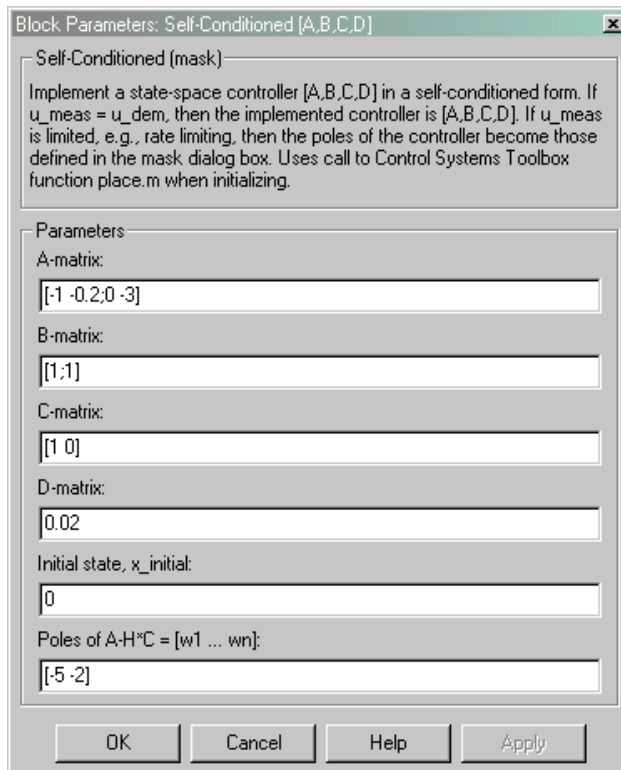
$$\dot{z} = (A - HC)z + (B - HD)e + Hu_{meas}$$

$$u_{dem} = Cz + De$$

The input u_{meas} is a vector of the achieved actuator positions, and the output u_{dem} is the vector of controller actuator demands. In the case that the actuators are not limited, then $u_{meas} = u_{dem}$ and substituting the output equation into the state equation returns the nominal controller. In the case that they are not equal, the dynamics of the controller are set by the poles of $A-HC$.

Hence H must be chosen to make the poles sufficiently fast to track u_{meas} but at the same time not so fast that noise on e is propagated to u_{dem} . The matrix H is designed by a callback to the Control Systems Toolbox command `place.m` to place the poles at defined locations.

Dialog Box



A-matrix

A-matrix of the state-space implementation.

B-matrix

B-matrix of the state-space implementation.

C-matrix

C-matrix of the state-space implementation.

D-matrix

D-matrix of the state-space implementation.

Self-Conditioned [A,B,C,D]

Initial state, x_{initial}

This is a vector of initial states for the controller, i.e., initial values for the state vector, z . It should have length equal to the size of the first dimension of A .

Poles of $A-H*C$

This is a vector of the desired poles of $A-H*C$. Hence the number of pole locations defined should be equal to the dimension of the A -matrix.

Inputs and Outputs

The first input is control error.

The second input is the measured actuator position.

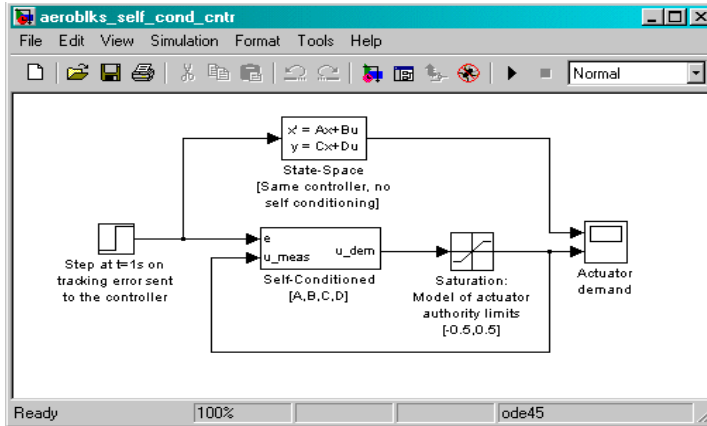
The output is the actuator demands.

Assumptions and Limitations

This block requires the Control Systems Toolbox.

Examples

This Simulink model shows a state-space controller implemented in both self-conditioned and standard state-space forms. The actuator authority limits of +/- 0.5 units are modeled by the saturation block.



Block Parameters: Self-Conditioned [A,B,C,D]

Self-Conditioned (mask) (link)

Implement a state-space controller [A,B,C,D] in a self-conditioned form. If $u_{meas} = u_{dem}$, then the implemented controller is [A,B,C,D]. If u_{meas} is limited, e.g., rate limiting, then the poles of the controller become those defined in the mask dialog box. Uses call to Control Systems Toolbox function `place.m` when initializing.

Parameters

A-matrix:
[0 -0.2; 0 -3]

B-matrix:
[1; 1]

C-matrix:
[1 0]

D-matrix:
0.02

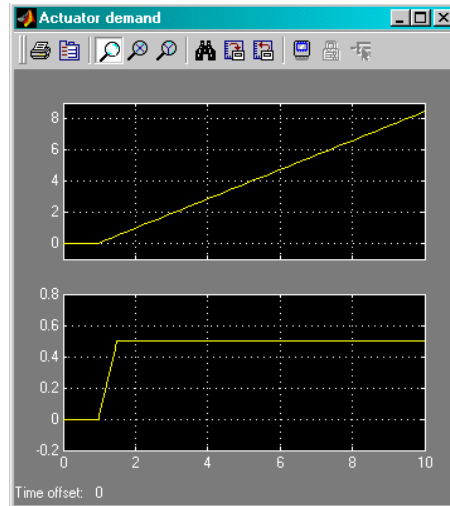
Initial state, $x_{initial}$:
0

Poles of $A-H^*C = [w1 \dots wn]$:
[-5 -2]

OK Cancel Help Apply

Self-Conditioned [A,B,C,D]

Notice that the A-matrix has a zero in the 1,1 element, indicating integral action.



The top trace shows the conventional state-space implementation. The output of the controller winds up well past the actuator upper authority limit of +0.5. The lower trace shows that the self-conditioned form results in an actuator demand that tracks the upper authority limit, which means that when the sign of the control error, e , is reversed, the actuator demand responds immediately.

References

The algorithm used to determine the matrix H is defined in Kautsky, Nichols, and Van Dooren, "Robust Pole Assignment in Linear State Feedback," *International Journal of Control*, Vol. 41, No. 5, pages 1129-1155, 1985.

See Also

1D Self-Conditioned [A(v),B(v),C(v),D(v)]
2D Self-Conditioned [A(v),B(v),C(v),D(v)]
3D Self-Conditioned [A(v),B(v),C(v),D(v)]

Purpose Convert from temperature units to desired temperature units

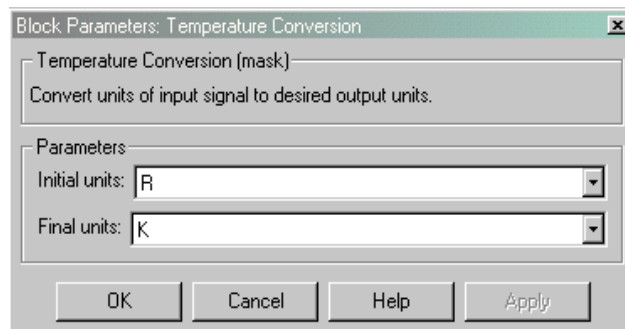
Library Transformations/Units

Description The Temperature Conversion block computes the conversion factor from specified input temperature units to specified output temperature units and applies the conversion factor to the input signal.



The Temperature Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

K	Degrees Kelvin
F	Degrees Fahrenheit
C	Degrees Celsius
R	Degrees Rankine

Inputs and Outputs

The input is temperature in initial temperature units.

The output is temperature in final temperature units.

Temperature Conversion

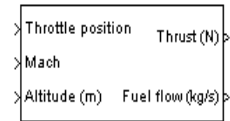
See Also

Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Velocity Conversion

Purpose Implement a first-order representation of a turbofan engine with controller

Library Propulsion

Description



The Turbofan Engine System block computes the thrust and the weight of fuel flow of a turbofan engine and controller at a specific throttle position, Mach number, and altitude.

This system is represented by a first-order system with unitless heuristic lookup tables for thrust, thrust specific fuel consumption (TSFC), and engine time constant. For the lookup table data, thrust is a function of throttle position and Mach number, TSFC is a function of thrust and Mach number, and engine time constant is a function of thrust. The unitless lookup table outputs are corrected for altitude using the relative pressure ratio δ and relative temperature ratio θ , and scaled by **Ratio of installed thrust to uninstalled thrust**, **Maximum sea-level static thrust**, **Sea-level static thrust specific fuel consumption**, and **Fastest engine time constant at sea-level static**.

The Turbofan Engine System block icon displays the input and output units selected from the **Units** pop-up menu.

Turbofan Engine System

Dialog Box

Block Parameters: Turbofan Engine System

Turbofan Engine System (mask)
Implement a turbofan engine system. The turbofan engine system includes both engine and controller.

Throttle position can vary from zero to one, corresponding to no to full throttle. Altitude, initial thrust, and maximum thrust are entered in the same unit system as selected from the block for thrust and fuel flow output.

Parameters

Units: Metric

Initial thrust source: Internal

Initial thrust:
0

Maximum sea-level static thrust:
45000

Fastest engine time constant at sea-level static (sec):
1

Sea-level static thrust specific fuel consumption:
0.35

Ratio of installed thrust to uninstalled thrust:
0.9

OK Cancel Help Apply

Units

Specifies the input and output units:

	Altitude	Thrust	Fuel Flow
Metric	Meters	Newtons	Kilograms per second
English	Feet	Pound force	Pound mass per second

Initial thrust source

Specifies the source of initial thrust:

Internal	Use initial thrust value from mask dialog.
External	Use external input for initial thrust value.

Initial thrust

Initial value for thrust.

Maximum sea-level static thrust

Maximum thrust at sea-level and at Mach = 0.

Fastest engine time constant at sea-level static**Sea-level static thrust specific fuel consumption**

Thrust specific fuel consumption at sea-level, at Mach = 0, and at maximum thrust, in specified mass units per hour per specified thrust units.

Ratio of installed thrust to uninstalled thrust

Coefficient representing the loss in thrust due to engine installation.

Inputs and Outputs

The first input is throttle position. Throttle position can vary from zero to one, corresponding to no to full throttle.

The second input is Mach number.

The third input is altitude, in specified length units.

The first output is thrust, in specified force units.

The second output is fuel flow, in specified mass units per second.

Assumptions and Limitations

The atmosphere is at standard day conditions and an ideal gas.

Mach number is limited to less than 1.0.

This engine system is for indication purposes only. It is not meant to be used as a reference model.

References

“Aeronautical Vestpocket Handbook,” United Technologies Pratt & Whitney, August, 1986.

Raymer, D.P., “Aircraft Design: A Conceptual Approach,” AIAA Education Series, Washington, DC, 1989.

Hill, P.G, and Peterson, C.R., “Mechanics and Thermodynamics of Propulsion,” Addison-Wesley Publishing Company, Reading, MA, 1970.

Velocity Conversion

Purpose Convert from velocity units to desired velocity units

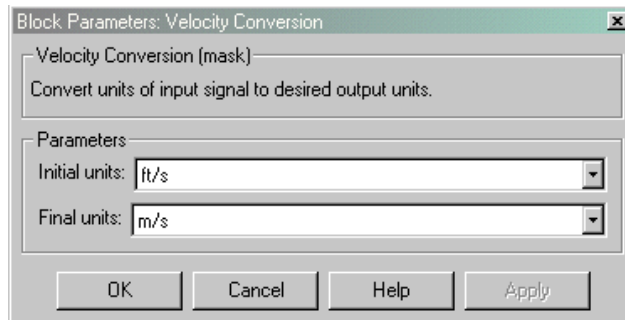
Library Transformations/Units

Description The Velocity Conversion block computes the conversion factor from specified input velocity units to specified output velocity units and applies the conversion factor to the input signal.



The Velocity Conversion block icon displays the input and output units selected from the **Initial units** and the **Final units** pop-up menus.

Dialog Box



Initial units

Specifies the input units.

Final units

Specifies the output units.

The following conversion units are available:

m/s	Meters per second
ft/s	Feet per second
km/s	Kilometers per second
in/s	Inches per second
km/h	Kilometers per hour

mph	Miles per hour
kts	Nautical miles per hour

Inputs and Outputs

The input is velocity in initial velocity units.

The output is velocity in final velocity units.

See Also

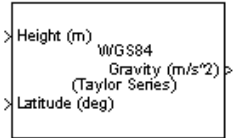
Acceleration Conversion
Angle Conversion
Angular Acceleration Conversion
Angular Velocity Conversion
Density Conversion
Force Conversion
Length Conversion
Mass Conversion
Pressure Conversion
Temperature Conversion

WGS84 Gravity Model

Purpose Implement the 1984 World Geodetic System (WGS84) representation of Earth's gravity

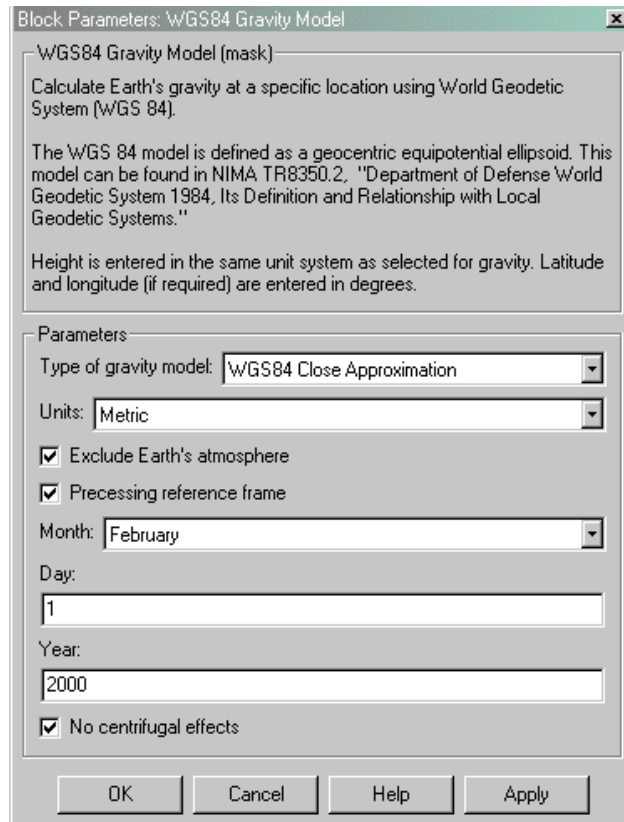
Library Environment/Gravity

Description The WGS84 Gravity Model block implements the mathematical representation of the geocentric equipotential ellipsoid of the World Geodetic System (WGS84). The block output is the Earth's gravity at a specific location. Gravity precision is controlled via **Type of gravity model**.



The WGS84 Gravity Model block icon displays the input and output units selected from the **Units** pop-up menu.

Dialog Box



Type of gravity model

Specifies the method to calculate gravity:

- WGS84 Taylor Series
- WGS84 Close Approximation
- WGS84 Exact

Units

Specifies the input and output units:

	Height	Gravity
Metric	Meters	Meters per second squared
English	Feet	Feet per second squared

Exclude Earth's atmosphere

When selected, the value for the Earth's gravitational field excludes the mass of the atmosphere.

If cleared, the value for the Earth's gravitational field includes the mass of the atmosphere.

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact.

Precessing reference frame

When selected, the angular velocity of the Earth is calculated using the International Astronomical Union (IAU) value of the Earth's angular velocity and the precession rate in right ascension. In order to obtain the precession rate in right ascension, Julian Centuries from Epoch J2000.0 is calculated using the dialog parameters of **Month**, **Day**, and **Year**.

If cleared, the angular velocity of the Earth used is the value of the standard Earth rotating at a constant angular velocity.

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact.

Month

Specifies the month used to calculate Julian Centuries from Epoch J2000.0.

WGS84 Gravity Model

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact and only when **Precessing reference frame** is selected.

Day

Specifies the day used to calculate Julian Centuries from Epoch J2000.0.

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact and only when **Precessing reference frame** is selected.

Year

Specifies the year used to calculate Julian Centuries from Epoch J2000.0. The year must be 2000 or greater.

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact and only when **Precessing reference frame** is selected.

No centrifugal effects

When selected, calculated gravity is based on pure attraction resulting from the normal gravitational potential.

If cleared, calculated gravity includes the centrifugal force resulting from the Earth's angular velocity.

This option is only available with **Type of gravity model** WGS84 Close Approximation or WGS84 Exact.

Inputs and Outputs

The first input is a vector containing altitudes in specified length units.

The second input is a vector containing latitudes in degrees.

The third input is a vector containing longitudes in degrees. This input is only available with **Type of Gravity Model** WGS84 Close Approximation or WGS84 Exact.

The output is a vector containing gravities in specified acceleration units.

Assumptions and Limitations

The WGS84 gravity calculations are based on the assumption of a geocentric equipotential ellipsoid of revolution. Since the gravity potential is assumed to be the same everywhere on the ellipsoid, there must be a specific theoretical gravity potential that can be uniquely determined from the four independent constants defining the ellipsoid.

Use of the WGS84 Taylor Series model should be limited to low geodetic heights. It will be sufficient near the surface when sub-microgal precision is not necessary. At medium and high geodetic heights, it will be less accurate.

Use of the WGS84 Close Approximation model should be limited to a geodetic height of 20000.0 m (approximately 65620.0 ft). Below this height, it will give results with sub-microgal precision.

References

[1] NIMA TR8350.2: "Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems."

Wind Shear Model

Purpose Calculate wind shear conditions

Library Environment/Wind

Description



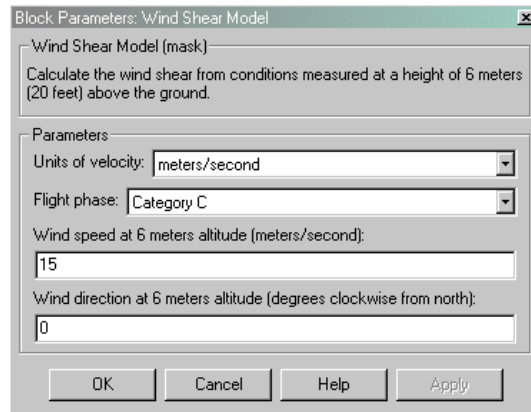
The Wind Shear Model block adds wind shear to the aerospace model. This implementation is based on the mathematical representation in the Military Specification MIL-F-8785C [1]. The magnitude of the wind shear is given by the following equation for the mean wind profile as a function of altitude and the measured wind speed at 20 feet (6 m) above the ground.

$$u_w = W_{20} \frac{\ln\left(\frac{h}{z_0}\right)}{\ln\left(\frac{20}{z_0}\right)}, \quad 3ft < h < 1000ft$$

where u_w is the mean wind speed, W_{20} is the measured wind speed at an altitude of 20 feet, h is the altitude, and z_0 is a constant equal to 0.15 feet for Category C flight phases and 2.0 feet for all other flight phases. Category C flight phases are defined in reference [1] to be terminal flight phases, which include takeoff, approach, and landing.

The resultant mean wind speed in the Earth-fixed axis frame is changed to body-fixed axis coordinates by multiplying by the direction cosine matrix (DCM) input to the block. The block output is the mean wind speed in the body-fixed axis.

Dialog Box



Units of velocity

Define the units of wind shear.

	Wind	Altitude
Meters/second	Meters/second	Meters
Feet/second	Feet/second	Feet
Knots	Knots	Feet

Flight Phase

Select flight phase:

- Category C - Terminal Flight Phases
- Other

Wind speed at 20 feet (or 6 m) altitude (meters/second, feet/second, or knots)

The measured wind speed at an altitude of 20 feet (6 m) above the ground.

Wind direction at 20 feet altitude (degrees clockwise from north)

The direction of the wind at an altitude of 20 feet (6 m), measured in degrees clockwise from the direction of the Earth x-axis (north). The wind direction is defined as the direction from which the wind is coming.

Wind Shear Model

Inputs and Outputs

The first input is altitude, in units selected.

The second input is a 3-by-3 direction cosine matrix.

The output is a 3-by-1 vector of the mean wind speed in the body axes frame, in the selected units.

Examples

See the `aeroblk_HL20.mdl` example included with the blockset.

References

Military Specification MIL-F-8785C, 5th November, 1980.

See Also

Discrete Wind Gust Model

Dryden Wind Turbulence Model

A

- Acceleration Conversion block 3-63
- Actuator library 3-2
- adding blocks 1-9
- aerolib 1-4
- Aerospace Blockset
 - accessing 1-5
 - getting started with 1-5
 - organization 1-3
 - overview 1-2
- Angle Conversion block 3-65
- Angular Acceleration Conversion block 3-67
- Angular Velocity Conversion block 3-69
- Animation library 3-2

B

- block diagrams
 - creating 1-9
- blocks
 - Acceleration Conversion 3-63
 - adding 1-9
 - Angle Conversion 3-65
 - Angular Acceleration Conversion 3-67
 - Angular Velocity Conversion 3-69
 - COESA Atmosphere Model 3-71
 - connecting 1-12
 - Density Conversion 3-73
 - Direction Cosine Matrix to Euler Angles 3-75
 - Direction Cosine Matrix to Quaternions 3-77
 - Discrete Wind Gust Model 3-79
 - Dryden Wind Turbulence Model 3-82
 - Equations of Motion 3-94
 - Euler Angles to Direction Cosine Matrix 3-90
 - Euler Angles to Quaternions 3-92
 - Force Conversion 3-98
 - Gain Scheduled Lead-Lag 3-100

- Incidence & Airspeed 3-101
- Interpolate Matrix(x) 3-102
- Interpolate Matrix(x,y) 3-104
- Interpolate Matrix(x,y,z) 3-106
- ISA Atmosphere Model 3-109
- Length Conversion 3-110
- Mass Conversion 3-112
- 1D Controller [A(v),B(v),C(v),D(v)] 3-11
- 1D Controller Blend $u=(1-L).K1.y+L.K2.y$ 3-14
- 1D Observer Form [A(v),B(v),C(v),F(v),H(v)] 3-17
- 1D Self-Conditioned [A(v),B(v),C(v),D(v)] 3-20
 - parameters for 1-12
- Pressure Conversion 3-114
- Quaternions to Direction Cosine Matrix 3-116
- Quaternions to Euler Angles 3-118
 - resizing 1-9
- Second Order Linear Actuator 3-120
- Second Order Nonlinear Actuator 3-122
- Self-Conditioned [A,B,C,D] 3-124
- 6DoF (Euler Angles) 3-55
- 6DoF (Quaternions) 3-59
- 6DoF Animation 3-53
- Temperature Conversion 3-129
- 3DoF Animation 3-50
- 3D Controller [A(v),B(v),C(v),D(v)] 3-39
- 3D Observer Form [A(v),B(v),C(v),F(v),H(v)] 3-42
- 3D Self-Conditioned [A(v),B(v),C(v),D(v)] 3-46
- 3x3 Cross Product 3-52
- Turbofan Engine System 3-131
- 2D Controller [A(v),B(v),C(v),D(v)] 3-24
- 2D Controller Blend 3-27
- 2D Observer Form [A(v),B(v),C(v),F(v),H(v)] 3-31

2D Self-Conditioned [A(v),B(v),C(v),D(v)]
3-35
Velocity Conversion 3-134
WGS84 Gravity Model 3-136
Wind Shear Model 3-140
building models 1-9

C

COESA Atmosphere Model block 3-71
connecting blocks and ports 1-12

D

defining models 1-9
Density Conversion block 3-73
Direction Cosine Matrix to Euler Angles block
3-75
Direction Cosine Matrix to Quaternions block
3-77
Discrete Wind Gust Model block 3-79
Dryden Wind Turbulence Model block 3-82

E

Environment/Atmosphere library 3-2
Environment/Gravity library 3-2
Environment/Wind library 3-2
Equations of Motion block 3-94
Equations of Motion/3DoF library 3-2
Equations of Motion/6DoF library 3-2
Euler Angles to Direction Cosine Matrix block
3-90
Euler Angles to Quaternions block 3-92

F

Force Conversion block 3-98

G

Gain Scheduled Lead-Lag block 3-100
GNC library 3-2

I

Incidence & Airspeed block 3-101
Interpolate Matrix(x) block 3-102
Interpolate Matrix(x,y) block 3-104
Interpolate Matrix(x,y,z) block 3-106
ISA Atmosphere Model block 3-109

L

Length Conversion block 3-110
libraries
Simulink 1-5

M

Mass Conversion block 3-112
M-files
running simulations from 1-20
missile guidance system 2-1
models
defining and building 1-9
simulating 1-19
Mux block 1-11

O

1D Controller [A(v),B(v),C(v),D(v)] block 3-11
1D Controller Blend $u=(1-L).K1.y+L.K2.y$ block
3-14

1D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$ block
3-17

1D Self-Conditioned $[A(v),B(v),C(v),D(v)]$ block
3-20

overview of Aerospace Blockset 1-2

P

parameters

definition of 1-12

setting 1-12

tuning 1-20

ports

connecting 1-12, 1-17

Pressure Conversion block 3-114

Propulsion library 3-2

Q

Quaternions to Direction Cosine Matrix block
3-116

Quaternions to Euler Angles block 3-118

R

resizing blocks 1-9

S

Scope block 1-11

Second Order Linear Actuator block

description 3-120

in tutorial 1-11

Second Order Nonlinear Actuator block 3-122

Self-Conditioned $[A,B,C,D]$ block 3-124

simulations

running 1-19

running from M-file 1-20

Simulink

accessing 1-5

learning 1-20

libraries 1-5

Library Browser 1-5

Library Window 1-7

simulink 1-5

6DoF (Euler Angles) block 3-55

6DoF (Quaternions) block 3-59

6DoF Animation block 3-53

T

Temperature Conversion block 3-129

3D Controller $[A(v),B(v),C(v),D(v)]$ block 3-39

3D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$ block
3-42

3D Self-Conditioned $[A(v),B(v),C(v),D(v)]$ block
3-46

3DoF Animation block 3-50

3x3 Cross Product block 3-52

Transformations/Axes library 3-2

Transformations/Units library 3-2

tuning parameters 1-20

Turbofan Engine System block 3-131

2D Controller $[A(v),B(v),C(v),D(v)]$ block 3-24

2D Controller Blend block 3-27

2D Observer Form $[A(v),B(v),C(v),F(v),H(v)]$ block
3-31

2D Self-Conditioned $[A(v),B(v),C(v),D(v)]$ block
3-35

U

using the Simulink Library Browser 1-5

using the Simulink Library Window 1-7

V

Velocity Conversion block 3-134

Virtual Reality Toolbox
used in visualization ix

W

WGS84 Gravity Model block 3-136

Wind Shear Model block 3-140